# FlexMeasures Documentation

***Release 0.2.4***

**Seita B.V.**

**Apr 02, 2021**

# CONTENTS

In a world with renewable energy, flexibility is crucial and valuable. Planning ahead allows flexible assets to serve the whole system with their flexibility, e.g. by shifting or curtailing energy use. This can also be profitable for their owners.

The FlexMeasures Platform is a tool for scheduling flexibility activations for energy assets. For this purpose, it performs three services:

- Monitoring of incoming measurements
- Forecasting of expected measurements
- Scheduling flexibility activations with custom optimisation

For more on FlexMeasures services, read *Services*. Or head right over to *Getting started*.

Using FlexMeasures benefits operators as well as asset owners, by allowing or automation, insight, autonomy and profit sharing. For more on benefits, consult *Benefits*.

FlexMeasures is compliant with the Universal Smart Energy Framework (USEF). Therefore, this documentation uses USEF terminology, e.g. for role definitions. The intended users of FlexMeasures are a Supplier (energy company) and its Prosumers (asset owners who have energy contracts with that Supplier). The platform operator of FlexMeasures can be an Aggregator.

CHAPTER

ONE

# GETTING STARTED

## 1.1 Quickstart

This section walks you through getting FlexMeasures to run with the least effort. We'll cover making a secret key, connecting a database and creating one user & one asset.

### 1.1.1 Install FlexMeasures

Install dependencies and the `flexmeasures` platform itself:

```
pip install flexmeasures
```

### 1.1.2 Make a secret key for sessions and password salts

Set a secret key which is used to sign user sessions and re-salt their passwords. The quickest way is with an environment variable, like this:

```
export SECRET_KEY=something-secret
```

(on Windows, use `set` instead of `export`)

This suffices for a quick start.

If you want to consistently use FlexMeasures, we recommend you add this setting to your config file at `~/.flexmeasures.cfg` and use a truly random string. Here is a Pythonic way to generate a good secret key:

```
python -c "import secrets; print(secrets.token_urlsafe())"
```

### 1.1.3 Configure environment

Set an environment variable to indicate in which environment you are operating (one out of development|testing|staging|production). We'll go with `development` here:

```
export FLASK_ENV=development
```

(on Windows, use `set` instead of `export`)

or:

```
echo "FLASK_ENV=development" >> .env
```

**Note:** The default is `production`, which will not work well on localhost due to SSL issues.

### 1.1.4 Preparing the time series database

- Make sure you have a Postgres (Version 9+) database for FlexMeasures to use. See *Handling data* (section "Getting ready to use") for instructions on this.
- Tell `flexmeasures` about it:

  ```
  export SQLALCHEMY_DATABASE_URI="postgresql://<user>:<password>@<host-
  ↪address>[:<port>]/<db>"
  ```

  If you install this on localhost, `host-address` is `127.0.0.1` and the port can be left out. (on Windows, use `set` instead of `export`)
- Create the Postgres DB structure for FlexMeasures:

  ```
  flexmeasures db upgrade
  ```

This suffices for a quick start.

**Note:** For a more permanent configuration, you can create your FlexMeasures configuration file at `~/.flexmeasures.cfg` and add this:

```
SQLALCHEMY_DATABASE_URI="postgresql://<user>:<password>@<host-address>[:<port>]/<db>"
```

### 1.1.5 Add a user

FlexMeasures is a web-based platform, so we need a user account:

```
flexmeasures add user --username <your-username> --email <your-email-address> --
↪roles=admin
```

- This will ask you to set a password for the user.
- Giving the first user the `admin` role is probably what you want.

### 1.1.6 Add structure

Populate the database with some standard energy asset types, weather sensor types and a dummy market:

```
flexmeasures add structure
```

### 1.1.7 Add your first weather sensor

Weather plays a role for almost all use cases. FlexMeasures supports a few weather sensor types out of the box ("temperature", "wind_speed" and "radiation"), but you need to decide which ones you need and where they are located. Let's use the `flexmeasures` *CLI Commands* to add one:

```
flexmeasures add weather-sensor --name "my rooftop thermometer" --weather-sensor-type-
↪name temperature --unit °C --event-resolution 15 --latitude 33 --longitude 2.4
```

### 1.1.8 Add your first asset

There are three ways to add assets:

Head over to `http://localhost:5000/assets` and add a new asset there.

Or, use the `flexmeasures` *CLI Commands*:

```
flexmeasures add asset --name "my basement battery pack" --asset-type-name battery --
↪capacity-in-MW 30 --event-resolution 2 --latitude 65 --longitude 123.76 --owner-id 1
```

Here, I left out the `--market-id` parameter, because in this quickstart scenario I'm fine with the dummy market created with `flexmeasures add structure` above. For the ownership, I got my user ID from the output of `flexmeasures add user` above, or I can browse to FlexMeasures' user listing and hover over my username.

Finally, you can also use the POST /api/v2_0/assets endpoint in the FlexMeasures API to create an asset.

### 1.1.9 Run FlexMeasures

It's finally time to start running FlexMeasures:

```
flexmeasures run
```

(This might print some warnings, see the next section where we go into more detail)

---

**Note:** In a production context, you shouldn't run a script - hand the `app` object to a WSGI process, as your platform of choice describes. Often, that requires a WSGI script. We provide an example WSGI script in continuous_integration.

---

You can visit `http://localhost:5000` now to see if the app's UI works. When you see the dashboard, the map will not work. For that, you'll need to get your *MAPBOX_ACCESS_TOKEN* and add it to your config file.

### 1.1.10 Add data

You can use the POST /api/v2_0/postMeterData endpoint in the FlexMeasures API to send meter data.

---

**Note:** issue 56 should create a CLI function for adding a lot of data at once, from a CSV dataset.

---

Also, you can add forecasts for your meter data with the `flexmeasures add` command, here is an example:

```
flexmeasures add forecasts --from-date 2020-03-08 --to-date 2020-04-08 --asset-type␣
↪Asset --asset my-solar-panel
```

**Note:** You can also use the API to send forecast data.

## 1.2 Other settings, for full functionality

### 1.2.1 Set mail settings

For FlexMeasures to be able to send email to users (e.g. for resetting passwords), you need an email account which can do that (e.g. GMail). Set the MAIL_* settings in your configuration, see *Mail*.

### 1.2.2 Install an LP solver

For planning balancing actions, the FlexMeasures platform uses a linear program solver. Currently that is the Cbc solver. See *FLEXMEASURES_LP_SOLVER* if you want to change to a different solver.

Installing Cbc can be done on Unix via:

```
apt-get install coinor-cbc
```

(also available in different popular package managers).

We provide a script for installing from source (without requiring `sudo` rights) in continuous_integration.

More information (e.g. for installing on Windows) on the Cbc website.

### 1.2.3 Start collecting weather data

To collect weather measurements and forecasts from the DarkSky API, there is a task you could run periodically, probably once per hour. Here is an example:

```
flexmeasures add external-weather-forecasts --location 33.4366,126.5269 --store-in-db
```

**Note:** DarkSky is not handing out tokens anymore, as they have been bought by Apple (see issue 3).

### 1.2.4 Preparing the job queue database and start workers

To let FlexMeasures queue forecasting and scheduling jobs, install a Redis server and configure access to it within FlexMeasures' config file (see above). You can find the necessary settings in *Redis*.

Then run one worker for each kind of job (in a separate terminal):

```
flexmeasures run-worker --queue forecasting
flexmeasures run-worker --queue scheduling
```

You can also clear the job queues:

```
flexmeasures clear-queue --queue forecasting
flexmeasures clear-queue --queue scheduling
```

When the main FlexMeasures process runs (e.g. by `flexmeasures run`), the queues of forecasting and scheduling jobs can be visited at `http://localhost:5000/tasks/forecasting` and `http://localhost:5000/tasks/schedules`, respectively (by admins).

When forecasts and schedules have been generated, they should be visible at `http://localhost:5000/analytics`. You can also access forecasts via the FlexMeasures API at GET /api/v2_0/getPrognosis, and schedules via GET /api/v2_0/getDeviceMessage.

CHAPTER

TWO

# CONFIGURATION

The following configurations are used by FlexMeasures.

Required settings (e.g. postgres db) are marked with a double star (**). To enable easier quickstart tutorials, these settings can be set by env vars. Recommended settings (e.g. mail, redis) are marked by one star (*).

---

**Note:** FlexMeasures is best configured via a config file. The config file for FlexMeasures can be placed in one of two locations:

---

- in the user's home directory (e.g. `~/.flexmeasures.cfg` on Unix). In this case, note the dot at the beginning of the filename!
- in the app's instance directory (e.g. `/path/to/your/flexmeasures/code/instance/flexmeasures.cfg`). The path to that instance directory is shown to you by running flexmeasures (e.g. `flexmeasures run`) with required settings missing or otherwise by running `flexmeasures shell`.

## 2.1 Basic functionality

### 2.1.1 LOGGING_LEVEL

Level above which log messages are added to the log file. See the `logging` package in the Python standard library.

Default: `logging.WARNING`

### 2.1.2 FLEXMEASURES_MODE

The mode in which FlexMeasures is being run, e.g. "demo" or "play". This is used to turn on certain extra behaviours.

Default: `""`

### 2.1.3 FLEXMEASURES_LP_SOLVER

The command to run the scheduling solver. This is the executable command which FlexMeasures calls via the pyomo library. Other values might be `cplex` or `glpk`. Consult their documentation to learn more.

Default: `"cbc"`

### 2.1.4 FLEXMEASURES_HOSTS_AND_AUTH_START

Configuration used for entity addressing. This contains the domain on which FlexMeasures runs and the first month when the domain was under the current owner's administration.

Default: `{"flexmeasures.io": "2021-01"}`

### 2.1.5 FLEXMEASURES_DB_BACKUP_PATH

Relative path to the folder where database backups are stored if that feature is being used.

Default: `"migrations/dumps"`

### 2.1.6 FLEXMEASURES_PROFILE_REQUESTS

Whether to turn on a feature which times requests made through FlexMeasures. Interesting for developers.

Default: `False`

## 2.2 UI

### 2.2.1 FLEXMEASURES_PLATFORM_NAME

Name being used in headings

Default: `"FlexMeasures"`

### 2.2.2 FLEXMEASURES_HIDE_NAN_IN_UI

Whether to hide the word "nan" if any value in metrics tables is `NaN`.

Default: `False`

### 2.2.3 RQ_DASHBOARD_POLL_INTERVAL

Interval in which viewing the queues dashboard refreshes itself, in milliseconds.

Default: `3000` (3 seconds)

## 2.3 Timing

### 2.3.1 FLEXMEASURES_TIMEZONE

Timezone in which the platform operates. This is useful when datetimes are being localized.

Default: `"Asia/Seoul"`

### 2.3.2 FLEXMEASURES_PLANNING_TTL

Time to live for UDI event ids of successful scheduling jobs. Set a negative timedelta to persist forever.

Default: `timedelta(days=7)`

### 2.3.3 FLEXMEASURES_PLANNING_HORIZON

The horizon to use when making schedules.

Default: `timedelta(hours=2 * 24)`

## 2.4 Tokens

### 2.4.1 DARK_SKY_API_KEY

Token for accessing the DarkSky weather forecasting service.

---

**Note:** DarkSky will soon become non-public (Aug 1, 2021), so they are not giving out new tokens. We'll use another service soon (see this issue). This is unfortunate. In the meantime, if you can't find anybody lending their token, consider posting weather forecasts to the FlexMeasures database yourself.

---

Default: `None`

### 2.4.2 MAPBOX_ACCESS_TOKEN

Token for accessing the mapbox API (for displaying maps on the dashboard and asset pages). You can learn how to obtain one here

Default: `None`

### 2.4.3 FLEXMEASURES_TASK_CHECK_AUTH_TOKEN

Token which external services can use to check on the status of recurring tasks within FlexMeasures.

Default: `None`

## 2.5 SQLAlchemy

This is only a selection of the most important settings. See the Flask-SQLAlchemy Docs for all possibilities.

### 2.5.1 SQLALCHEMY_DATABASE_URI (**)

Connection string to the postgres database, format: `postgresql://<user>:<password>@<host-address>[:<port>]/<db>`

Default: `None`

### 2.5.2 SQLALCHEMY_ENGINE_OPTIONS

Configuration of the SQLAlchemy engine.

Default:

```
{
    "pool_recycle": 299,
    "pool_pre_ping": True,
    "connect_args": {"options": "-c timezone=utc"},
}
```

## 2.6 Security

This is only a selection of the most important settings. See the Flask-Security Docs as well as the Flask-CORS docs for all possibilities.

### 2.6.1 SECRET_KEY (**)

Used to sign user sessions and also as extra salt (a.k.a. pepper) for password salting if `SECURITY_PASSWORD_SALT` is not set. This is actually part of Flask - but is also used by Flask-Security to sign all tokens.

It is critical this is set to a strong value. For python3 consider using: `secrets.token_urlsafe()` You can also set this in a file (which some Flask tutorials advise).

**Note:** Leave this setting set to `None` to get more instructions when you attempt to run FlexMeasures.

Default: `None`

### 2.6.2 SECURITY_PASSWORD_SALT

Extra password salt (a.k.a. pepper)

Default: `None` (falls back to `SECRET_KEY`)

### 2.6.3 SECURITY_TOKEN_AUTHENTICATION_HEADER

Name of the header which carries the auth bearer token in API requests.

Default: `Authorization`

### 2.6.4 SECURITY_TOKEN_MAX_AGE

Maximal age of security tokens in seconds.

Default: `60 * 60 * 6` (six hours)

### 2.6.5 SECURITY_TRACKABLE

Wether to track user statistics. Turning this on requires certain user fields. We do not use this feature, but we do track number of logins.

Default: `False`

### 2.6.6 CORS_ORIGINS

Allowed cross-origins. Set to "*" to allow all. For development (e.g. javascript on localhost) you might use "null" in this list.

Default: `[]`

### 2.6.7 CORS_RESOURCES:

FlexMeasures resources which get cors protection. This can be a regex, a list of them or dict with all possible options.

Default: `[r"/api/*"]`

### 2.6.8 CORS_SUPPORTS_CREDENTIALS

Allows users to make authenticated requests. If true, injects the Access-Control-Allow-Credentials header in responses. This allows cookies and credentials to be submitted across domains.

---

**Note:** This option cannot be used in conjunction with a "*" origin.

---

Default: `True`

## 2.7 Mail

For FlexMeasures to be able to send email to users (e.g. for resetting passwords), you need an email account which can do that (e.g. GMail).

This is only a selection of the most important settings. See the Flask-Mail Docs for others.

### 2.7.1 MAIL_SERVER (*)

Email name server domain.

Default: `"localhost"`

### 2.7.2 MAIL_PORT (*)

SMTP port of the mail server.

Default: `25`

### 2.7.3 MAIL_USE_TLS

Whether to use TLS.

Default: `False`

### 2.7.4 MAIL_USE_SSL

Whether to use SSL.

Default: `False`

### 2.7.5 MAIL_USERNAME (*)

Login name of the mail system user.

Default: `None`

### 2.7.6 MAIL_DEFAULT_SENDER (*)

Tuple of shown name of sender and their email address.

Default:

```
(
    "FlexMeasures",
    "no-reply@example.com",
)
```

### 2.7.7 MAIL_PASSWORD

Password of mail system user.

Default: `None`

## 2.8 Redis

FlexMeasures uses the Redis database to support our forecasting and scheduling job queues.

### 2.8.1 FLEXMEASURES_REDIS_URL (*)

URL of redis server.

Default: `"localhost"`

### 2.8.2 FLEXMEASURES_REDIS_PORT (*)

Port of redis server.

Default: `6379`

### 2.8.3 FLEXMEASURES_REDIS_DB_NR (*)

Number of the redis database to use (Redis per default has 16 databases, nubered 0-15)

Default: `0`

### 2.8.4 FLEXMEASURES_REDIS_PASSWORD (*)

Password of the redis server.

Default: `None`

## 2.9 Demonstrations

### 2.9.1 FLEXMEASURES_PUBLIC_DEMO_CREDENTIALS

When `FLEXMEASURES_MODE=demo`, this can hold login credentials (demo user email and password, e.g. `("demo at seita.nl", "flexdemo")`), so anyone can log in and try out the platform.

Default: `None`

### 2.9.2 FLEXMEASURES_DEMO_YEAR

When `FLEXMEASURES_MODE=demo`, this setting can be used to make the FlexMeasures platform select data from a specific year (e.g. 2015), so that old imported data can be demoed as if it were current

Default: `None`

### 2.9.3 FLEXMEASURES_SHOW_CONTROL_UI

The control page is still mocked, so this setting controls if it is to be shown.

Default: `False`

CHAPTER

THREE

# FLEXMEASURES CHANGELOG

## 3.1 v0.2.4 | April 2, 2021

### 3.1.1 New features

- FlexMeasures can be installed with `pip` and its CLI commands can be run with `flexmeasures` [see PR #54]
- Optionally setting recording time when posting data [see PR #41]
- Add assets and weather sensors with CLI commands [see PR #74]

**Note:** Read more on these features on the FlexMeasures blog.

### 3.1.2 Bugfixes

- Show screenshots in documentation and add some missing content [see PR #60]
- Documentation listed 2.0 API endpoints twice [see PR #59]
- Better xrange and title if only schedules are plotted [see PR #67]
- User page did not list number of assets correctly [see PR #64]
- Missing *postPrognosis* endpoint for >1.0 API blueprints [part of PR #41]

### 3.1.3 Infrastructure / Support

- Added concept pages to documentation [see PR #65]
- Dump and restore postgres database as CLI commands [see PR #68]
- Improved installation tutorial as part of [PR #54]
- Moved developer docs from Readmes into the main documentation [see PR #73]
- Ensured unique sensor ids for all sensors [see PR #70 and (fix) PR #77]

## 3.2 v0.2.3 | February 27, 2021

### 3.2.1 New features

- Power charts available via the API [see PR #39]
- User management via the API [see PR #25]
- Better visibility of asset icons on maps [see PR #30]

**Note:** Read more on these features on the FlexMeasures blog.

### 3.2.2 Bugfixes

- Fix maps on new asset page (update MapBox lib) [see PR #27]
- Some asset links were broken [see PR #20]
- Password reset link on account page was broken [see PR #23]

### 3.2.3 Infrastructure / Support

- CI via Github Actions [see PR #1]
- Integration with timely beliefs lib: Sensors [see PR #13]
- Apache 2.0 license [see PR #16]
- Load js & css from CDN [see PR #21]
- Start using marshmallow for input validation, also introducing `HTTP status 422` in the API [see PR #25]
- Replace `solarpy` with `pvlib` (due to license conflict) [see PR #16]
- Stop supporting the creation of new users on asset creation (to reduce complexity) [see PR #36]

CHAPTER

FOUR

# SERVICES

## 4.1 Monitoring

The FlexMeasures platform continuously reads in meter data from your assets. To assist your maintenance, it can alert you to situations which need your attention:

- Breaches of thresholds (protect devices)
- Data gaps & strange outliers (assure data quality)
- Idle processes / leaks (minimise waste)

## 4.2 Forecasting

The FlexMeasures platform continuously creates forecasts for the rest of day.

All relevant data should be forecasted:

- Energy assets
- Weather data
- Market prices

## 4.3 Scheduling

The FlexMeasures platform optimises schedules for your flexible assets. This is where energy flexibility is valorised!

Examples are:

- Charging schedules of batteries
- Heat pumps management
- Buffering of machinery

The goals can be maximal cost savings, maximal usage of solar power or stable energy supply for the most crucial consumers.

CHAPTER

FIVE

# BENEFITS

## 5.1 Automation

FlexMeasures provides decision-making support so that the platform operator can schedule flexibility activations. It forecasts the state of assets and proposes the best flexibility activations (shifting or curtailment) for future periods. This is done with modern forecasting and scheduling intelligence.

## 5.2 Insight

Both platform operator and asset owners can monitor the assets - past and current states as well as forecasts are displayed numerically in plots and tables. Activations of flexibility which were ordered in the past can be reviewed. Proposed and scheduled flexibility activations show their expected effects (on imbalance as well as on financial returns).

## 5.3 Autonomy

The companies connected to FlexMeasures only give up as much control as necessary. The asset owners still control the main behaviour of their assets. The owners allow the platform operator to schedule flexibility activations within limits they can set.

Also the platform operator stays in charge: They can choose to approve all proposed flexibility activations manually or to let FlexMeasures automatically schedule them. As FlexMeasures is open source, they can choose to host it themselves or let a third party (like Seita BV) do that.

## 5.4 Profit sharing

The platform operator (as ESCo or Aggregator) and asset owners can share the profit made from flexibility activations between them. FlexMeasures plans on providing basic accounting for this.

**Note:** Read more on flexibility opportunities and activations, as well as profit sharing on *Benefits from energy flexibility*

CHAPTER

SIX

# BENEFITS FROM ENERGY FLEXIBILITY

FlexMeasures was created so that the value of energy flexibility can be realised. This will make energy cheaper to use, and can also reduce $CO_2$ emissions. Here, we define a few terms around this idea, which come up in other parts of this documentation.



## 6.1 Flexibility opportunities and activation

### 6.1.1 Opportunities

In an energy system with flexible energy assets present (e.g. batteries, heating/cooling), there are opportunities to profit from the availability and activation of their flexibility.

Energy flexibility can come from the ability to store energy ("storage"), to delay (or advance) planned consumption or production ("shifting"), the ability to lower production ("curtailment"), or the ability to increase or decrease consumption ("demand response") — see *Types of flexibility* for a deeper discussion.

Under a given incentive, this flexibility represents an opportunity to profit by scheduling consumption or production differently than originally planned. Within FlexMeasures, flexibility is represented as the difference between a suggested schedule and a given baseline. By default, a baseline is determined by our own forecasts.

Opportunities are expressed with respect to given economical and ecological incentives. For example, a suggested schedule may represent an opportunity to save X EUR and Y tonnes of $CO_2$.

### 6.1.2 Activation

The activation of flexibility usually happens in a context of incentives. Often, that context is a market. We recommend the USEF white paper on the flexibility value chain for an excellent introduction of who can benefit from energy flexibility and how it can be delivered. The high-level takeaways are these:

- the value of flexibility flows back to Prosumers along a chain of roles involved in the activation of their flexibility: the **Flexibility Value Chain**.
- a portfolio of flexible assets (and even individual assets) may provide services in multiple contexts in the same period: **value stacking**.
- **Explicit demand-side flexibility** services involve Aggregators, while **implicit demand-side flexibility** services involve Energy Service Companies (ESCos).
- Many remuneration components for flexibility services requires the determination of a baseline according to some **baseline methodology**.
- Both availability and activation of flexibility have value.

The overall value (from availability and activation of flexibility), and how this value is shared amongst stakeholders in the various roles in the Flexibility Value Chain, can be accounted for by the platform operator. We talk more about this in *Profits of flexibility activation*.

## 6.2 An example: the balancing market

An example of a market on which flexibility can be activated is the balancing market, which is meant to bring the grid frequency back to a target level within a matter of minutes. Consider the aforementioned differences between suggested schedules and a given baseline. In the context of the balancing market, differences indicating an increase in production or a decrease in consumption on activation both result in an increasing grid frequency (back towards the target frequency).

The balancing market pays for such services, and they are often referred to as *"up-regulation"*. It works the other way around, too: differences indicating a decrease in production or an increase in consumption both result in a decreasing grid frequency (*"down-regulation"*).

## 6.3 Types of flexibility

The FlexMeasures platform distinguishes between different types of flexibility. We explain them here in more detail, together with examples.

### 6.3.1 Curtailment

Curtailment happens when an asset temporarily lowers or stops its production or consumption. A defining feature of curtailment is that total production or consumption decreases when this this flexibility is activated.

- A typical example of curtailing production is when a wind turbine adjusts the pitch angle of its blades to decrease the generator torque.
- An example of curtailing consumption is load shedding of energy intensive industries.

Curtailment offers may specify some freedom in terms of how much energy can be curtailed. In these cases, the user can select the energy volume (in MWh) to be ordered, within constraints set by the relevant Prosumer. The net effect of a curtailment action is also measured in terms of an energy volume (see the flexibility metrics in the *Portfolio overview* page).

Note that the volume ordered is not necessarily equal to the volume curtailed: the ordered volume relates only to the selected time window, while the curtailed volume may include volumes outside of the selected time window. For example, an asset that runs an all-or-nothing consumption process of 2 hours can be ordered to curtail consumption for 1 hour, but will in effect stop the entire process. In this case, the curtailed volume will be higher than the ordered volume, and the platform will take into account the total expected curtailment in its calculations.

### 6.3.2 Shifting

Shifting happens when an asset delays or advances its energy production or consumption. A defining feature of shifting is that total production or consumption remains the same when this flexibility is activated.

- An example of delaying consumption is when a charging station postpones the charging process of an electric vehicle.
- An example of advancing consumption is when a cooling unit starts to cool before the upper temperature bound was reached (pre-cooling).

Shifting offers may specify some freedom in terms of how much energy can be shifted. In these cases, the user can select the energy volume (in MWh) to be ordered, within constraints set by the relevant Prosumer. This energy volume represents how much energy is shifting into or out of the selected time window. The net effect of a shifting action is measured in terms of an energy-time volume (see the flexibility metrics in the *Portfolio overview* page). This volume is a multiplication of the energy volume being shifted and the duration of that shift.

## 6.4 Profits of flexibility activation

The realised value from activating flexibility has to be computed and accounted for. Both of these activities depend on the context in which FlexMeasures is being used, and we expect that it will often have to be implemented in a custom manner (much as the actual scheduling optimisation).

**Todo:** Making it possible to configure custom scheduling and value accounting is on the roadmap for FlexMeasures.

### 6.4.1 Computing value

The computation of the value is what drives the scheduling optimisation. This value is usually monetary, and in that case there should be some form of market configured. This can be a constant or time-of-use tariff, or a real market. However, there are other possibilities, for instance if the optimisation goal is to minimise $CO_2$ emissions. Then, the realised value is avoided $CO_2$, which nowadays has an assumed value, e.g. in the EU ETS carbon market.

### 6.4.2 Accounting / Sharing value

The realisation of payments is outside of the scope of FlexMeasures, but it can provide the accounting to enable them (as was said above, this is usually a part of the optimisation problem formulation).

However, next to fuelling algorithmic optimisation, the way that the value of energy flexibility is shared among the stakeholders will also be an important driver for project participation. Accounting plays an important role here.

There are different roles in a modern smart energy system (e.g. “Prosumer”, “DSO”, Aggregator”, “ESCo”), and they all enjoy the benefits of flexibility in different ways (see for example this resource for more details).

In our opinion, the only way to successful implementation of energy flexibility is if profits are shared between these stakeholders. This assumes contractual relationships. Use cases which FlexMeasures can support well are the following relationships:

- between Aggregator and Prosumer, where the Aggregator sells the balancing power to a third party and shares the profits with the Prosumer according to some contracted method for profit sharing. In this case the stated costs and revenues for the Prosumer may be after deducting the Aggregator fee (which typically include price components per flex activation and price components per unit of time, but may include arbitrarily complex price components).
- between ESCo and Prosumer, where the ESCo advises the Prosumer to optimise against e.g. dynamic prices. Likewise, stated numbers may be after deducting the ESCo fee.

FlexMeasures can take these intricacies into account if a custom optimisation algorithm is plugged in to model them.

Alternatively, we can assume that all profit from activating flexibility goes to the Prosumer, or simply report the profits before sharing (and before deducting any service fees).

CHAPTER

SEVEN

# ALGORITHMS



## 7.1 Forecasting

Forecasting algorithms are used by FlexMeasures to assess the likelihood of future consumption/production and prices. Weather forecasting is included in the platform, but is usually not the result of an internal algorithm (weather forecast services are being used by import scripts, e.g. with this tool).

FlexMeasures uses linear regression and falls back to naive forecasting of the last known value if errors happen. What might be even more important than the type of algorithm is the features handed to the model — lagged values (e.g. value of the same time yesterday) and regressors (e.g. wind speed prediction to forecast wind power production).

The performance of our algorithms is indicated by the mean absolute error (MAE) and the weighted absolute percentage error (WAPE). Power profiles on an asset level often include zero values, such that the mean absolute percentage error (MAPE), a common statistical measure of forecasting accuracy, is undefined. For such profiles, it is more useful to report the WAPE, which is also known as the volume weighted MAPE. The MAE of a power profile gives an indication of the size of the uncertainty in consumption and production. This allows the user to compare an asset's predictability to its flexibility, i.e. to the size of possible flexibility activations.

Example benchmarks per asset type are listed in the table below for various assets and forecasting horizons. FlexMeasures updates the benchmarks automatically for the data currently selected by the user. Amongst other factors, accuracy is influenced by:

- The chosen metric (see below)
- Resolution of the forecast
- Horizon of the forecast

- Asset type
- Location / Weather conditions
- Level of aggregation

Accuracies in the table are reported as 1 minus WAPE, which can be interpreted as follows:

- 100% accuracy denotes that all values are correct.
- 50% accuracy denotes that, on average, the values are wrong by half of the reference value.
- 0% accuracy denotes that, on average, the values are wrong by exactly the reference value (i.e. zeros or twice the reference value).
- negative accuracy denotes that, on average, the values are off-the-chart wrong (by more than the reference value itself).

| Asset | Building | Charge Points | Solar | Wind (offshore) | Day-ahead market |
|---|---|---|---|---|---|
| Average power per asset | 204 W | 75 W | 140 W | 518 W | |
| 1 - WAPE (1 hour ahead) | 93.4 % | 87.6 % | 95.2 % | 81.6 % | 88.0 % |
| 1 - WAPE (6 hours ahead) | 92.6 % | 73.0 % | 83.7 % | 73.8 % | 81.9 % |
| 1 - WAPE (24 hours ahead) | 92.4 % | 65.2 % | 46.1 % | 60.1 % | 81.4 % |
| 1 - WAPE (48 hours ahead) | 92.1 % | 63.7 % | 43.3 % | 56.9 % | 72.3 % |

Defaults:

- The application uses an ordinary least squares auto-regressive model with external variables.
- Lagged outcome variables are selected based on the periodicity of the asset (e.g. daily and/or weekly).
- Common external variables are weather forecasts of temperature, wind speed and irradiation.
- Timeseries data with frequent zero values are transformed using a customised Box-Cox transformation.
- To avoid over-fitting, cross-validation is used.
- Before fitting, explicit annotations of expert knowledge to the model (like the definition of asset-specific seasonality and special time events) are possible.
- The model is currently fit each day for each asset and for each horizon.

Improvements:

- Most assets have yearly seasonality (e.g. wind, solar) and therefore forecasts would benefit from >= 2 years of history.

## 7.2 Scheduling

Given price conditions or other conditions of relevance, a scheduling algorithm is used by the Aggregator (in case of explicit DR) or by the Energy Service Company (in case of implicit DR) to form a recommended schedule for the Prosumer's flexible assets.

### 7.2.1 Storage devices

So far, FlexMeasures provides algorithms for storage — for batteries (e.g. home batteries or EVs) and car charging stations. We thus cover the asset types "battery", "one-way_evse" and "two-way_evse".

These algorithms schedule the storage assets based directly on the latest beliefs regarding market prices, within the specified time window. They are mixed integer linear programs, which are configured in FlexMeasures and then handed to a dedicated solver.

For all scheduling algorithms, a starting state of charge (SOC) as well as a set of SOC targets can be given. If no SOC is available, we set the starting SOC to 0.

Also, per default we incentivise the algorithms to prefer scheduling charging now rather than later, and discharging later rather than now. We achieve this by adding a tiny artificial price slope. We penalise the future with at most 1 per thousand times the price spread. This behaviour can be turned off with the *prefer_charging_sooner* parameter set to *False*.

---

**Note:** For the resulting consumption schedule, consumption is defined as positive values.

---

## 7.3 Possible future work on algorithms

Enabling more algorithmic expression in FlexMeasures is crucial. This are a few ideas for future work. Some of them are excellent topics for Bachelor or Master theses. so get in touch if that is of interest to you.

### 7.3.1 More configurable forecasting

On the roadmap for FlexMeasures is to make features easier to configure, especially regressors. Furthermore, we plan to add more types of forecasting algorithms, like random forest or even LSTM.

### 7.3.2 Other optimisation goals for scheduling

Next to market prices, optimisation goals like reduced $CO_2$ emissions are sometimes required. There are multiple ways to measure this, e.g. against the $CO_2$ mix in the grid, or the use of fossil fuels.

### 7.3.3 Scheduling of other flexible asset types

Next to storage, there are other interesting flexible assets which can require specific implementations. For shifting, there are heat pumps and other buffers. For curtailment, there are wind turbines and solar panels.

---

**Note:** See *Types of flexibility* for more info on shifting and curtailment.

---

### 7.3.4 Broker algorithm

A broker algorithm is used by the Aggregator to analyse flexibility in the Supplier's portfolio of assets, and to suggest the most valuable flexibility activations to take for each time slot. The differences to single-asset scheduling are that these activations are based on a helicopter perspective (the Aggregator optimises a portfolio, not a single asset) and that the flexibility offers are presented to the Supplier in the form of an order book.

### 7.3.5 Trading algorithm

A trading algorithm is used to assist the Supplier with its decision-making across time slots, based on the order books made by the broker (see above). The algorithm suggests which offers should be accepted next, and the Supplier may automate its decision-making by letting the algorithm place orders on its behalf.

A default approach would be a myopic greedy strategy — order all flexibility opportunities with a positive expected value in the first available timeslot, then those in the second available timeslot, and so on.

CHAPTER

EIGHT

# SECURITY ASPECTS

## 8.1 Data

There are two types of data on FlexMeasures servers - files (e.g. source code, images) and data in a database (e.g. user data and time series for energy consumption/generation or weather).

- Files are stored on EBS volumes on Amazon Web Services. These are shared with other customers of Amazon, but protected from them by Linux's chroot system – each user can see only the files in their own section of the disk.
- Database data is stored in PostgresDB instances which are not shared with other Amazon customers. They are password-protected.
- Finally, The application communicates all data with HTTPS, the Hypertext Transfer Protocol encrypted by Transport Layer Security. This is used even if the application is accessed via *http://*.

## 8.2 Authentication and Authorisation

*Authentication* is the system by which users tell the FlexMeasures platform that they are who they claim they are. This involves a username/password combination ("credentials") or an access token.

- No user passwords are stored in clear text on any server - the FlexMeasures platform only stores the hashed passwords (encrypted with the bcrypt hashing algorithm). If an attacker steals these password hashes, they cannot compute the passwords from them in a practical amount of time.
- Access tokens are used so that the sending of usernames and passwords is limited (even if they are encrypted via https, see above) when dealing with the part of the FlexMeasures platform which sees the most traffic: the API functionality. Tokens thus have use cases for some scenarios, where developers want to treat authentication information with a little less care than credentials should be treated with, e.g. sharing among computers. However, they also expire fast, which is a common industry practice (by making them short-lived and requiring refresh, FlexMeasures limits the time an attacker can abuse a stolen token). At the moment, the access tokens on FlexMeasures platform expire after six hours. Access tokens are encrypted and validated with the sha256_crypt algorithm, and the functionality to expire tokens is realised by storing the seconds since January 1, 2011 in the token. The maximum age of access tokens in FlexMeasures can be altered by setting the env variable *SECURITY_TOKEN_MAX_AGE* to the number of seconds after which tokens should expire.

*Authorisation* is the system by which the FlexMeasures platform decides whether an authenticated user can access a feature. For instance, many features are reserved for administrators, others for Prosumers (the owner of assets).

- This is achieved via *roles*. Each user has at least one role, but could have several, as well.
- Roles cannot be edited via the UI at the moment. They are decided when a user is created.

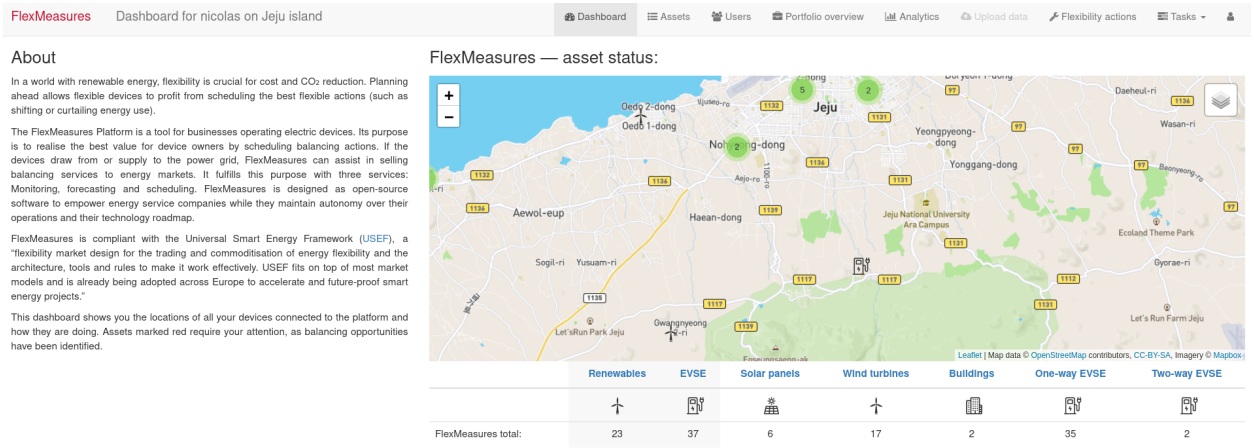
CHAPTER

NINE

# DASHBOARD

The dashboard shows where the user's assets are located and how many different asset types are connected to the platform. The view serves to quickly identify the status of assets, such as whether there are upcoming opportunities to valorise on flexibility activations. In particular, the page contains:

- *Interactive map of assets*
- *Summary of asset types*




## 9.1 Interactive map of assets

The map shows all of the user's assets with icons for each asset type. Clicking on an asset allows the user to see its current state (e.g. latest measurement of wind power production) and to navigate to the *Client analytics* page to see more details, for instance forecasts.

## 9.2 Summary of asset types

The summary below the map lists all asset types that the user has hooked up to the platform and how many of each there are. Clicking on the asset type name leads to the *Client analytics* page, where data is shown aggregated for that asset type.

CHAPTER

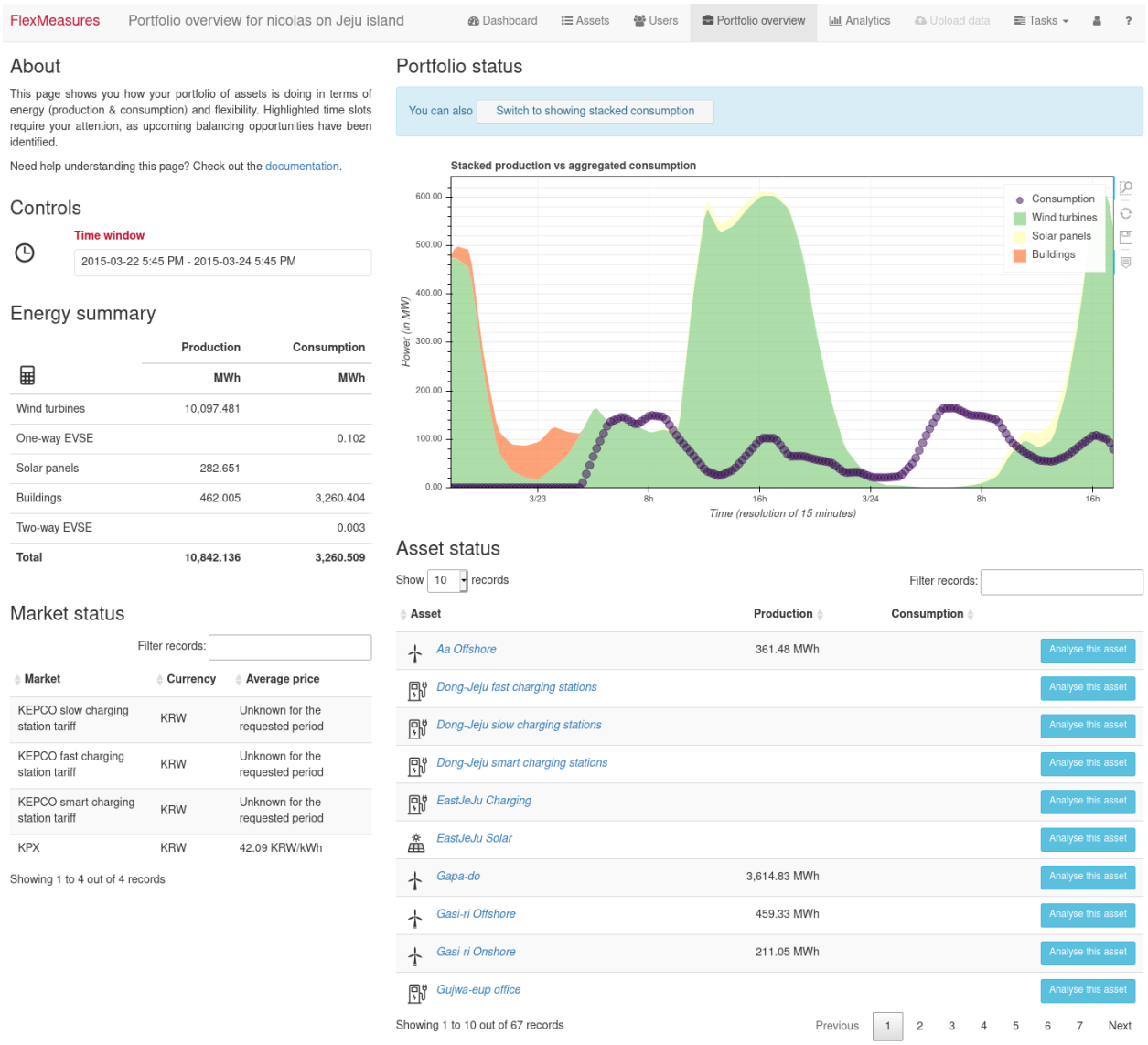
TEN

# PORTFOLIO OVERVIEW

The portfolio overview shows results and opportunities regarding the user's asset portfolio. The view serves to get an overview over the portfolio's energy status and can be viewed with either the consumption or the generation side aggregated.

In particular, the page contains:

- *Statements about energy and flex activations*
- *Power profile measurements and forecasts*
- *Changes to the power profile due to flexible schedules*
- *Opportunities to valorise on flexibility*




## 10.1 Statements about energy and flex activations

The financial statements separate the effects of energy consumption/production and flexible schedules over two tables.

### 10.1.1 Energy summary

The top table lists the effects of energy trading for each asset type in the user's portfolio. Production and consumption values are total volumes within the selected time window.[1]

Costs and revenues are calculated based on the relevant market prices for the user within the selected time window. A consumer will only have costs, while a prosumer may have both costs and revenues. A supplier has revenues, since it sells energy to the other roles within FlexMeasures.

Finally, the financial statements show the total profit or loss per asset type.

[1] For time windows that include future time slots, future values are based on forecasts.

### 10.1.2 Market status

**Note:** This feature is mocked for now.

The bottom table lists the effects of flexible schedules for each asset type in the user's portfolio. Separate columns are stated for each type of scheduled deviation from the status quo, e.g. curtailment and shifting (see *Types of flexibility*), with relevant total volumes within the selected time window.[?]

Costs and revenues are calculated based on the following internal method for profit sharing: Asset owners that follow flexible schedules via the platform will generate revenues. Suppliers that follow flexible schedules via the platform will generate both costs and revenues, where the revenues come from interacting with external markets. Finally, the financial statements show the total profit or loss per asset.

## 10.2 Power profile measurements and forecasts

The power profile shows total production and consumption over the selected time window. A switch allows the user to view the contribution of each asset type to either total as a stacked plot. Past time slots show measurement data, whereas future time slots show forecasts. When suggested changes exist in flexible schedules during the selected time window, the plot is overlaid with highlights (see *Opportunities to valorise on flexibility* ).

## 10.3 Changes to the power profile due to flexible schedules

A crucial goal of FlexMeasures is to visualise the opportunities within flexible schedules. This goal is not yet completely realised, but we show a mock here of how this could like when realised:

Just below the power profile, the net effect of flexible schedules that have previously been computed by FlexMeasures is plotted. The profile indicates the change in power resulting from schedules that are planned in the future, as well as from schedules that had been planned in the past. Positive values indicate an increase in production or a decrease in consumption, both of which result in an increased load on the network. For short-term changes in power due to activation of flexibility, this is sometimes called up-regulation. Negative values indicate a decrease in production or an increase in consumption, which result in a decreased load on the network (down-regulation). When flexibility opportunities exist in the selected time window, the plot is overlaid with highlights (see *Opportunities to valorise on flexibility* ).

## 10.4 Opportunities to valorise on flexibility

When flexibility opportunities exist in the selected time window, plots are overlaid with highlights indicating time slots in which flexible scheduling adjustments can be taken in the future or were missed in the past. The default time window (the next 24 hours) shows immediately upcoming opportunities to valorise on flexibility opportunities. The user could learn more about identified opportunities on a yet-to-be-developed view which goes further into details.

CHAPTER

ELEVEN

# FLEXIBILITY OPPORTUNITIES

Flexibility opportunities have commercial value that users can valorise on. When FlexMeasures has identified commercial value of flexibility, the user is suggested to act on it. This might happen in an automated fashion (scripts reading out suggested schedules from the FlexMeasures API and implementing them to local operations if possible) or manually (operators agreeing with the opportunities identified by FlexMeasures and acting on the suggested schedules).

For this latter case, in the Flexibility opportunities web-page (a yet-to-be designed UI feature discussed below), FlexMeasures could show all flexibility opportunities that the user can act on for a selected time window.



## 11.1 Visualisation of opportunities

Visualising flexibility opportunities and their effects is not straightforward. Flexibility opportunities can cause changes to the power profile of an asset in potentially complex ways. One example is called the rebound effect, where a decrease in consumption leads to an increase in consumption at a later point in time, because consumption is essentially postponed. Such effects could be taken into account by FlexMeasures and shown to the user, e.g. as a part of expected value calculations and power profile forecasts.

Below is an example of what this could look like. This is a potential UX design which we have not implemented yet.


The operator can select flexibility opportunities with an attached value, and see the effects on the power profile in a visual manner. Listed flexibility opportunities include previously realised opportunities and currently offered opportunities. Currently offered opportunities are presented as an order book, where they are sorted according to their commercial value.

Of course, depending on the time window selection and constraints set by the asset owner, the rebound effects of an opportunity may partially take place outside of the selected time window.

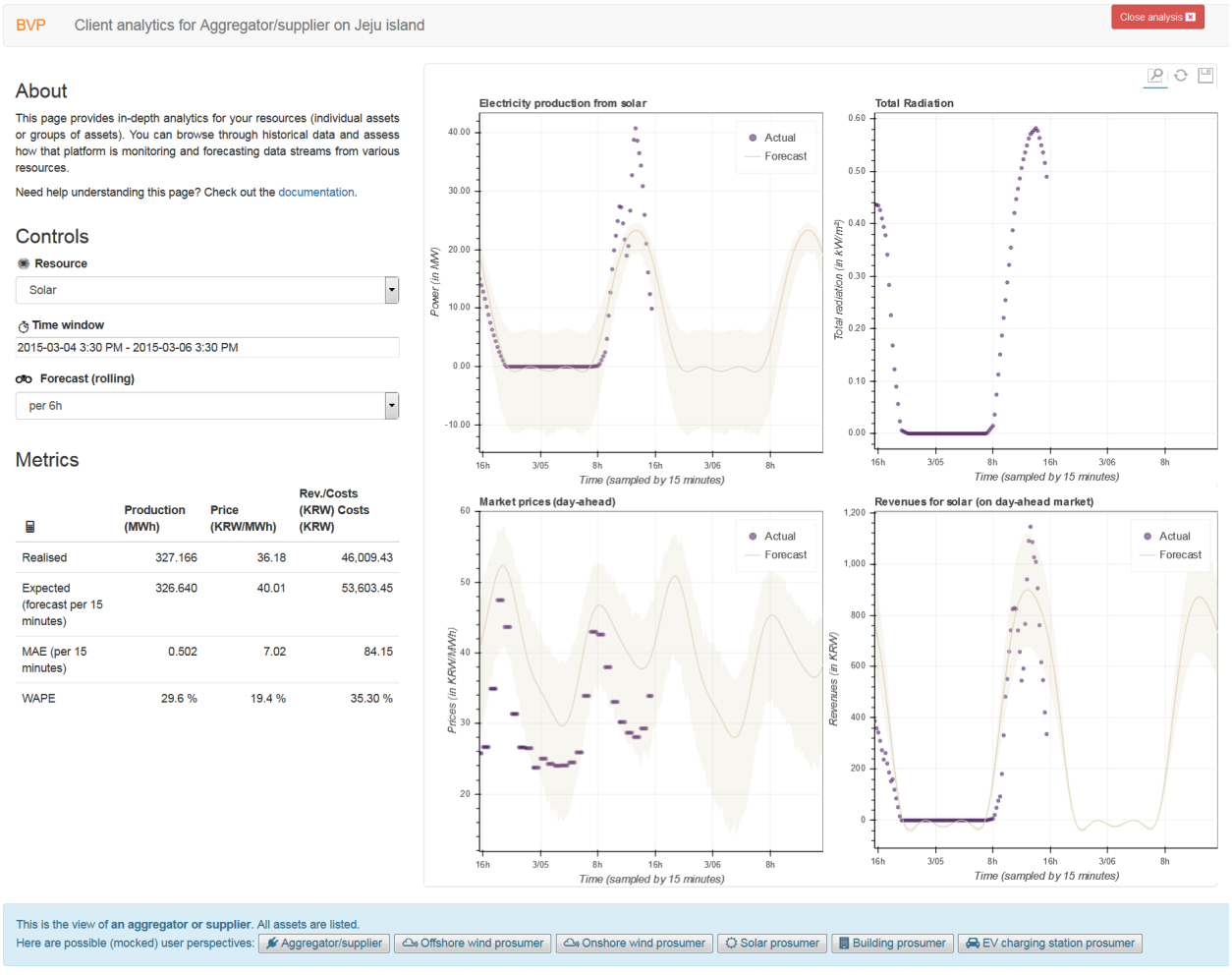
CHAPTER

TWELVE

# CLIENT ANALYTICS

The client analytics page shows relevant data to an asset's operation: production and consumption, market prices and weather data. The view serves to browse through available data history and to assess how the app is monitoring and forecasting data streams from various sources. In particular, the page contains:





## 12.1 Data filtering

FlexMeasures offers data analytics on various aggregation levels: per asset, per asset type or even per higher aggregation levels like all renewables.

The time window is freely selectable.

In addition, the source of market and weather data can be selected, as well as the forecast horizon.

For certain assets, which bundle meters on the same location, individual traces can be shown next to each other in the (upper left) power plot, for comparison.

## 12.2 Data visualisation

In each plot, the data is shown for different types of data: measurements (e.g. of power or prices), forecasts and schedules (only for power, obviously).

In the FlexMeasures platform, forecasting models can indicate a range of uncertainty around their forecasts, which will also be shown in plots if available.

## 12.3 Metrics

FlexMeasures summarises the visualised data as realised (by measurement) and expected (by forecast) sums. In addition, the mean average error (MAE) and the weighted absolute percentage error (WAPE) are computed for power, weather and price data if forecasts are available for the chosen time range.

CHAPTER

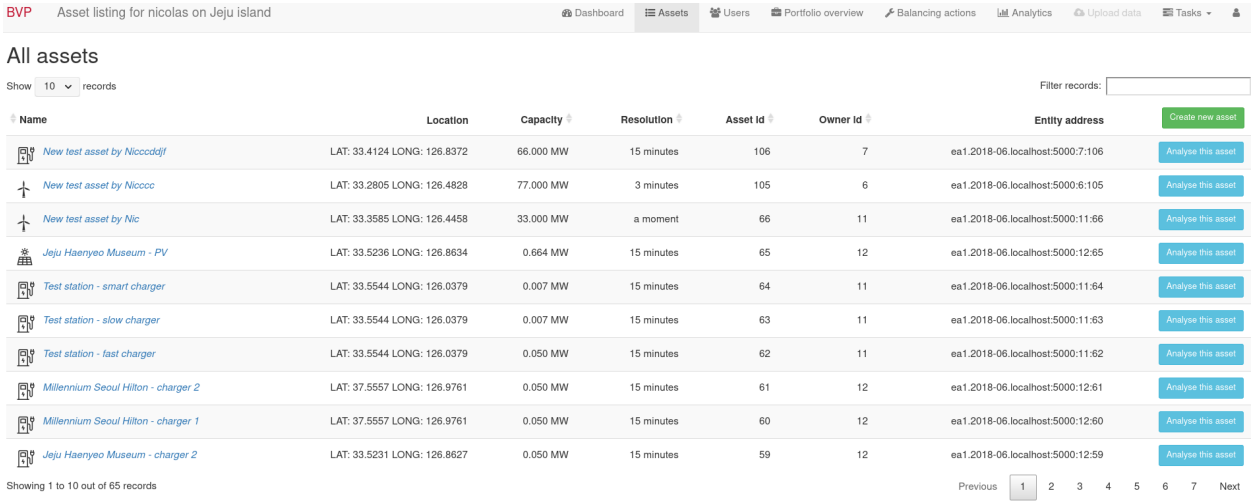
THIRTEEN

# ADMINISTRATION

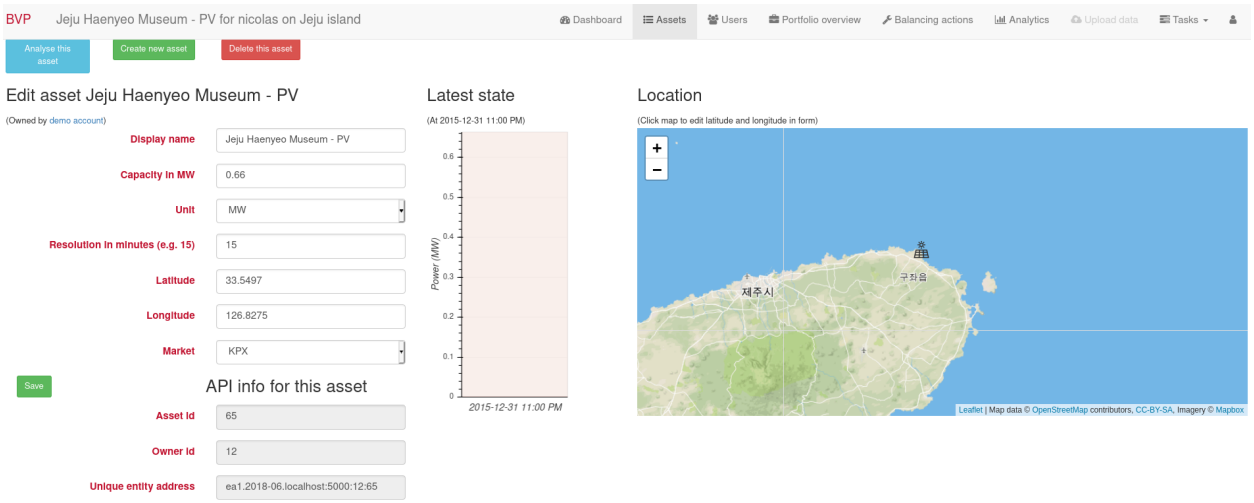
The administrator can edit assets and user accounts here.

## 13.1 Assets

Listing all assets:

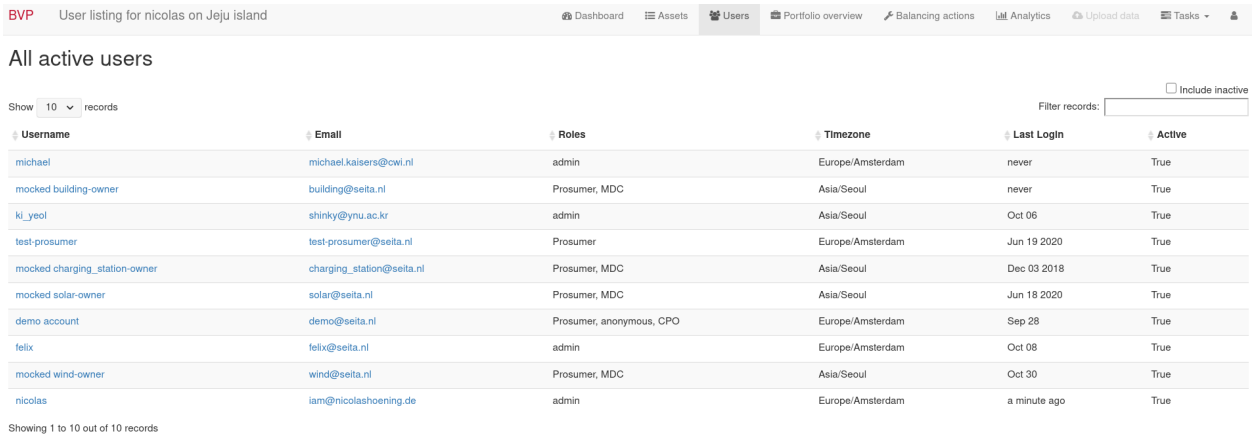



Viewing & editing one asset:

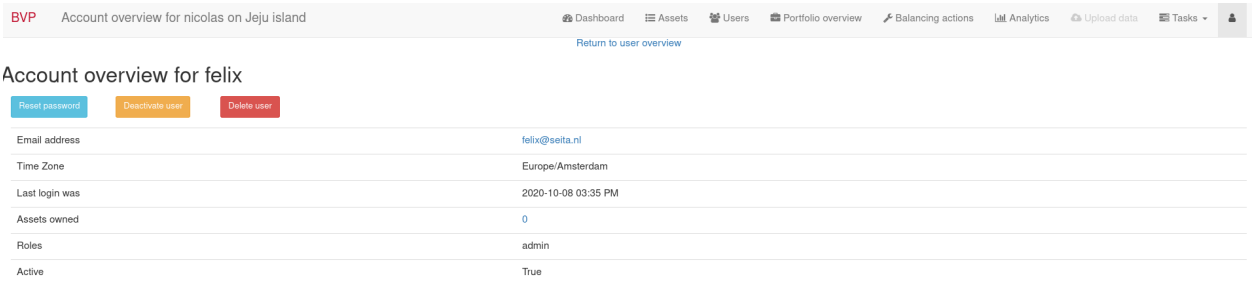


## 13.2 Users

Listing all users:




Viewing one user:



CHAPTER

# FOURTEEN

# INTRODUCTION

This document details the Application Programming Interface (API) of the FlexMeasures web service. The API supports user automation for flexibility valorisation in the energy sector, both in a live setting and for the purpose of simulating scenarios. The web service adheres to the concepts and terminology used in the Universal Smart Energy Framework (USEF). We assume in this document that the FlexMeasures instance you want to connect to is hosted at https://company.flexmeasures.io.

New versions of the API are released on:

```
https://company.flexmeasures.io/api
```

A list of services offered by (a version of) the FlexMeasures web service can be obtained by sending a *getService* request. An optional field "access" can be used to specify a user role for which to obtain only the relevant services.

**Example request**

```
{
    "type": "GetServiceRequest",
    "version": "1.0"
}
```

**Example response**

```
{
    "type": "GetServiceResponse",
    "version": "1.0",
    "services": [
        {
            "name": "getMeterData",
            "access": ["Aggregator", "Supplier", "MDC", "DSO", "Prosumer", "ESCo"],
            "description": "Request meter reading"
        },
        {
            "name": "postMeterData",
            "access": ["MDC"],
            "description": "Send meter reading"
        }
    ]
}
```

## 14.1 Authentication

Service usage is only possible with a user access token specified in the request header, for example:

```
{
    "Authorization": "<token>"
}
```

A fresh "<token>" can be generated on the user's profile after logging in:

```
https://company.flexmeasures.io/account
```

or through a POST request to the following endpoint:

```
https://company.flexmeasures.io/api/requestAuthToken
```

using the following JSON message for the POST request data:

```
{
    "email": "<user email>",
    "password": "<user password>"
}
```

**Note:** Each access token has a limited lifetime, see *Authentication and Authorisation*.

## 14.2 Roles

We distinguish the following roles with different access rights to the individual services. Capitalised roles are defined by USEF:

- public
- user
- admin
- Aggregator
- Supplier: an energy retailer (see supplier)
- Prosumer: an asset owner (see prosumer)
- ESCo: an energy service company (see esco)
- MDC: a meter data company (see mdc)
- DSO: a distribution system operator (see dso)

## 14.3 Sources

Requests for data may limit the data selection by specifying a source, for example, a specific user. USEF roles are also valid source selectors. For example, to obtain data originating from either a meter data company or user 42, include the following:

```
{
    "sources": ["MDC", "42"],
}
```

## 14.4 Notation

All requests and responses to and from the web service should be valid JSON messages.

### 14.4.1 Singular vs plural keys

Throughout this document, keys are written in singular if a single value is listed, and written in plural if multiple values are listed, for example:

```
{
    "keyToValue": "this is a single value",
    "keyToValues": ["this is a value", "and this is a second value"]
}
```

The API, however, does not distinguish between singular and plural key notation.

### 14.4.2 Connections

Connections are end points of the grid at which an asset is located. Connections should be identified with an entity address following the EA1 addressing scheme prescribed by USEF[1], which is mostly taken from IETF RFC 3720 [2]:

This is the complete structure of an EA1 address:

```
{
    "connection": "ea1.{date code}.{reversed domain name}:{locally unique string}"
}
```

Here is a full example for a FlexMeasures connection address:

```
{
    "connection": "ea1.2021-02.io.flexmeasures.company:30:73"
}
```

where FlexMeasures runs at *company.flexmeasures.io* and the owner ID is 30 and the asset ID is 73. The owner ID is optional. Both the owner ID and the asset ID, as well as the full entity address can be obtained on the asset's listing after logging in:

```
https://company.flexmeasures.io/assets
```

Some deeper explanations about an entity address:

- "ea1" is a constant, indicating this is a type 1 USEF entity address

- The date code "must be a date during which the naming authority owned the domain name used in this format, and should be the first month in which the domain name was owned by this naming authority at 00:01 GMT of the first day of the month.
- The reversed domain name is taken from the naming authority (person or organization) creating this entity address
- The locally unique string can be used for local purposes, and FlexMeasures uses it to identify the resource (more information in parse_entity_address).

[1] https://www.usef.energy/app/uploads/2020/01/USEF-Flex-Trading-Protocol-Specifications-1.01.pdf

[2] https://tools.ietf.org/html/rfc3720

#### Notation for simulation

For version 1 of the API, the following simplified addressing scheme may be used:

```
{
    "connection": "<owner-id>:<asset-id>"
}
```

or even simpler:

```
{
    "connection": "<asset-id>"
}
```

### 14.4.3 Groups

Data such as measurements, load prognoses and tariffs are usually stated per group of connections. When the attributes "start", "duration" and "unit" are stated outside of "groups" they are inherited by each of the individual groups. For example:

```
{
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 2"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connection": "CS 3",
            "values": [
                306.66,
                0,
                0,
```

(continues on next page)

(continued from previous page)

```
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2016-05-01T12:45:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

In case of a single group of connections, the message may be flattened to:

```
{
    "connections": [
        "CS 1",
        "CS 2"
    ],
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2016-05-01T12:45:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

### 14.4.4 Timeseries

Timestamps and durations are consistent with the ISO 8601 standard. All timestamps in requests to the API must be timezone-aware. The timezone indication "Z" indicates a zero offset from UTC. Additionally, we use the following shorthand for sequential values within a time interval:

```
{
    "values": [
        10,
        5,
        8
    ],
    "start": "2016-05-01T13:00:00Z",
    "duration": "PT45M"
}
```

is equal to:

```
{
    "timeseries": [
        {
            "value": 10,
            "start": "2016-05-01T13:00:00Z",
            "duration": "PT15M"
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "value": 5,
            "start": "2016-05-01T13:15:00Z",
            "duration": "PT15M"
        },
        {
            "value": 8,
            "start": "2016-05-01T13:30:00Z",
            "duration": "PT15M"
        }
    ]
}
```

This intuitive convention allows us to reduce communication by sending univariate timeseries as arrays.

#### Notation for v1

For version 1 of the API, only univariate timeseries data is expected to be communicated. Therefore:

- only the array notation should be used,
- "start" should be a timestamp on the hour or a multiple of 15 minutes thereafter, and
- "duration" should be a multiple of 15 minutes.

### 14.4.5 Beliefs

By regarding all time series data as beliefs that have been recorded at a certain time, data can be filtered accordingly. Some GET endpoints have two optional timing fields to allow such filtering. The "prior" field (a timestamp) can be used to select beliefs recorded before some moment in time. It can be used to "time-travel" to see the state of information at some moment in the past. In addition, the "horizon" field (a duration) can be used to select beliefs recorded before some moment in time, relative to each event. For example, to filter out meter readings communicated within a day (denoted by a negative horizon) or forecasts created at least a day beforehand (denoted by a positive horizon). In addition to these two timing filters, beliefs can be filtered by their source (see *Sources*).

The two timing fields follow the ISO 8601 standard and are interpreted as follows:

- "horizon": recorded at least <duration> before the fact (indicated by a positive horizon), or at most <duration> after the fact (indicated by a negative horizon).
- "prior": recorded prior to <timestamp>.

For example:

```
{
    "horizon": "PT6H",
    "prior": "2020-08-01T17:00:00Z"
}
```

These fields denote that the data should have been recorded at least 6 hours before the fact (i.e. forecasts) and prior to 5 PM on August 1st 2020 (UTC).

### 14.4.6 Prognoses

Some POST endpoints have two optional fields to allow setting the time at which beliefs are recorded explicitly. This is useful to keep an accurate history of what was known at what time, especially for prognoses. If not used, FlexMeasures will infer the prior from the arrival time of the message.

The "prior" field (a timestamp) can be used to set a single time at which the entire prognosis was recorded. Alternatively, the "horizon" field (a duration) can be used to set the recording times relative to each prognosed event. In case both fields are set, the earliest possible recording time is determined and recorded for each prognosed event.

The two timing fields follow the ISO 8601 standard and are interpreted as follows:

```
{
    "values": [
        10,
        5,
        8
    ],
    "start": "2016-05-01T13:00:00Z",
    "duration": "PT45M",
    "prior": "2016-05-01T07:45:00Z",
}
```

This message implies that the entire prognosis was recorded at 7:45 AM UTC, i.e. 6 hours before the end of the entire time interval.

```
{
    "values": [
        10,
        5,
        8
    ],
    "start": "2016-05-01T13:00:00Z",
    "duration": "PT45M",
    "horizon": "PT6H"
}
```

This message implies that all prognosed values were recorded 6 hours in advance. That is, the value for 1:00-1:15 PM was made at 7:15 AM, the value for 1:15-1:30 PM was made at 7:30 AM, and the value for 1:30-1:45 PM was made at 7:45 AM.

Negative horizons may also be stated (breaking with the ISO 8601 standard) to indicate a prognosis about something that has already happened (i.e. after the fact, or simply *ex post*). For example, the following message implies that all prognosed values were made 10 minutes after the fact:

```
{
    "values": [
        10,
        5,
        8
    ],
    "start": "2016-05-01T13:00:00Z",
    "duration": "PT45M",
    "horizon": "-PT10M"
}
```

Note that, for a horizon indicating a prognosis 10 minutes after the *start* of each 15-minute interval, the "horizon" would have been "PT5M". This denotes that the prognosed interval has 5 minutes left to be concluded.

### 14.4.7 Resolutions

Specifying a resolution is redundant for POST requests that contain both "values" and a "duration". Also, posted data is checked against the required resolution of the assets which are posted to.

GET requests (such as *getMeterData*) return data in the resolution which the sensor is configured for. A "resolution" may be specified explicitly to obtain the data in downsampled form, which can be very beneficial for download speed. The specified resolution needs to be a multiple of the asset's resolution, e.g. hourly or daily values if the asset's resolution is 15 minutes.

### 14.4.8 Units

Valid units for timeseries data in version 1 of the API are "MW" only.

### 14.4.9 Signs

USEF recommends to use positive power values to indicate consumption and negative values to indicate production, i.e. to take the perspective of the Prosumer. If an asset has been configured as a pure producer or pure consumer, the web service will help avoid mistakes by checking the sign of posted power values.

CHAPTER

FIFTEEN

# SIMULATION

This document details examples for using a FlexMeasures server for simulation. The API on a server that is set up for simulation is extended with several features that make it possible to run simulations of energy flows and control actions. Please read the *Introduction* for explanations of the message fields, specifically regarding:

- The sign of values (*Signs*)
- Valid durations (*Resolutions*)
- Valid horizons (*Prognoses*)
- Valid units (*Simulation*)

**Table of contents**

## 15.1 Setting up

Researchers require an admin account to set up a new simulation with a number of assets.

### 15.1.1 Creating assets and owners

New assets can be created through the UI on:

```
https://company.flexmeasures.io/assets/new
```

We recommend that researchers choose their own admin account as the asset's owner. This way, the simulation will only require refreshing of the access token for the admin account. Alternatively, researchers can set up unique accounts for each agent in a multi-agent simulation by creating new owners. In this case, access tokens need to be refreshed by each agent separately.

### 15.1.2 Authentication

Service usage is only possible with a user authentication token specified in the request header, for example:

```
{
    "Authorization": "<token>"
}
```

The "<token>" can be obtained on your profile after logging in:

```
https://company.flexmeasures.io/account
```

For security reasons, tokens expire after a certain amount of time (see _auth). To automate token renewal, use the following POST endpoint:

```
https://company.flexmeasures.io/api/requestAuthToken
```

Providing applicable user credentials:

```
{
    "email": "<email>",
    "password": "<password>"
}
```

## 15.2 Posting weather data

Weather data (both observations and forecasts) can be posted to the following POST endpoint:

```
https://company.flexmeasures.io/api/<version>/postWeatherData
```

Weather data can be posted for the following three types of weather sensors:

- "radiation" (with kW/m$^2$ as unit)
- "temperature" (with °C as unit)
- "wind_speed" (with m/s as unit)

The sensor type is part of the unique entity address for each sensor, together with the sensor's latitude and longitude.

This "PostWeatherDataRequest" message posts temperature forecasts for 15-minute intervals between 3.00pm and 4.30pm for a weather sensor located at latitude 33.4843866 and longitude 126.477859. The forecasts were made at noon.

```
{
    "type": "PostWeatherDataRequest",
    "sensor": "ea1.2018-06.io.flexmeasures.company:temperature:33.4843866:126.477859",
    "values": [
        20.04,
        20.23,
        20.41,
        20.51,
        20.55,
        20.57
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT1H30M",
    "prior": "2015-01-01T12:00:00+09:00",
    "unit": "°C"
}
```

### 15.2.1 Observations vs forecasts

To post an observation rather than a forecast, simply set the prior to the moment at which the observations were made, e.g. at "2015-01-01T16:30:00+09:00". This denotes that the observation was made exactly after realisation of this list of temperature readings, i.e. at 4.30pm.

Alternatively, to indicate that each individual observation was made directly after the end of its 15-minute interval (i.e. at 3.15pm, 3.30pm and so on), set a horizon to "PT0H" instead of a prior.

Finally, delays in reading out sensor data can be simulated by setting the horizon field to a negative value. For example, a horizon of "-PT1H" would denote that each temperature reading was observed one hour after the fact (i.e. at 4.15pm, 4.30 pm and so on).

See *Prognoses* for more information regarding the prior and horizon fields.

## 15.3 Posting price data

Price data (both observations and forecasts) can be posted to the following POST endpoint:

```
https://company.flexmeasures.io/api/<version>/postPriceData
```

This example "PostPriceDataRequest" message posts prices for hourly intervals between midnight and midnight the next day for the Korean Power Exchange (KPX) day-ahead auction. The horizon indicates that the prices were published at 3pm on December 31st 2014 (i.e. 33 hours ahead of midnight the next day).

```
{
    "type": "PostPriceDataRequest",
    "market": "ea1.2018-06.io.flexmeasures.company:kpx_da",
    "values": [
        52.37,
        51.14,
        49.09,
```

(continues on next page)

(continued from previous page)

```
        48.35,
        48.47,
        49.98,
        58.7,
        67.76,
        69.21,
        70.26,
        70.46,
        70,
        70.7,
        70.41,
        70,
        64.53,
        65.92,
        69.72,
        70.51,
        75.49,
        70.35,
        70.01,
        66.98,
        58.61
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT24H",
    "horizon": "PT33H",
    "unit": "KRW/kWh"
}
```

### 15.3.1 Observations vs forecasts

For markets, the time at which the market is cleared (i.e. when contracts are signed) determines the difference between an ex-post observation and an ex-ante forecast. For example, at the KPX day-ahead auction this is every day at 3pm. To post a forecast rather than an observation, simply increase the horizon. For example, a horizon of "PT57H" would denote a forecast of 24 hours ahead of clearing.

## 15.4 Posting power data

For power data, USEF specifies separate message types for observations and forecasts. Correspondingly, FlexMeasures uses separate endpoints to communicate these messages. Observations of power data can be posted to the following POST endpoint:

```
https://company.flexmeasures.io/api/<version>/postMeterData
```

while forecasts of power data can be posted to the following POST endpoint:

```
https://company.flexmeasures.io/api/<version>/postPrognosis
```

For both endpoints, power data can be posted in various ways. The following examples assume that the endpoint for power data observations (i.e. meter data) is used.

### 15.4.1 Single value, single connection

A single average power value for a 15-minute time interval for a single connection, posted 5 minutes after realisation.

```
{
    "type": "PostMeterDataRequest",
    "connection": "ea1.2018-06.io.flexmeasures.company:1:1",
    "value": 220,
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT0H15M",
    "horizon": "-PT5M",
    "unit": "MW"
}
```

### 15.4.2 Multiple values, single connection

Multiple values (indicating a univariate timeseries) for 15-minute time intervals for a single connection, posted 5 minutes after realisation.

```
{
    "type": "PostMeterDataRequest",
    "connection": "ea1.2018-06.io.flexmeasures.company:1:1",
    "values": [
        220,
        210,
        200
    ],
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT0H45M",
    "horizon": "-PT5M",
    "unit": "MW"
}
```

### 15.4.3 Single identical value, multiple connections

Single identical value for a 15-minute time interval for two connections, posted 5 minutes after realisation. Please note that both connections consumed at 10 MW, i.e. the value does not represent the total of the two connections. We recommend to use this notation for zero values only.

```
{
    "type": "PostMeterDataRequest",
    "connections": [
        "ea1.2018-06.io.flexmeasures.company:1:1",
        "ea1.2018-06.io.flexmeasures.company:1:2"
    ],
    "value": 10,
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT0H15M",
    "horizon": "-PT5M",
    "unit": "MW"
}
```

### 15.4.4 Single different values, multiple connections

Single different values for a 15-minute time interval for two connections, posted 5 minutes after realisation.

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connection": "ea1.2018-06.io.flexmeasures.company:1:1",
            "value": 220
        },
        {
            "connection": "ea1.2018-06.io.flexmeasures.company:1:2",
            "value": 300
        }
    ],
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT0H15M",
    "horizon": "-PT5M",
    "unit": "MW"
}
```

### 15.4.5 Multiple values, multiple connections

Multiple values (indicating a univariate timeseries) for 15-minute time intervals for two connections, posted 5 minutes after realisation.

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connection": "ea1.2018-06.io.flexmeasures.company:1:1",
            "values": [
                220,
                210,
                200
            ]
        },
        {
            "connection": "ea1.2018-06.io.flexmeasures.company:1:2",
            "values": [
                300,
                303,
                306
            ]
        }
    ],
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT0H45M",
    "horizon": "-PT5M",
    "unit": "MW"
}
```

## 15.5 Getting prognoses

Prognoses are power forecasts that are used by FlexMeasures to determine the best control signals to valorise on balancing opportunities. Researchers can check the accuracy of these forecasts by downloading the prognoses and comparing them against the meter data, i.e. the realised power measurements. A prognosis can be requested for a single asset at the following GET endpoint:

```
https://company.flexmeasures.io/api/<version>/getPrognosis
```

This example requests a prognosis with a rolling horizon of 6 hours before realisation.

```
{
    "type": "GetPrognosisRequest",
    "connection": "ea1.2018-06.io.flexmeasures.company:1:1",
    "start": "2015-01-01T00:00:00+00:00",
    "duration": "PT24H",
    "horizon": "PT6H",
    "resolution": "PT15M",
    "unit": "MW"
}
```

## 15.6 Posting flexibility constraints

Prosumers that have Active Demand & Supply can post the constraints of their flexible devices to FlexMeasures at the following POST endpoint:

```
https://company.flexmeasures.io/api/<version>/postUdiEvent
```

This example posts a state of charge value for a battery device (asset 10 of owner 7) as UDI event 203.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "value": 12.1,
    "datetime": "2015-06-02T10:00:00+00:00",
    "unit": "kWh"
}
```

Some devices also accept target values for their state of charge. As an example, consider the same UDI event as above with an additional target value.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:204:soc-with-targets",
    "value": 12.1,
    "datetime": "2015-06-02T10:00:00+00:00",
    "unit": "kWh",
    "targets": [
        {
            "value": 25,
            "datetime": "2015-06-02T16:00:00+00:00"
        }
    ]
}
```

## 15.7 Getting control signals

A Prosumer can query FlexMeasures for control signals for its flexible devices using the following GET endpoint:

```
https://company.flexmeasures.io/api/<version>/getDeviceMessage
```

Control signals can be queried by UDI event for up to 1 week after the UDI event was posted. This example requests a control signal for UDI event 203 posted previously.

```
{
    "type": "GetDeviceMessageRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc"
}
```

The following example response indicates that FlexMeasures planned ahead 45 minutes. The list of consecutive power values represents the target consumption of the battery (negative values for production). Each value represents the average power over a 15 minute time interval.

```
{
    "type": "GetDeviceMessageResponse",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203",
    "values": [
        2.15,
        3,
        2
    ],
    "start": "2015-06-02T10:00:00+00:00",
    "duration": "PT45M",
    "unit": "MW"
}
```

One way of reaching the target consumption in this example is to let the battery start to consume with 2.15 MW at 10am, increase its consumption to 3 MW at 10.15am and decrease its consumption to 2 MW at 10.30am. However, because the targets values represent averages over 15-minute time intervals, the battery still has some degrees of freedom. For example, the battery might start to consume with 2.1 MW at 10.00am and increase its consumption to 2.25 at 10.10am, increase its consumption to 5 MW at 10.15am and decrease its consumption to 2 MW at 10.20am. That should result in the same average values for each quarter-hour.

## 15.8 Resetting the server

All power, price and weather data on the simulation server can be cleared using the following PUT endpoint (admin rights are required):

```
https://company.flexmeasures.io/api/<version>/restoreData
```

This example restores the database to a backup named demo_v0, which contains no timeseries data.

```
{
    "backup": "demo_v0"
}
```

CHAPTER

# SIXTEEN

# VERSION 2.0

## 16.1 Summary

| Resource | Operation | Description |
|---|---|---|
| Asset | *DELETE /api/v2_0/asset/(id)* | Delete an asset, together with its existing data. |
| | *GET /api/v2_0/asset/(id)* | Get an asset |
| | *PATCH /api/v2_0/asset/(id)* | Patch data for an existing asset |
| | *GET /api/v2_0/assets* | Download asset list |
| | *POST /api/v2_0/assets* | Post a new asset |
| | *GET /api/v2_0/getConnection* | Retrieve entity addresses of connections |
| Chart | *GET /api/v2_0/charts/power* | Get a power chart |
| | *GET /api/v2_0/charts/power* | |
| Control | *GET /api/v2_0/getDeviceMessage* | Download control signal from the platform |
| | *POST /api/v2_0/postUdiEvent* | Upload flexibility constraints to the platform |
| Data | *GET /api/v2_0/getMeterData* | Download meter data from the platform |
| | *GET /api/v2_0/getPrognosis* | Download prognosis from the platform |
| Public | *GET /api/* | List available API versions |
| | *POST /api/requestAuthToken* | Obtain an authentication token |
| | *GET /api/v2_0/getService* | Obtain a service listing for this version |
| User | *POST /api/v2_0/postMeterData* | Upload meter data to the platform |
| | *POST /api/v2_0/postPriceData* | Upload price data to the platform |
| | *POST /api/v2_0/postPrognosis* | Upload prognosis to the platform |
| | *POST /api/v2_0/postWeatherData* | Upload weather data to the platform |
| | *GET /api/v2_0/user/(id)* | Get a user |
| | *PATCH /api/v2_0/user/(id)* | Patch data for an existing user |
| | *PATCH /api/v2_0/user/(id)/password-reset* | Password reset |
| | *GET /api/v2_0/users* | Download user list |

## 16.2 API Details

**GET /api/**
Public endpoint to list API versions.

**POST /api/requestAuthToken**
API endpoint to get a fresh authentication access token. Be aware that this fresh token has a limited lifetime (which depends on the current system setting SECURITY_TOKEN_MAX_AGE).

Pass the *email* parameter to identify the user. Pass the *password* parameter to authenticate the user (if not already authenticated in current session)

**DELETE /api/v2_0/asset/**(*id*)
API endpoint to delete an asset, and its sensed data.

This endpoint deletes an existing asset, as well as all measurements recorded for it. Only users who own the asset are allowed to delete the asset.

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 204 No Content – DELETED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

**GET /api/v2_0/asset/**(*id*)
API endpoint to get an asset.

This endpoint gets an asset. Only users who own the asset can use this endpoint.

**Example response**


```
{
    "asset_type": "battery",
    "capacity_in_mw": 2.0,
    "display_name": "Test battery",
    "event_resolution": 5,
    "id": 1,
    "latitude": 10,
    "longitude": 100,
    "market": 1,
    "max_soc_in_mwh": 5,
    "min_soc_in_mwh": 0,
    "name": "Test battery",
    "owner": 2,
    "soc_datetime": "2015-01-01T00:00:00+00:00",
    "soc_in_mwh": 2.5,
    "soc_udi_event_id": 203,
```


(continues on next page)

(continued from previous page)

```
    "unit": "kW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

**PATCH /api/v2_0/asset/**(*id*)

API endpoint to patch asset data.

This endpoint sets data for an existing asset. Any subset of asset fields can be sent. Only users who own the asset are allowed to update its data.

Several fields are not allowed to be updated, e.g. id. They are ignored.

**Example request**

```
{
    "latitude": 11.1,
    "longitude": 99.9,
}
```

Note that event_resolution is expected as the number of minutes and soc_datetime is expected as ISO8601 datetime string.

**Example response**

The whole asset is returned in the response:

```
{
    "asset_type": "battery",
    "capacity_in_mw": 2.0,
    "display_name": "Test battery",
    "event_resolution": 5,
    "id": 1,
    "latitude": 11.1,
    "longitude": 99.9,
    "market": 1,
    "max_soc_in_mwh": 5,
    "min_soc_in_mwh": 0,
    "name": "Test battery",
    "owner": 2,
    "soc_datetime": "2015-01-01T00:00:00+00:00",
```

(continues on next page)

(continued from previous page)


```
    "soc_in_mwh": 2.5,
    "soc_udi_event_id": 203,
    "unit": "kW"
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – UPDATED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**GET /api/v2_0/assets**

API endpoint to get assets.

This endpoint returns all accessible assets for a given owner. The *owner_id* query parameter can be used to set an owner. If no owner is set, all accessible assets are returned. A non-admin user can only access its own assets.

**Example response**

An example of one asset being returned:


```
[
    {
        "asset_type": "battery",
        "capacity_in_mw": 2.0,
        "display_name": "Test battery",
        "event_resolution": 10,
        "id": 1,
        "latitude": 10,
        "longitude": 100,
        "market": 1,
        "max_soc_in_mwh": 5,
        "min_soc_in_mwh": 0,
        "name": "Test battery",
        "owner": 2,
        "soc_datetime": "2015-01-01T00:00:00+00:00",
        "soc_in_mwh": 2.5,
        "soc_udi_event_id": 203,
        "unit": "MW"
    }
]
```


Note that event_resolution is returned as the number of minutes and soc_datetime is returned as ISO8601 datetime string.

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

**POST /api/v2_0/assets**

API endpoint to post a new asset.

This endpoint creates a new asset. Only users with the admin role are allowed to create assets.

**Example request**

The following example contains the required fields only, plus the two state of charge (soc) fields which a battery asset needs to specify:

```
{
    "name": "Test battery",
    "asset_type": "battery",
    "unit": "kW",
    "owner": 2,
    "market": 1,
    "event_resolution": 5,
    "capacity_in_mw": 4.2,
    "latitude": 40,
    "longitude": 170.3,
    "max_soc_in_mwh": 5,
    "min_soc_in_mwh": 0
}
```

Note that event_resolution is expected as the number of minutes and soc_datetime is expected as ISO8601 datetime string.

**Example response**

The newly posted asset, including all fields, is returned in the response:

```
{
    "id": 1,
    "asset_type": "battery",
    "unit": "kW"
    "capacity_in_mw": 4.2,
    "display_name": "Test battery",
    "event_resolution": 5,
    "latitude": 40,
    "longitude": 170.3,
    "max_soc_in_mwh": 5,
    "min_soc_in_mwh": 0,
    "name": "Test battery",
```

(continues on next page)

(continued from previous page)

```
    "owner": 2,
    "market": 1,
    "soc_datetime": null,
    "soc_in_mwh": null,
    "soc_udi_event_id": null
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 201 Created – CREATED
- 400 Bad Request – INVALID_REQUEST
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

**GET /api/v2_0/charts/power**

**GET /api/v2_0/charts/power**

API endpoint to get a chart for power data which can be embedded in web pages.

This endpoint returns a Bokeh chart with power data which can be embedded in a website. It includes forecasts and even schedules, if available.

**Example request**

An example of a chart request:

```
{
    "resource": ""my-battery,
    "start_time": "2020-02-20:10:00:00UTC",
    "end_time": "2020-02-20:11:00:00UTC",
    "resolution": "PT15M",
    "consumption_as_positive": true
    "resolution": "PT6H",
    "show_individual_traces_for": "none"  // can be power or schedules
}
```

On your webpage, you need to include the Bokeh libraries, e.g.:

<script src="https://cdn.pydata.org/bokeh/release/bokeh-1.0.4.min.js"></script>

(The version needs to match the version used by the FlexMeasures server, see requirements/app.txt)

Then you can call this endpoint and include the result like this:

```
<script>
    fetch('http://localhost:5000/api/v2_0/charts/power?' + urlData.toString(),
    {
        method: "GET",
```

(continues on next page)

(continued from previous page)

```
        mode: "cors",
        headers:
            {
                "Content-Type": "application/json",
                "Authorization": "<users auth token>"
            },
    })
    .then(function(response) { return response.json(); })
    .then(function(item) { Bokeh.embed.embed_item(item, "<ID of the div >"); });
</script>
```

where *urlData* is a *URLSearchData* object and contains the chart request parameters (see above).

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**GET /api/v2_0/getConnection**

API endpoint to get the user's connections as entity addresses ordered from newest to oldest.

**Example request**

```
{
    "type": "GetConnectionRequest",
}
```

**Example response**

This "GetConnectionResponse" message indicates that the user had access rights to retrieve four entity addresses owned by three different users.

```
{
    "type": "GetConnectionResponse",
    "connections": [
        "ea1.2018-06.io.flexmeasures.company:3:4",
        "ea1.2018-06.io.flexmeasures.company:8:3",
        "ea1.2018-06.io.flexmeasures.company:9:2",
        "ea1.2018-06.io.flexmeasures.company:3:1"
    ],
    "names": [
        "CS 4",
        "CS 3",
        "CS 2",
```

(continues on next page)

(continued from previous page)

```
        "CS 1"
    ]
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v2_0/getDeviceMessage**

API endpoint to get device message.

**Optional fields**

- "duration" (6 hours by default; can be increased to plan further into the future)

**Example request**

This "GetDeviceMessageRequest" message requests targeted consumption for UDI event 203 of device 10 of owner 7.

```
{
    "type": "GetDeviceMessageRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc"
}
```

**Example response**

This "GetDeviceMessageResponse" message indicates that the target for UDI event 203 is to consume at various power rates from 10am UTC onwards for a duration of 45 minutes.

```
{
    "type": "GetDeviceMessageResponse",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "values": [
        2.15,
        3,
        2
    ],
    "start": "2015-06-02T10:00:00+00:00",
    "duration": "PT45M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_DOMAIN, INVALID_UNIT, UNKNOWN_SCHEDULE, UNRECOGNIZED_CONNECTION_GROUP, or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**`GET /api/v2_0/getMeterData`**

API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
```

(continues on next page)

(continued from previous page)

```
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v2_0/getPrognosis**

API endpoint to get prognosis.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetPrognosisRequest" message requests prognosed consumption between 0.00am and 1.30am for charging station 1, with a rolling horizon of 6 hours before the end of each 15 minute time interval.

```
{
    "type": "GetPrognosisRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT6H",
    "resolution": "PT15M",
    "unit": "MW"
}
```

**Example response**

This "GetPrognosisResponse" message indicates that a prognosis of consumption for charging station 1 was available 6 hours before the start of each 15 minute time interval.


```
{
    "type": "GetPrognosisResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`GET /api/v2_0/getService`**

API endpoint to get a service listing for this version.

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED

**`POST /api/v2_0/postMeterData`**

API endpoint to post meter data.

**Optional fields**

- "horizon" (see *Prognoses*)
- "prior" (see *Prognoses*)

**Example request**

This "PostMeterDataRequest" message posts measured consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production).


```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                306.66,
                0,
                0,
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2021-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```


It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostMeterDataResponse" message indicates that the measurement has been processed without any error.


```
{
    "type": "PostMeterDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v2_0/postPriceData`**

API endpoint to post price data.

**Optional fields**

- "horizon" (see *Prognoses*)
- "prior" (see *Prognoses*)

**Example request**

This "PostPriceDataRequest" message posts prices for hourly intervals between midnight and midnight the next day for the EPEX SPOT day-ahead auction. The prior indicates that the prices were published at 1pm on December 31st 2020.


```
{
    "type": "PostPriceDataRequest",
    "market": "ea1.2018-06.localhost:epex_da",
    "values": [
        52.37,
        51.14,
        49.09,
        48.35,
        48.47,
        49.98,
        58.7,
        67.76,
        69.21,
        70.26,
        70.46,
        70,
        70.7,
        70.41,
        70,
        64.53,
        65.92,
        69.72,
        70.51,
        75.49,
        70.35,
        70.01,
        66.98,
        58.61
    ],
    "start": "2021-01-01T00:00:00+01:00",
```


(continues on next page)

(continued from previous page)

```
    "duration": "PT24H",
    "prior": "2020-12-31T13:00:00+01:00",
    "unit": "EUR/MWh"
}
```

**Example response**

This "PostPriceDataResponse" message indicates that the prices have been processed without any error.

```
{
    "type": "PostPriceDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_MARKET
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v2_0/postPrognosis`**

API endpoint to post prognoses about meter data.

**Optional fields**

- "horizon" (see *Prognoses*)
- "prior" (see *Prognoses*)

**Example request**

This "PostPrognosisRequest" message posts prognosed consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production), prognosed at 6pm the previous day.

```
{
    "type": "PostPrognosisRequest",
    "groups": [
        {
            "connections": [
                "ea1.2018-06.localhost:1:3",
                "ea1.2018-06.localhost:1:4"
```

(continues on next page)

(continued from previous page)


```
            ],
            "values": [
                300,
                300,
                300,
                0,
                0,
                300
            ]
        },
        {
            "connections": [
                "ea1.2018-06.localhost:1:5"
            ],
            "values": [
                300,
                0,
                0,
                0,
                300,
                300
            ]
        }
    ],
    "start": "2021-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "prior": "2020-12-31T18:00:00Z",
    "unit": "MW"
}
```


It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostPrognosisResponse" message indicates that the prognosis has been processed without any error.


```
{
    "type": "PostPrognosisResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP

- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v2_0/postUdiEvent**

API endpoint to post UDI event.

**Example request A**

This "PostUdiEventRequest" message posts a state of charge (soc) of 12.1 kWh at 10.00am as UDI event 203 of device 10 of owner 7.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "value": 12.1,
    "unit": "kWh",
    "datetime": "2015-06-02T10:00:00+00:00"
}
```

**Example request B**

This "PostUdiEventRequest" message posts a state of charge (soc) of 12.1 kWh at 10.00am, and a target state of charge of 25 kWh at 4.00pm, as UDI event 204 of device 10 of owner 7.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:204:soc-with-targets",
    "value": 12.1,
    "unit": "kWh",
    "datetime": "2015-06-02T10:00:00+00:00",
    "targets": [
        {
            "value": 25,
            "datetime": "2015-06-02T16:00:00+00:00"
        }
    ]
}
```

**Example response**

This "PostUdiEventResponse" message indicates that the UDI event has been processed without any error.

```
{
    "type": "PostUdiEventResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INCOMPLETE_UDI_EVENT, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_DATETIME, INVALID_DOMAIN, INVALID_UNIT, OUTDATED_UDI_EVENT, PTUS_INCOMPLETE, OUTDATED_UDI_EVENT or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v2_0/postWeatherData**

API endpoint to post weather data, such as:

- "radiation" (with kW/m$^2$ as unit)
- "temperature" (with °C as unit)
- "wind_speed" (with m/s as unit)

The sensor type is part of the unique entity address for each sensor, together with the sensor's latitude and longitude.

**Optional fields**

- "horizon" (see *Prognoses*)
- "prior" (see *Prognoses*)

**Example request**

This "PostWeatherDataRequest" message posts temperature forecasts for 15-minute intervals between 3.00pm and 4.30pm for a weather sensor located at latitude 33.4843866 and longitude 126.477859. The forecasts were made at noon.

```
{
    "type": "PostWeatherDataRequest",
    "groups": [
        {
            "sensor": "ea1.2018-06.localhost:temperature:33.4843866:126.477859",
            "values": [
                20.04,
                20.23,
                20.41,
                20.51,
                20.55,
                20.57
            ]
        }
    ],
    "start": "2021-01-01T15:00:00+09:00",
    "duration": "PT1H30M",
    "prior": "2021-01-01T12:00:00+09:00",
    "unit": "°C"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostWeatherDataResponse" message indicates that the forecast has been processed without any error.

```
{
    "type": "PostWeatherDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_SENSOR
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v2_0/user/** (*id*)

API endpoint to get a user.

This endpoint gets a user. Only admins or the user themselves can use this endpoint.

**Example response**

```
{
    'active': True,
    'email': 'test_prosumer@seita.nl',
    'flexmeasures_roles': [1, 3],
    'id': 1,
    'timezone': 'Europe/Amsterdam',
    'username': 'Test Prosumer'
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED

- 403 Forbidden – INVALID_SENDER

**PATCH /api/v2_0/user/**(*id*)

API endpoint to patch user data.

This endpoint sets data for an existing user. Any subset of user fields can be sent. Only the user themselves or admins are allowed to update its data, while a non-admin can only edit a few of their own fields.

Several fields are not allowed to be updated, e.g. id. They are ignored.

**Example request**

```
{
    "active": false,
}
```

**Example response**

The whole user is returned in the response:

```
{
    'active': True,
    'email': 'test_prosumer@seita.nl',
    'flexmeasures_roles': [1, 3],
    'id': 1,
    'timezone': 'Europe/Amsterdam',
    'username': 'Test Prosumer'
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – UPDATED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**PATCH /api/v2_0/user/**(*id*)**/password-reset**

API endpoint to reset the user password. They'll get an email to choose a new password.

Reset the user's password, and send them instructions on how to reset the password. This endoint is useful from a security standpoint, in case of worries the password might be compromised. It sets the current password to something random, invalidates cookies and auth tokens, and also sends an email for resetting the password to the user.

Only admins can use this endpoint.

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST, REQUIRED_INFO_MISSING, UNEXPECTED_PARAMS
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

**GET /api/v2_0/users**

API endpoint to get users.

This endpoint returns all accessible users. By default, only active users are returned. The *include_inactive* query parameter can be used to also fetch inactive users. Only admins can use this endpoint.

**Example response**

An example of one user being returned:

```
[
    {
        'active': True,
        'email': 'test_prosumer@seita.nl',
        'flexmeasures_roles': [1, 3],
        'id': 1,
        'timezone': 'Europe/Amsterdam',
        'username': 'Test Prosumer'
    }
]
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_REQUEST
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

CHAPTER

# SEVENTEEN

# VERSION 1.3

## 17.1 Summary

| Resource | Operation | Description |
|---|---|---|
| Asset | *GET /api/v1_3/getConnection* | Retrieve entity addresses of connections |
| Control | *GET /api/v1_3/getDeviceMessage* | Download control signal from the platform |
| | *POST /api/v1_3/postUdiEvent* | Upload flexibility constraints to the platform |
| Data | *GET /api/v1_3/getMeterData* | Download meter data from the platform |
| | *GET /api/v1_3/getPrognosis* | Download prognosis from the platform |
| | *POST /api/v1_3/postMeterData* | Upload meter data to the platform |
| | *POST /api/v1_3/postPriceData* | Upload price data to the platform |
| | *POST /api/v1_3/postPrognosis* | Upload prognosis to the platform |
| | *POST /api/v1_3/postWeatherData* | Upload weather data to the platform |
| Public | *GET /api/* | List available API versions |
| | *POST /api/requestAuthToken* | Obtain an authentication token |
| | *GET /api/v1_3/getService* | Obtain a service listing for this version |

## 17.2 API Details

**`GET /api/`**
Public endpoint to list API versions.

**`POST /api/requestAuthToken`**
API endpoint to get a fresh authentication access token. Be aware that this fresh token has a limited lifetime (which depends on the current system setting SECURITY_TOKEN_MAX_AGE).

Pass the *email* parameter to identify the user. Pass the *password* parameter to authenticate the user (if not already authenticated in current session)

**`GET /api/v1_3/getConnection`**
API endpoint to get the user's connections as entity addresses ordered from newest to oldest.

**Example request**

```
{
    "type": "GetConnectionRequest",
}
```

**Example response**

This "GetConnectionResponse" message indicates that the user had access rights to retrieve four entity addresses owned by three different users.

```
{
    "type": "GetConnectionResponse",
    "connections": [
        "ea1.2018-06.io.flexmeasures.company:3:4",
        "ea1.2018-06.io.flexmeasures.company:8:3",
        "ea1.2018-06.io.flexmeasures.company:9:2",
        "ea1.2018-06.io.flexmeasures.company:3:1"
    ],
    "names": [
        "CS 4",
        "CS 3",
        "CS 2",
        "CS 1"
    ]
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_3/getDeviceMessage**

API endpoint to get device message.

**Optional fields**

- "duration" (6 hours by default; can be increased to plan further into the future)

**Example request**

This "GetDeviceMessageRequest" message requests targeted consumption for UDI event 203 of device 10 of owner 7.

```
{
    "type": "GetDeviceMessageRequest",
```

(continues on next page)

(continued from previous page)

```
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc"
}
```

**Example response**

This "GetDeviceMessageResponse" message indicates that the target for UDI event 203 is to consume at various power rates from 10am UTC onwards for a duration of 45 minutes.

```
{
    "type": "GetDeviceMessageResponse",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "values": [
        2.15,
        3,
        2
    ],
    "start": "2015-06-02T10:00:00+00:00",
    "duration": "PT45M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_DOMAIN, INVALID_UNIT, UNKNOWN_SCHEDULE, UNRECOGNIZED_CONNECTION_GROUP, or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**GET /api/v1_3/getMeterData**

API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_3/getPrognosis**

API endpoint to get prognosis.

**Optional fields**

- "resolution" (see *Resolutions*)

- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetPrognosisRequest" message requests prognosed consumption between 0.00am and 1.30am for charging station 1, with a rolling horizon of 6 hours before the end of each 15 minute time interval.

```
{
    "type": "GetPrognosisRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT6H",
    "resolution": "PT15M",
    "unit": "MW"
}
```

**Example response**

This "GetPrognosisResponse" message indicates that a prognosis of consumption for charging station 1 was available 6 hours before the start of each 15 minute time interval.

```
{
    "type": "GetPrognosisResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_3/getService**

API endpoint to get a service listing for this version.

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED

**POST /api/v1_3/postMeterData**

API endpoint to post meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostMeterDataRequest" message posts measured consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production).

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                306.66,
                0,
                0,
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostMeterDataResponse" message indicates that the measurement has been processed without any error.

```
{
    "type": "PostMeterDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v1_3/postPriceData**

API endpoint to post price data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPriceDataRequest" message posts prices for hourly intervals between midnight and midnight the next day for the EPEX SPOT day-ahead auction. The horizon indicates that the prices were published at 1pm on December 31st 2014 (i.e. 35 hours ahead of midnight the next day).

```
{
    "type": "PostPriceDataRequest",
    "market": "ea1.2018-06.localhost:epex_da",
    "values": [
        52.37,
        51.14,
        49.09,
        48.35,
        48.47,
        49.98,
        58.7,
        67.76,
        69.21,
        70.26,
```

(continues on next page)

(continued from previous page)

```
        70.46,
        70,
        70.7,
        70.41,
        70,
        64.53,
        65.92,
        69.72,
        70.51,
        75.49,
        70.35,
        70.01,
        66.98,
        58.61
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT24H",
    "horizon": "PT35H",
    "unit": "EUR/MWh"
}
```

**Example response**

This "PostPriceDataResponse" message indicates that the prices have been processed without any error.

```
{
    "type": "PostPriceDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_MARKET
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v1_3/postPrognosis**

API endpoint to post prognoses about meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPrognosisRequest" message posts prognosed consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production), prognosed at 6pm the previous day.


```
{
    "type": "PostPrognosisRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                300,
                300,
                300,
                0,
                0,
                300
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                300,
                0,
                0,
                0,
                300,
                300
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT7H30M",
    "unit": "MW"
}
```


It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostPrognosisResponse" message indicates that the prognosis has been processed without any error.


```
{
    "type": "PostPrognosisResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_3/postUdiEvent`**

API endpoint to post UDI event.

**Example request A**

This "PostUdiEventRequest" message posts a state of charge (soc) of 12.1 kWh at 10.00am as UDI event 203 of device 10 of owner 7.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "value": 12.1,
    "unit": "kWh",
    "datetime": "2015-06-02T10:00:00+00:00"
}
```

**Example request B**

This "PostUdiEventRequest" message posts a state of charge (soc) of 12.1 kWh at 10.00am, and a target state of charge of 25 kWh at 4.00pm, as UDI event 204 of device 10 of owner 7.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:204:soc-with-targets",
    "value": 12.1,
    "unit": "kWh",
    "datetime": "2015-06-02T10:00:00+00:00",
    "targets": [
        {
            "value": 25,
            "datetime": "2015-06-02T16:00:00+00:00"
        }
    ]
}
```

**Example response**

This "PostUdiEventResponse" message indicates that the UDI event has been processed without any error.

```
{
    "type": "PostUdiEventResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INCOMPLETE_UDI_EVENT, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_DATETIME, INVALID_DOMAIN, INVALID_UNIT, OUTDATED_UDI_EVENT, PTUS_INCOMPLETE, OUTDATED_UDI_EVENT or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_3/postWeatherData`**

API endpoint to post weather data, such as:

- "radiation" (with kW/m$^2$ as unit)
- "temperature" (with °C as unit)
- "wind_speed" (with m/s as unit)

The sensor type is part of the unique entity address for each sensor, together with the sensor's latitude and longitude.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostWeatherDataRequest" message posts temperature forecasts for 15-minute intervals between 3.00pm and 4.30pm for a weather sensor located at latitude 33.4843866 and longitude 126.477859. The forecasts were made at noon.

```
{
    "type": "PostWeatherDataRequest",
    "groups": [
        {
            "sensor": "ea1.2018-06.localhost:temperature:33.4843866:126.477859",
            "values": [
                20.04,
                20.23,
                20.41,
                20.51,
                20.55,
                20.57
            ]
        }
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT1H30M",
```

(continues on next page)

(continued from previous page)

```
    "horizon": "PT3H",
    "unit": "°C"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostWeatherDataResponse" message indicates that the forecast has been processed without any error.

```
{
    "type": "PostWeatherDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_SENSOR
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

CHAPTER

# EIGHTEEN

# VERSION 1.2

## 18.1 Summary

| Resource | Operation | Description |
|---|---|---|
| Asset | *GET /api/v1_2/getConnection* | Retrieve entity addresses of connections |
| Control | *GET /api/v1_2/getDeviceMessage* | Download control signal from the platform |
| | *POST /api/v1_2/postUdiEvent* | Upload flexibility constraints to the platform |
| Data | *GET /api/v1_2/getMeterData* | Download meter data from the platform |
| | *GET /api/v1_2/getPrognosis* | Download prognosis from the platform |
| | *POST /api/v1_2/postMeterData* | Upload meter data to the platform |
| | *POST /api/v1_2/postPriceData* | Upload price data to the platform |
| | *POST /api/v1_2/postPrognosis* | Upload prognosis to the platform |
| | *POST /api/v1_2/postWeatherData* | Upload weather data to the platform |
| Public | *GET /api/* | List available API versions |
| | *POST /api/requestAuthToken* | Obtain an authentication token |
| | *GET /api/v1_2/getService* | Obtain a service listing for this version |

## 18.2 API Details

**`GET /api/`**
Public endpoint to list API versions.

**`POST /api/requestAuthToken`**
API endpoint to get a fresh authentication access token. Be aware that this fresh token has a limited lifetime (which depends on the current system setting SECURITY_TOKEN_MAX_AGE).

Pass the *email* parameter to identify the user. Pass the *password* parameter to authenticate the user (if not already authenticated in current session)

**`GET /api/v1_2/getConnection`**
API endpoint to get the user's connections as entity addresses ordered from newest to oldest.

**Example request**

```
{
    "type": "GetConnectionRequest",
}
```

**Example response**

This "GetConnectionResponse" message indicates that the user had access rights to retrieve four entity addresses owned by three different users.

```
{
    "type": "GetConnectionResponse",
    "connections": [
        "ea1.2018-06.io.flexmeasures.company:3:4",
        "ea1.2018-06.io.flexmeasures.company:8:3",
        "ea1.2018-06.io.flexmeasures.company:9:2",
        "ea1.2018-06.io.flexmeasures.company:3:1"
    ],
    "names": [
        "CS 4",
        "CS 3",
        "CS 2",
        "CS 1"
    ]
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_2/getDeviceMessage**

API endpoint to get device message.

**Optional fields**

- "duration" (6 hours by default; can be increased to plan further into the future)

**Example request**

This "GetDeviceMessageRequest" message requests targeted consumption for UDI event 203 of device 10 of owner 7.

```
{
    "type": "GetDeviceMessageRequest",
```

(continues on next page)

(continued from previous page)

```
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc"
}
```

**Example response**

This "GetDeviceMessageResponse" message indicates that the target for UDI event 203 is to consume at various power rates from 10am UTC onwards for a duration of 45 minutes.

```
{
    "type": "GetDeviceMessageResponse",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "values": [
        2.15,
        3,
        2
    ],
    "start": "2015-06-02T10:00:00+00:00",
    "duration": "PT45M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, UNKNOWN_PRICES, UNRECOGNIZED_CONNECTION_GROUP, or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD
- 422 Unprocessable Entity – UNPROCESSABLE_ENTITY

**GET /api/v1_2/getMeterData**

API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`GET /api/v1_2/getPrognosis`**

API endpoint to get prognosis.

**Optional fields**

- "resolution" (see *Resolutions*)

- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetPrognosisRequest" message requests prognosed consumption between 0.00am and 1.30am for charging station 1, with a rolling horizon of 6 hours before the end of each 15 minute time interval.

```
{
    "type": "GetPrognosisRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT6H",
    "resolution": "PT15M",
    "unit": "MW"
}
```

**Example response**

This "GetPrognosisResponse" message indicates that a prognosis of consumption for charging station 1 was available 6 hours before the start of each 15 minute time interval.

```
{
    "type": "GetPrognosisResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER

- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_2/getService**

API endpoint to get a service listing for this version.

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED

**POST /api/v1_2/postMeterData**

API endpoint to post meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostMeterDataRequest" message posts measured consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production).

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                306.66,
                0,
                0,
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostMeterDataResponse" message indicates that the measurement has been processed without any error.

```
{
    "type": "PostMeterDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v1_2/postPriceData**

API endpoint to post price data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPriceDataRequest" message posts prices for hourly intervals between midnight and midnight the next day for the EPEX SPOT day-ahead auction. The horizon indicates that the prices were published at 1pm on December 31st 2014 (i.e. 35 hours ahead of midnight the next day).

```
{
    "type": "PostPriceDataRequest",
    "market": "ea1.2018-06.localhost:epex_da",
    "values": [
        52.37,
        51.14,
        49.09,
        48.35,
        48.47,
        49.98,
        58.7,
        67.76,
        69.21,
        70.26,
```

(continues on next page)

(continued from previous page)

```
        70.46,
        70,
        70.7,
        70.41,
        70,
        64.53,
        65.92,
        69.72,
        70.51,
        75.49,
        70.35,
        70.01,
        66.98,
        58.61
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT24H",
    "horizon": "PT35H",
    "unit": "EUR/MWh"
}
```

**Example response**

This "PostPriceDataResponse" message indicates that the prices have been processed without any error.

```
{
    "type": "PostPriceDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_MARKET
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_2/postPrognosis`**

API endpoint to post prognoses about meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPrognosisRequest" message posts prognosed consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production), prognosed at 6pm the previous day.


```
{
    "type": "PostPrognosisRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                300,
                300,
                300,
                0,
                0,
                300
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                300,
                0,
                0,
                0,
                300,
                300
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT7H30M",
    "unit": "MW"
}
```


It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostPrognosisResponse" message indicates that the prognosis has been processed without any error.


```
{
    "type": "PostPrognosisResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_2/postUdiEvent`**

API endpoint to post UDI event.

**Example request**

This "PostUdiEventRequest" message posts a state of charge (soc) of 12.1 kWh at 10.00am as UDI event 203 of device 10 of owner 7.

```
{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
    "value": 12.1,
    "unit": "kWh",
    "datetime": "2015-06-02T10:00:00+00:00",
}
```

**Example response**

This "PostUdiEventResponse" message indicates that the UDI event has been processed without any error.

```
{
    "type": "PostUdiEventResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_DATETIME, INVALID_UNIT, PTUS_INCOMPLETE, OUTDATED_UDI_EVENT or UNRECOGNIZED_UDI_EVENT
- 401 Unauthorized – UNAUTHORIZED

- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v1_2/postWeatherData**

API endpoint to post weather data, such as:

- "radiation" (with kW/m$^2$ as unit)
- "temperature" (with °C as unit)
- "wind_speed" (with m/s as unit)

The sensor type is part of the unique entity address for each sensor, together with the sensor's latitude and longitude.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostWeatherDataRequest" message posts temperature forecasts for 15-minute intervals between 3.00pm and 4.30pm for a weather sensor located at latitude 33.4843866 and longitude 126.477859. The forecasts were made at noon.

```
{
    "type": "PostWeatherDataRequest",
    "groups": [
        {
            "sensor": "ea1.2018-06.localhost:temperature:33.4843866:126.477859",
            "values": [
                20.04,
                20.23,
                20.41,
                20.51,
                20.55,
                20.57
            ]
        }
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT1H30M",
    "horizon": "PT3H",
    "unit": "°C"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostWeatherDataResponse" message indicates that the forecast has been processed without any error.

```
{
    "type": "PostWeatherDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token

- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_SENSOR
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

CHAPTER

# NINETEEN

# VERSION 1.1

## 19.1 Summary

| Resource | Operation | Description |
|---|---|---|
| Asset | *GET /api/v1_1/getConnection* | Retrieve entity addresses of connections |
| Data | *GET /api/v1_1/getMeterData* | Download meter data from the platform |
| | *GET /api/v1_1/getPrognosis* | Download prognosis from the platform |
| | *POST /api/v1_1/postMeterData* | Upload meter data to the platform |
| | *POST /api/v1_1/postPriceData* | Upload price data to the platform |
| | *POST /api/v1_1/postPrognosis* | Upload prognosis to the platform |
| | *POST /api/v1_1/postWeatherData* | Upload weather data to the platform |
| Public | *GET /api/* | List available API versions |
| | *POST /api/requestAuthToken* | Obtain an authentication token |
| | *GET /api/v1_1/getService* | Obtain a service listing for this version |

## 19.2 API Details

**`GET /api/`**
Public endpoint to list API versions.

**`POST /api/requestAuthToken`**
API endpoint to get a fresh authentication access token. Be aware that this fresh token has a limited lifetime (which depends on the current system setting SECURITY_TOKEN_MAX_AGE).

Pass the *email* parameter to identify the user. Pass the *password* parameter to authenticate the user (if not already authenticated in current session)

**`GET /api/v1_1/getConnection`**
API endpoint to get the user's connections as entity addresses ordered from newest to oldest.

**Example request**

```
{
    "type": "GetConnectionRequest",
}
```

**Example response**

This "GetConnectionResponse" message indicates that the user had access rights to retrieve four entity addresses owned by three different users.

```
{
    "type": "GetConnectionResponse",
    "connections": [
        "ea1.2018-06.io.flexmeasures.company:3:4",
        "ea1.2018-06.io.flexmeasures.company:8:3",
        "ea1.2018-06.io.flexmeasures.company:9:2",
        "ea1.2018-06.io.flexmeasures.company:3:1"
    ],
    "names": [
        "CS 4",
        "CS 3",
        "CS 2",
        "CS 1"
    ]
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_1/getMeterData**

API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
```

(continues on next page)

(continued from previous page)

```
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_1/getPrognosis**

API endpoint to get prognosis.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetPrognosisRequest" message requests prognosed consumption between 0.00am and 1.30am for charging station 1, with a rolling horizon of 6 hours before the end of each 15 minute time interval.

```
{
    "type": "GetPrognosisRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT6H",
    "resolution": "PT15M",
    "unit": "MW"
}
```

**Example response**

This "GetPrognosisResponse" message indicates that a prognosis of consumption for charging station 1 was available 6 hours before the start of each 15 minute time interval.

```
{
    "type": "GetPrognosisResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1_1/getService**

API endpoint to get a service listing for this version.

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED

**POST /api/v1_1/postMeterData**

API endpoint to post meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostMeterDataRequest" message posts measured consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production).

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                306.66,
                0,
                0,
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostMeterDataResponse" message indicates that the measurement has been processed without any error.

```
{
    "type": "PostMeterDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**POST /api/v1_1/postPriceData**

API endpoint to post price data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPriceDataRequest" message posts prices for hourly intervals between midnight and midnight the next day for the EPEX SPOT day-ahead auction. The horizon indicates that the prices were published at 1pm on December 31st 2014 (i.e. 35 hours ahead of midnight the next day).

```
{
    "type": "PostPriceDataRequest",
    "market": "ea1.2018-06.localhost:epex_da",
    "values": [
        52.37,
        51.14,
        49.09,
        48.35,
        48.47,
        49.98,
        58.7,
        67.76,
        69.21,
        70.26,
        70.46,
        70,
        70.7,
        70.41,
```

(continues on next page)

(continued from previous page)

```
        70,
        64.53,
        65.92,
        69.72,
        70.51,
        75.49,
        70.35,
        70.01,
        66.98,
        58.61
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT24H",
    "horizon": "PT35H",
    "unit": "EUR/MWh"
}
```

**Example response**

This "PostPriceDataResponse" message indicates that the prices have been processed without any error.

```
{
    "type": "PostPriceDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_MARKET
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_1/postPrognosis`**

API endpoint to post prognoses about meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostPrognosisRequest" message posts prognosed consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production), prognosed at 6pm the previous day.


```
{
    "type": "PostPrognosisRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                300,
                300,
                300,
                0,
                0,
                300
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                300,
                0,
                0,
                0,
                300,
                300
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "horizon": "PT7H30M",
    "unit": "MW"
}
```


It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostPrognosisResponse" message indicates that the prognosis has been processed without any error.


```
{
    "type": "PostPrognosisResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```


**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1_1/postWeatherData`**

API endpoint to post weather data, such as:

- "radiation" (with kW/m$^2$ as unit)
- "temperature" (with °C as unit)
- "wind_speed" (with m/s as unit)

The sensor type is part of the unique entity address for each sensor, together with the sensor's latitude and longitude.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostWeatherDataRequest" message posts temperature forecasts for 15-minute intervals between 3.00pm and 4.30pm for a weather sensor located at latitude 33.4843866 and longitude 126.477859. The forecasts were made at noon.

```
{
    "type": "PostWeatherDataRequest",
    "groups": [
        {
            "sensor": "ea1.2018-06.localhost:temperature:33.4843866:126.477859",
            "values": [
                20.04,
                20.23,
                20.41,
                20.51,
                20.55,
                20.57
            ]
        }
    ],
    "start": "2015-01-01T15:00:00+09:00",
    "duration": "PT1H30M",
    "horizon": "PT3H",
    "unit": "°C"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostWeatherDataResponse" message indicates that the forecast has been processed without any error.

```
{
    "type": "PostWeatherDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_SENSOR
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

CHAPTER

# TWENTY

# VERSION 1.0

## 20.1 Summary

| Resource | Operation | Description |
|---|---|---|
| Data | *GET /api/v1/getMeterData* | Download meter data from the platform |
| | *POST /api/v1/getMeterData* | |
| | *POST /api/v1/postMeterData* | Upload meter data to the platform |
| Public | *GET /api/* | List available API versions |
| | *POST /api/requestAuthToken* | Obtain an authentication token |
| | *GET /api/v1/getService* | Obtain a service listing for this version |

## 20.2 API Details

**`GET /api/`**
Public endpoint to list API versions.

**`POST /api/requestAuthToken`**
API endpoint to get a fresh authentication access token. Be aware that this fresh token has a limited lifetime (which depends on the current system setting SECURITY_TOKEN_MAX_AGE).

Pass the *email* parameter to identify the user. Pass the *password* parameter to authenticate the user (if not already authenticated in current session)

**`GET /api/v1/getMeterData`**
API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)
- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**`POST /api/v1/getMeterData`**

API endpoint to get meter data.

**Optional fields**

- "resolution" (see *Resolutions*)
- "horizon" (see *Beliefs*)

- "prior" (see *Beliefs*)
- "source" (see *Sources*)

**Example request**

This "GetMeterDataRequest" message requests measured consumption between 0.00am and 1.30am for charging station 1.

```
{
    "type": "GetMeterDataRequest",
    "connection": "CS 1",
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Example response**

This "GetMeterDataResponse" message indicates that consumption for charging station 1 was measured in 15-minute intervals.

```
{
    "type": "GetMeterDataResponse",
    "connection": "CS 1",
    "values": [
        306.66,
        306.66,
        0,
        0,
        306.66,
        306.66
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_SOURCE, INVALID_TIMEZONE, INVALID_UNIT, UNRECOGNIZED_ASSET, or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

**GET /api/v1/getService**

API endpoint to get a service listing for this version.

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED

**POST /api/v1/postMeterData**

API endpoint to post meter data.

**Optional fields**

- "horizon" (see *Prognoses*)

**Example request**

This "PostMeterDataRequest" message posts measured consumption for 15-minute intervals between 0.00am and 1.30am for charging stations 1, 2 and 3 (negative values denote production).

```
{
    "type": "PostMeterDataRequest",
    "groups": [
        {
            "connections": [
                "CS 1",
                "CS 3"
            ],
            "values": [
                306.66,
                306.66,
                0,
                0,
                306.66,
                306.66
            ]
        },
        {
            "connections": [
                "CS 2"
            ],
            "values": [
                306.66,
                0,
                0,
                0,
                306.66,
                306.66
            ]
        }
    ],
    "start": "2015-01-01T00:00:00Z",
    "duration": "PT1H30M",
    "unit": "MW"
}
```

It is allowed to send higher resolutions (in this example for instance, 30 minutes) which will be upsampled.

**Example response**

This "PostMeterDataResponse" message indicates that the measurement has been processed without any error.

```
{
    "type": "PostMeterDataResponse",
    "status": "PROCESSED",
    "message": "Request has been processed."
}
```

**Request Headers**

- Authorization – The authentication token
- Content-Type – application/json

**Response Headers**

- Content-Type – application/json

**Status Codes**

- 200 OK – PROCESSED
- 400 Bad Request – INVALID_DOMAIN, INVALID_MESSAGE_TYPE, INVALID_TIMEZONE, INVALID_UNIT, REQUIRED_INFO_MISSING, UNRECOGNIZED_ASSET or UNRECOGNIZED_CONNECTION_GROUP
- 401 Unauthorized – UNAUTHORIZED
- 403 Forbidden – INVALID_SENDER
- 405 Method Not Allowed – INVALID_METHOD

CHAPTER

# TWENTYONE

# API CHANGE LOG

## 21.1 v2.0 | 2021-03-23

- [**Breaking change**] Switched the interpretation of horizons to rolling horizons.
- [**Breaking change**] Deprecated the use of ISO 8601 repeating time intervals to denote rolling horizons.
- [**Breaking change**] Deprecated the automatic inference of horizons for *postMeterData*, *postPrognosis*, *postPriceData* and *postWeatherData* endpoints for API version below v2.0.
- Introduced the "prior" field for *postMeterData*, *postPrognosis*, *postPriceData* and *postWeatherData* endpoints.
- Changed the Introduction section:
  - Rewrote the subsection on prognoses to explain the horizon and prior fields.
- Changed the Simulation section:
  - Rewrote relevant examples using horizon and prior fields.

## 21.2 v2.0 | 2021-02-19

- REST endpoints for managing users: */users/* (GET), */user/<id>* (GET, PATCH) and */user/<id>/password-reset* (PATCH).

## 21.3 v2.0 | 2020-11-14

- REST endpoints for managing assets: */assets/* (GET, POST) and */asset/<id>* (GET, PATCH, DELETE).

## 21.4 v1.3-7 | 2020-12-16

*Affects all versions since v1.0.*

- Separated the dual purpose of the "horizon" field in the *getMeterData* and *getPrognosis* endpoints by introducing the "prior" field:
  - The "horizon" field in GET endpoints is now always interpreted as a rolling horizon, regardless of whether it is stated as an ISO 8601 repeating time interval.

  - The *getMeterData* and *getPrognosis* endpoints now accept an optional "prior" field to select only data recorded before a certain ISO 8601 timestamp (replacing the unintuitive usage of the horizon field for specifying a latest time of belief).

## 21.5 v1.3-6 | 2020-12-11

*Affects all versions since v1.0.*

- The *getMeterData* and *getPrognosis* endpoints now return the INVALID_SOURCE status 400 response in case the optional "source" field is used and no relevant sources can be found.

## 21.6 v1.3-5 | 2020-10-29

*Affects all versions since v1.0.*

- Endpoints to POST meter data will now check incoming data to see if the required asset's resolution is being used — upsampling is done if possible. These endpoints can now return the REQUIRED_INFO_MISSING status 400 response.
- Endpoints to GET meter data will return data in the asset's resolution — downsampling to the "resolution" field is done if possible.
- As they need to determine the asset, all of the mentioned POST and GET endpoints can now return the UNRECOGNIZED_ASSET status 4000 response.

## 21.7 v1.3-4 | 2020-06-18

- Improved support for use cases of the *getDeviceMessage* endpoint in which a longer duration, between posting UDI events and retrieving device messages based on those UDI events, is required; the default *time to live* of UDI event identifiers is prolonged from 500 seconds to 7 days, and can be set as a config variable (*FLEXMEASURES_PLANNING_TTL*)

## 21.8 v1.3-3 | 2020-06-07

- Changed backend support (API specifications unaffected) for scheduling charging stations to scheduling Electric Vehicle Supply Equipment (EVSE), in accordance with the Open Charge Point Interface (OCPI).

## 21.9 v1.3-2 | 2020-03-11

- Fixed example entity addresses in simulation section

## 21.10 v1.3-1 | 2020-02-08

- Backend change: the default planning horizon can now be set in FlexMeasures's configuration (*FLEXMEASURES_PLANNING_HORIZON*)

## 21.11 v1.3-0 | 2020-01-28

- Introduced new event type "soc-with-targets" to support scheduling charging stations (see extra example for the *postUdiEvent* endpoint)
- The *postUdiEvent* endpoint now triggers scheduling jobs to be set up (rather than scheduling directly triggered by the *getDeviceMessage* endpoint)
- The *getDeviceMessage* now queries the job queue and database for an available schedule

## 21.12 v1.2-3 | 2020-01-28

- Updated endpoint descriptions with additional possible status 400 responses:
  - INVALID_DOMAIN for invalid entity addresses
  - UNKNOWN_PRICES for infeasible schedules due to missing prices

## 21.13 v1.2-2 | 2018-10-08

- Added a list of registered types of weather sensors to the Simulation section and *postWeatherData* endpoint
- Changed example for the *postPriceData* endpoint to reflect Korean situation

## 21.14 v1.2-1 | 2018-09-24

- Added a local table of contents to the Simulation section
- Added a description of the *postPriceData* endpoint in the Simulation section
- Added a description of the *postWeatherData* endpoint in the Simulation section
- Revised the subsection about posting power data in the Simulation section
- Revised the entity address for UDI events to include the type of the event

```
i.e.

{
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203:soc",
}

rather than the erroneously double-keyed:

{
```

(continues on next page)

(continued from previous page)

```
    "type": "PostUdiEventRequest",
    "event": "ea1.2018-06.io.flexmeasures.company:7:10:203",
    "type": "soc"
}
```

## 21.15 v1.2-0 | 2018-09-08

- Added a description of the *postUdiEvent* endpoint in the Prosumer and Simulation sections
- Added a description of the *getDeviceMessage* endpoint in the Prosumer and Simulation sections

## 21.16 v1.1-5 | 2020-06-18

- Fixed the *getConnection* endpoint where the returned list of connection names had been unnecessarily nested

## 21.17 v1.1-4 | 2020-03-11

- Added support for posting daily and weekly prices for the *postPriceData* endpoint

## 21.18 v1.1-3 | 2018-09-08

- Added the Simulation section:
  - Added information about setting up a new simulation
  - Added examples for calling the *postMeterData* endpoint
  - Added example for calling the *getPrognosis* endpoint

## 21.19 v1.1-2 | 2018-08-15

- Added the *postPrognosis* endpoint
- Added the *postPriceData* endpoint
- Added a description of the *postPrognosis* endpoint in the Aggregator section
- Added a description of the *postPriceData* endpoint in the Aggregator and Supplier sections

## 21.20 v1.1-1 | 2018-08-06

- Added the *getConnection* endpoint
- Added the *postWeatherData* endpoint
- Changed the Introduction section:
  - Added information about the sign of power values (production is negative)
  - Updated information about horizons (now anchored to the end of each time interval rather than to the start)
- Added an optional horizon to the *postMeterData* endpoint

## 21.21 v1.1-0 | 2018-07-15

- Added the *getPrognosis* endpoint
- Changed the *getMeterData* endpoint to accept an optional resolution, source, and horizon
- Changed the Introduction section:
  - Added information about timeseries resolutions
  - Added information about sources
  - Updated information about horizons
- Added a description of the *getPrognosis* endpoint in the Supplier section

## 21.22 v1.0-1 | 2018-07-10

- Moved specifications to be part of the platform's Sphinx documentation:
  - Each API service is now documented in the docstring of its respective endpoint
  - Added sections listing all endpoints per version
  - Documentation includes specifications of **all** supported API versions (supported versions have a registered Flask blueprint)

## 21.23 v1.0-0 | 2018-07-10

- Started change log
- Added Introduction section with notes regarding:
  - Authentication
  - Relevant roles for the API
  - Key notation
  - The addressing scheme for assets
  - Connection group notation
  - Timeseries notation

  - Prognosis notation
  - Units of timeseries data
- Added a description of the *getService* endpoint in the Introduction section
- Added a description of the *postMeterData* endpoint in the MDC section
- Added a description of the *getMeterData* endpoint in the Prosumer section

CHAPTER

# TWENTYTWO

# CLI COMMANDS

FlexMeasures comes with a command-line utility, which helps to manage data. Below, we list all available commands.

Each command has more extensive documentation if you call it with `--help`.

We keep track of changes to these commands in *FlexMeasures CLI Changelog*. You can also get the current overview over the commands you have available by:

```
flexmeasures --help
```

This also shows admin commands made available through Flask and installed extensions (such as Flask-Security and Flask-Migrate), of which some are referred to in this documentation.

## 22.1 `add` - Add data

| | |
|---|---|
| `flexmeasures add structure` | Initialize structural data like asset types, market types and weather sensor types. |
| `flexmeasures add user` | Create a FlexMeasures user. |
| `flexmeasures add asset` | Create a new asset. |
| `flexmeasures add weather-sensor` | Add a weather sensor. |
| `flexmeasures add external-weather-forecasts` | Collect weather forecasts from the DarkSky API. |
| `flexmeasures add forecasts` | Create forecasts. |

## 22.2 `delete` - Delete data

| | |
|---|---|
| `flexmeasures delete structure` | Delete all structural (non time-series) data like assets (types), markets (types) and weather sensors (types) and users. |
| `flexmeasures delete user` | Delete a user & also their assets and power measurements. |
| `flexmeasures delete measurements` | Delete measurements (with horizon <= 0). |
| `flexmeasures delete prognoses` | Delete forecasts and schedules (forecasts > 0). |

## 22.3 `jobs` - Job queueing

| | |
|---|---|
| `flexmeasures jobs run-worker` | Start a worker process for forecasting and/or scheduling jobs. |
| `flexmeasures jobs clear-queue` | Clear a job queue. |

## 22.4 `db-ops` - Operations on the whole database

| | |
|---|---|
| `flexmeasures db-ops dump` | Create a dump of all current data (using *pg_dump*). |
| `flexmeasures db-ops load` | Load backed-up contents (see *db-ops save*), run *reset* first. |
| `flexmeasures db-ops reset` | Reset database data and re-create tables from data model. |
| `flexmeasures db-ops restore` | Restore the dump file, see *db-ops dump* (run *reset* first). |
| `flexmeasures db-ops save` | Backup db content to files. |

CHAPTER

TWENTYTHREE

# FLEXMEASURES CLI CHANGELOG

## 23.1 since v0.2.4 | March XX, 2021

- Refactor CLI into the main groups `add`, `delete`, `jobs` and `db-ops`
- Add `flexmeasures add asset`, `flexmeasures add user` and `flexmeasures add weather-sensor`
- split the `populate-db` command into `flexmeasures add structure` and `flexmeasures add forecasts`

CHAPTER

TWENTYFOUR

# DEVELOPING FOR FLEXMEASURES

This page instructs developers who work on FlexMeasures how to set up the development environment. Furthermore, we discuss several guidelines and best practices.

**Table of contents**



## 24.1 Getting started

### 24.1.1 Virtual environment

Using a virtual environment is best practice for Python developers. We also strongly recommend using a dedicated one for your work on FlexMeasures, as our make target (see below) will use `pip-sync` to install dependencies, which could interfere with some libraries you already have installed.

- Make a virtual environment: `python3.8 -m venv flexmeasures-venv` or use a different tool like `mkvirtualenv` or virtualenvwrapper. You can also use an Anaconda distribution as base with `conda create -n flexmeasures-venv python=3.8`.
- Activate it, e.g.: `source flexmeasures-venv/bin/activate`

### 24.1.2 Dependencies

Install all dependencies including the ones needed for development:

```
make install-for-dev
```

### 24.1.3 Configuration

Follow the configuration Quickstart advice in *Getting started* and *Configuration*.

### 24.1.4 Loading data

If you have a SQL Dump file, you can load that:

```
psql -U {user_name} -h {host_name} -d {database_name} -f {file_path}
```

### 24.1.5 Run locally

Now, to start the web application, you can run:

```
flexmeasures run
```

Or:

```
python run-local.py
```

And access the server at http://localhost:5000

## 24.2 Tests

You can run automated tests with:

```
make test
```

which behind the curtains installs dependencies and calls pytest.

A coverage report can be created like this:

```
pytest --cov=flexmeasures --cov-config .coveragerc
```

You can add –cov-report=html after which a htmlcov/index.html is generated.

It's also possible to use:

```
python setup.py test
```

## 24.3 Versioning

We use setuptool_scm for versioning, which bases the FlexMeasures version on the latest git tag and the commits since then.

So as a developer, it's crucial to use git tags for versions only.

We use semantic versioning, and we always include the patch version, not only max and min, so that setuptools_scm makes the correct guess about the next minor version. Thus, we should use `2.0.0` instead of `2.0`.

See `to_pypi.sh` for more commentary on the development versions.

Our API has its own version, which moves much slower. This is important to explicitly support outside apps who were coded against older versions.

## 24.4 Auto-applying formatting and code style suggestions

We use Black to format our Python code and Flake8 to enforce the PEP8 style guide and linting. We also run mypy on many files to do some static type checking.

We do this so real problems are found faster and the discussion about formatting is limited. All of these can be installed by using `pip`, but we recommend using them as a pre-commit hook. To activate that behaviour, do:

```
pip install pre-commit
pre-commit install
```

in your virtual environment.

Now each git commit will first run `flake8`, then `black` and finally `mypy` over the files affected by the commit (`pre-commit` will install these tools into its own structure on the first run).

This is also what happens automatically server-side when code is committed to a branch (via Github Actions), but having those tests locally as well will help you spot these issues faster.

If `flake8`, `black` or `mypy` propose changes to any file, the commit is aborted (saying that it "failed"). The changes proposed by `black` are implemented automatically (you can review them with *git diff*). Some of them might even resolve the `flake8` warnings :)

## 24.5 A hint about using notebooks

If you edit notebooks, make sure results do not end up in git:

```
conda install -c conda-forge nbstripout
nbstripout --install
```

(on Windows, maybe you need to look closer at https://github.com/kynan/nbstripout)

## 24.6 A hint for Unix developers

I added this to my ~/.bashrc, so I only need to type `fm` to get started and have the ssh agent set up, as well as up-to-date code and dependencies in place.

```
addssh(){
    eval `ssh-agent -s`
    ssh-add ~/.ssh/id_bitbucket
}
fm(){
    addssh
    cd ~/workspace/flexmeasures
    git pull  # do not use if any production-like app runs from the git code
    workon flexmeasures-venv  # this depends on how you created your virtual␣
↪environment
    make install-for-dev
}
```

**Note:** All paths depend on your local environment, of course.

CHAPTER

TWENTYFIVE

# HANDLING DATA

This document describes how to get the postgres database ready to use and maintain it (do migrations / changes to the structure).

We also spend a few words on coding with database transactions in mind.

Finally, we'll discuss how FlexMeasures is using Redis and redis-queues. When setting up on Windows, a guide to install the Redis-based queuing system for handling (forecasting) jobs.

**Table of contents**

## 25.1 Getting ready to use

Notes:

- We use postgres 12 at the moment, but any version starting with 9 probably works.
- We assume `flexmeasures` for your database and username here. You can use anything you like, of course.
- The name `flexmeasures_test` for the test database is good to keep this way, as automated tests are looking for that database / user / password.

### 25.1.1 Install

On Unix:

```
sudo apt-get install postgresql
pip install psycopg2-binary
```

On Windows:

- Download postgres here: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
- Install and remember your `postgres` user password
- Add the lib and bin directories to your Windows path: http://bobbyong.com/blog/installing-postgresql-on-windoes/
- `conda install psycopg2`

### 25.1.2 Make sure postgres represents datetimes in UTC timezone

(Otherwise, pandas can get confused with daylight saving time.)

Luckily, many web hosters already have `timezone= 'UTC'` set correctly by default, but local postgres installations often use `timezone='localtime'`.

In any case, check both your local installation and the server, like this:

Find the `postgres.conf` file. Mine is at `/etc/postgresql/9.6/main/postgresql.conf`. You can also type `SHOW config_file;` in a postgres console session (as superuser) to find the config file.

Find the `timezone` setting and set it to 'UTC'.

Then restart the postgres server.

### 25.1.3 Setup the "flexmeasures" Unix user

This may in fact not be needed:

```
sudo /usr/sbin/adduser flexmeasures
```

### 25.1.4 Create “flexmeasures” and “flexmeasures_test” databases and users

From the terminal:

Open a console (use your Windows key and type `cmd`). Proceed to create a database as the postgres superuser (using your postgres user password):

```
sudo -i -u postgres
createdb -U postgres flexmeasures
createdb -U postgres flexmeasures_test
createuser --pwprompt -U postgres flexmeasures      # enter your password
createuser --pwprompt -U postgres flexmeasures_test  # enter "flexmeasures_test" as␣
↪password
exit
```

Or, from within Postgres console:

```
CREATE USER flexmeasures WITH UNENCRYPTED PASSWORD 'this-is-your-secret-choice';
CREATE DATABASE flexmeasures WITH OWNER = flexmeasures;
CREATE USER flexmeasures_test WITH UNENCRYPTED PASSWORD 'flexmeasures_test';
CREATE DATABASE flexmeasures_test WITH OWNER = flexmeasures_test;
```

Log in as the postgres superuser and connect to your newly-created database:

```
sudo -u postgres psql
```

```
\connect flexmeasures
```

Add the following extensions while logged in as the postgres superuser:

```
CREATE EXTENSION cube;
CREATE EXTENSION earthdistance;
```

Connect to the `flexmeasures_test` database and repeat creating these extensions there. Then `exit`.

Finally, try logging in as the flexmeasures user once:

```
psql -U flexmeasures --password -h 127.0.0.1 -d flexmeasures
```

```
\q
```

### 25.1.5 Configure FlexMeasures app for that database

Write:

```
SQLALCHEMY_DATABASE_URI = "postgresql://flexmeasures:<password>@127.0.0.1/flexmeasures
↪"
```

into the config file you are using, e.g. ~/flexmeasures.cfg

### 25.1.6 Get structure (and some data) into place

You need data to enjoy the benefits of FlexMeasures or to develop features for it. In this section, there are some ways to get started.

#### Import from another database

Here is a short recipe to import data from a FlexMeasures database (e.g. a demo database) into your local system.

On the to-be-exported database:

```
flexmeasures db-ops dump
```

**Note:** Only the data gets dumped here.

Then, we create the structure in our database anew, based on the data model given by the local codebase:

```
flexmeasures db-ops reset
```

Then we import the data dump we made earlier:

```
flask db-ops restore <DATABASE DUMP FILENAME>
```

A potential `alembic_version` error should not prevent other data tables from being restored. You can also choose to import a complete db dump into a freshly created database, of course.

**Note:** To make sure passwords will be decrypted correctly when you authenticate, set the same SECURITY_PASSWORD_SALT value in your config as the one that was in use when the dumped passwords were encrypted!

#### Create data manually

First, you can get the database structure with:

```
flexmeasures db upgrade
```

**Note:** If you develop code (and might want to make changes to the data model), you should also check out the maintenance section about database migrations.

You can create users with the `new-user` command. Check it out:

```
flexmeasures add user --help
```

You can create some pre-determined asset types and data sources with this command:

```
flexmeasures add structure
```

**Todo:** We should instead offer CLI commands to be able to create asset types as needed.

You can create assets in the FlexMeasures UI.

**Todo:** Maybe a CLI command would help to script all data creation.

**Todo:** We still need a decent way to load in metering data, e.g. from CSV - often, a custom loading script will be necessary anyways.

You can create forecasts for your existing metered data with this command:

```
flexmeasures add forecasts
```

Check out it's `--help` content to learn more. You can set which assets and which time window you want to forecast. Of course, making forecasts takes a while for a larger dataset. You can also simply queue a job with this command (and run a worker to process the *Redis queue*).

Just to note, there are also commands to get rid of data, such as:

```
flexmeasures delete structure
flexmeasures delete measurements
flexmeasures delete forecasts
```

Check out the *CLI Commands* documentation for more details.

## 25.2 Visualize the data model

You can visualise the data model like this:

```
make show-data-model
```

This will generate a picture based on the model code. You can also generate picture based on the actual database, see inside the Makefile.

## 25.3 Maintenance

Maintenance is supported with the alembic tool. It reacts automatically to almost all changes in the SQLAlchemy code. With alembic, multiple databases, such as development, staging and production databases can be kept in sync.

### 25.3.1 Make first migration

Run these commands from the repository root directory (read below comments first):

```
flexmeasures db init
flexmeasures db migrate
flexmeasures db upgrade
```

The first command (`flexmeasures db init`) is only needed here once, it initialises the alembic migration tool. The second command generates the SQL for your current db model and the third actually gives you the db structure.

With every migration, you get a new migration step in `migrations/versions`. Be sure to add that to `git`, as future calls to `flexmeasures db upgrade` will need those steps, and they might happen on another computer.

Hint: You can edit these migrations steps, if you want.

### 25.3.2 Make another migration

Just to be clear that the `db init` command is needed only at the beginning - you usually do, if your model changed:

```
flexmeasures db migrate --message "Please explain what you did, it helps for later"
flexmeasures db upgrade
```

### 25.3.3 Get database structure updated

The goal is that on any other computer, you can always execute

```
flexmeasures db upgrade
```

to have the database structure up-to-date with all migrations.

### 25.3.4 Working with the migration history

The history of migrations is at your fingertips:

```
flexmeasures db current
flexmeasures db history
```

You can move back and forth through the history:

```
flexmeasures db downgrade
flexmeasures db upgrade
```

Both of these accept a specific revision id parameter, as well.

### 25.3.5 Check out database status

Log in into the database:

```
psql -U flexmeasures --password -h 127.0.0.1 -d flexmeasures
```

with the password from flexmeasures/development_config.py. Check which tables are there:

```
\dt
```

To log out:

```
\q
```

## 25.4 Transaction management

It is really useful (and therefore an industry standard) to bundle certain database actions within a transaction. Transactions are atomic - either the actions in them all run or the transaction gets rolled back. This keeps the database in a sane state and really helps having expectations during debugging.

Please see the package `flexmeasures.data.transactional` for details on how a FlexMeasures developer should make use of this concept. If you are writing a script or a view, you will find there the necessary structural help to bundle your work in a transaction.

## 25.5 Redis queue

FlexMeasures supports jobs (e.g. forecasting) running asynchronously to the main FlexMeasures application using Redis Queue.

It relies on a Redis server, which is has to be installed locally, or used on a separate host. In the latter case, configure *Redis* details in your FlexMeasures config file.

Forecasting jobs are usually created (and enqueued) when new data comes in via the API. To asynchronously work on these forecasting jobs, run this in a console:

```
flexmeasures jobs run_worker --queue forecasting
```

You should be able to run multiple workers in parallel, if necessary. You can add the `--name` argument to keep them a bit more organized.

The FlexMeasures unit tests use fakeredis to simulate this task queueing, with no configuration required.

### 25.5.1 Inspect the queue and jobs

The first option to inspect the state of the `forecasting` queue should be via the formidable RQ dashboard. If you have admin rights, you can access it at `your-flexmeasures-url/rq/`, so for instance `http://localhost:5000/rq/`. You can also start RQ dashboard yourself (but you need to know the redis server credentials):

```
pip install rq-dashboard
rq-dashboard --redis-host my.ip.addr.ess --redis-password secret --redis-database 0
```

RQ dashboard shows you ongoing and failed jobs, and you can see the error messages of the latter, which is very useful.

Finally, you can also inspect the queue and jobs via a console (see the nice RQ documentation), which is more powerful. Here is an example of inspecting the finished jobs and their results:

```
from redis import Redis
from rq import Queue
from rq.job import Job
from rq.registry import FinishedJobRegistry

r = Redis("my.ip.addr.ess", port=6379, password="secret", db=2)
q = Queue("forecasting", connection=r)
finished = FinishedJobRegistry(queue=q)

finished_job_ids = finished.get_job_ids()
```

(continues on next page)

(continued from previous page)

```
print("%d jobs finished successfully." % len(finished_job_ids))

job1 = Job.fetch(finished_job_ids[0], connection=r)
print("Result of job %s: %s" % (job1.id, job1.result))
```

### 25.5.2 Redis queues on Windows

On Unix, the rq system is automatically set up as part of FlexMeasures's main setup (the `rq` dependency).

However, rq is not functional on Windows without the Windows Subsystem for Linux.

On these versions of Windows, FlexMeasures's queuing system uses an extension of Redis Queue called `rq-win`. This is also an automatically installed dependency of FlexMeasures.

However, the Redis server needs to be set up separately. Redis itself does not work on Windows, so it might be easiest to commission a Redis server in the cloud (e.g. on kamatera.com).

If you want to install Redis on Windows itself, it can be set up on a virtual machine as follows:

- Install Vagrant on Windows and VirtualBox
- Download the vagrant-redis vagrant configuration
- Extract `vagrant-redis.zip` in any folder, e.g. in `c:\vagrant-redis`
- Set `config.vm.box = "hashicorp/precise64"` in the Vagrantfile, and remove the line with `config.vm.box_url`
- Run `vagrant up` in Command Prompt
- In case `vagrant up` fails because VT-x is not available, enable it in your bios if you can (more debugging tips here if needed)

CHAPTER

# TWENTYSIX

# DEVELOPING ON THE API

The FlexMeasures API is the main way that third-parties can automate their interaction with FlexMeasures, so it's highly important.

This is a small guide for creating new versions of the API and its docs.

**Todo:** A guide for endpoint design, e.g. Marshmallow, custom USEF-like responses, validators (common code shared between endpoints).

**Table of contents**



## 26.1 Introducting a new API version

Larger changes to the API, other than fixes and refactoring, should be done by creating a new API version. In the guide we're assuming the new version is `v1.1`.

Whether we need a new API version or not, doesn't have a clear set of rules yet. Certainly backward-incopatible changes should require one, but as you'll see, there is also certain iverhead in creating a new version, so a careful trade-off is advised.

**Note:** For the rest of this guide we'll assume your new API version is `v1_1`.

### 26.1.1 Set up new module with routes

In `flexmeasures/api` create a new module (folder with `__init__.py`). Copy over the `routes.py` from the previous API version. By default we import all routes from the previous version:

```
from flexmeasures.api.v1 import routes as v1_routes, implementations as v1_
↪implementations
```

Set the service listing for this version (or overwrite completely if needed):

```
v1_1_service_listing = copy.deepcopy(v1_routes.v1_service_listing)
v1_1_service_listing["version"] = "1.1"
```

Then update and redecorate each API endpoint as follows:

```
@flexmeasures_api.route("/getService", methods=["GET"])
@as_response_type("GetServiceResponse")
@append_doc_of(v1_routes.get_service)
def get_service():
    return v1_implementations.get_service_response(v1_1_service_listing)
```

### 26.1.2 Set up a new blueprint

In the new module's `flexmeasures/api/v1_1/__init.py__`, copy the contents of `flexmeasures/api/v1/__init.py__` (previous API version). Change all references to the version name in the new file (for example: `flexmeasures_api_v1` should become `flexmeasures_api_v1_1`).

In `flexmeasures/api/__init__.py` update the version listing in `get_versions()` and register a blueprint for the new api version by adding:

```
from flexmeasures.api.v1_1 import register_at as v1_1_register_at
v1_1_register_at(app)
```

### 26.1.3 New or updated endpoint implementations

Write functionality of new or updated endpoints in:

```
flexmeasures/api/v1_1/implementations.py
```

Utility functions that are commonly shared between endpoint implementations of different versions should go in:

```
flexmeasures/api/common/utils
```

where we distinguish between response decorators, request validators and other utils.

### 26.1.4 Testing

If you changed an endpoint in the new version, write a test for it. Usually, there is no need to copy the tests for unchanged endpoints, if not a major API version is being released.

Test the entire api or just your new version:

```
pytest -k api
pytest -k v1_1
```

### 26.1.5 UI Crud

In `ui/crud`, we support FlexMeasures' in-built UI with Flask endpoints, which then talk to our internal API. The routes used there point to an API version. You should consider updating them to point to your new version.

### 26.1.6 Documentation

In `documentation/api` start a new specification `v1_1.rst` with contents like this:

```
.. _v1_1:

Version 1.1
===========

Summary
-------

.. qrefflask:: flexmeasures.app:create()
  :blueprints: flexmeasures_api, flexmeasures_api_v1_1
  :order: path
  :include-empty-docstring:

API Details
-----------

.. autoflask:: flexmeasures.app:create()
  :blueprints: flexmeasures_api, flexmeasures_api_v1_1
  :order: path
  :include-empty-docstring:
```

If you are ready to publish the new specifications, enter your changes in `documentation/api/change_log.rst` and update the api toctree in `documentation/index.rst` to include the new version in the table of contents.

You're not done. Several sections in the API documentation list endpoints as examples. If you want other developers to use your new API version, make sure those examples reference the latest endpoints. Remember that Sphinx autoflask likes to prefix the names of endpoints with the blueprint's name, for example:

```
.. autoflask:: flexmeasures.app:create()
   :endpoints: flexmeasures_api_v1_1.post_meter_data
```

CHAPTER

# TWENTYSEVEN

# CONTINUOUS INTEGRATION

Here you can learn how to get FlexMeasures onto a server.

**Table of contents**



**Note:** It would be great to enable Dockerization of FlexMeasures, let us know if this matters to you.

## 27.1 WSGI configuration

Here is an example how to serve FlexMeasures as WSGI app:

```
# This file contains the WSGI configuration required to serve up your
# web application.
# It works by setting the variable 'application' to a WSGI handler of some␣
↪description.

import sys
import os
from dotenv import load_dotenv

# add your project directory to the sys.path
project_home = u'/path/to/your/code/flexmeasures'
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

load_dotenv(os.path.join(project_home, '.env'))

# create flask app - need to call it "application" for WSGI to work
from flexmeasures.app import create as create_app
application = create_app()
```

## 27.2 Install the linear solver on the server

To compute schedules, FlexMeasures uses the Cbc mixed integer linear optimization solver. It is used through Pyomo, so in principle supporting a different solver would be possible.

Cbc needs to be present on the server where FlexMeasures runs, under the `cbc` command.

You can install it on Debian like this:

```
apt-get install coinor-cbc
```

If you can't use the package manager on your host, the solver has to be installed from source. We provide an example script to do that, where you can also pass a directory for the installation.

In case you want to install a later version, adapt the version in the script.

## 27.3 Automate deployment via Github actions and Git

At FlexMeasures headquarters, we implemented a specifig workflow to automate our deployment. It uses the Github action workflow (see the `.github/workflows` directory), which pushes to a remote upstream repository. We use this workflow to build and deploy the project to our staging server.

Documenting this might be useful for self-hosters, as well. The GitHub Actions workflows are triggered by commits being pushed to the repository, but it can also inspire your custom deployment script.

We'll refer to Github Actions as our "CI environment" and our staging server as the "deployment server".

- In `lint-and-test.yml`, we set up the app, then run the tests and linters. If testing succeeds and if the commit was on the `main` branch, `deploy.yml` deploys the code from the CI environment to the deployment server.
- Of course, the CI environment needs to properly authenticate at the deployment server.
- With the hooks functionality of Git, a post-receive script can then (re-)start the FlexMeasures app on the deployment server.

Let's review these three steps in detail:

### 27.3.1 Using git to deploy code (remote upstream)

We support deployment of the FlexMeasures project on a staging server via Git checkout.

The deployment uses git's ability to push code to a remote upstream repository. This repository needs to be installed on your staging server.

We trigger this deployment in `deploy.yml` and it's being done in `DEPLOY.sh`. There, we add the remote and then push the current branch to it.

We thus need to tell the deployment evnironment two things:

- Add the setting `STAGING_REMOTE_REPO` as an environment variable on the deployment environment (e.g. `deploy.yml` expects it in the Github repository secrets). An example value is `seita@ssh.our-server.com:/home/seita/flexmeasures-staging/flexmeasures.git`.
- Make sure the env variable `BRANCH_NAME` is set, e.g. to "main", so that the deployment environment knows what exact code to push to your deployment server.

### 27.3.2 Authenticate at the deployment server (with an ssh key)

The CI environment needs to authenticate at the deployment server using an SSH key pair (use `ssh-keygen` to create one, using no password).

To make this work, we need to configure the following:

- Add the deployment server to `~/.ssh/known_hosts` of the deployment environment, so that the deployment environment knows it's okay to talk to the deployment server (e.g. `deploy.yml` expects it in the Github repository secrets as `KNOWN_DEPLOYMENT_HOSTS`). You can create this entry with `ssh-keyscan -t rsa <your host>`.
- Add the private part of the ssh key pair as key in the deployment environment, so that the deployment server can accept the pushed code. (e.g. as `~/.ssh/id_rsa`). In `deploy.yml`, we expect it as the secret `SSH_DEPLOYMENT_KEY`, which addds the key for us.
- Finally, the public part of the key pair should be in `~/.ssh/authorized_keys` on your deployment server.

### 27.3.3 (Re-)start FlexMeasures on the deployment server (install Post-Receive Hook)

Only pushing the code will not actually deploy the updated FlexMeasures into a usable web app on the deployment server. For this, we need to trigger a script.

Log on to the server (via SSH) and install a script to (re-)start FlexMeasures as a Git Post Receive Hook in the remote repo where we deployed the code (see above). This hook will be triggered when a push is received from the deployment environment.

The example script below can be a Post Receive Hook (save as `hooks/post-receive` in your remote origin repo and update paths). It will force checkout the main branch, update dependencies, upgrade the database structure, update the documentation and finally touch the wsgi.py file. This last step is often a way to soft restart the running application, but here you need to adapt to your circumstances.

```
 #!/bin/bash

PATH_TO_GIT_WORK_TREE=/path/to/where/you/want/to/checkout/code/to
ACTIVATE_VENV="command-to-activate-your-venv"
PATH_TO_WSGI=/path/to/wsgi/script/for/the/app

echo "CHECKING OUT CODE TO GIT WORK TREE ($PATH_TO_GIT_WORK_TREE) ..."
GIT_WORK_TREE=$PATH_TO_GIT_WORK_TREE git checkout -f

cd $PATH_TO_GIT_WORK_TREE
PATH=$PATH_TO_VENV/bin:$PATH

echo "INSTALLING DEPENDENCIES ..."
make install-deps

echo "INSTALLING FlexMeasures ..."
make install-flexmeasures

echo "UPGRADING DATABASE STRUCTURE ..."
make upgrade-db

echo "UPDATING DOCUMENTATION ..."
make update-docs
```

(continues on next page)

(continued from previous page)

```
echo "RESTARTING APPLICATION ..."
touch $PATH_TO_WSGI
```

CHAPTER

TWENTYEIGHT

# CODE DOCUMENTATION

Go To source.

# HTTP ROUTING TABLE

## /api