myBeNext API-documentation

Jelle Verstraaten

January 4, 2019

Contents

1 Basic API information

1.1 API Terms of service

The API terms of service are currently available in dutch only. If you use the API you will automatically agree with the terms of service of the API. The latest version of the terms of service can be found here: http://api.benext.eu/documentation/gebruiksovereenkomst.pdf.

1.2 Authentication

Authentication with the API is done through one of four options: HTTP Basic authentication, HTTP Apikey authorization, HTTP HMAC authorization or session authentication. If authentication is required for an API-call and the request doesn't contain any authentication information a 401 `UNAUTHORIZED` is returned with a request for Basic authentication.

1.2.1 Basic

Basic authentication can be used to authenticate as a single user. This form of authentication can be used to control most parts of the system and is convenient for apps. By adding basic authentication to every request you can avoid receiving 401 errors or session timeouts. In order to authenticate a user must pass an `Authorization` HTTP-header. The authorization key is constructed as follows:

1. Username and password are combined into a string "username:password"
2. The resulting string is then encoded using the RFC2045-MIME variant of Base64, except not limited to 76 char/line
3. The authorization method and a space i.e. "Basic" is then put before the encoded string.

An example of this header is: `Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`
An in-depth explanation of Basic authentication can be found here: http://tools.ietf.org/html/rfc2617 and here http://en.wikipedia.org/wiki/Basic_access_authentication.
Authentication can only be done for 1 user at a time. Using Basic authentication a user will only have access to their account. If access to multiple accounts is required an API-key is needed.

1.2.2 API-key

An API-key is required to manage multiple accounts without having access to their username / password. The API-key can either be passed as query parameter: `apikey=<api_key>` or in the `Authorization` header: `Authorization: Apikey <api_key>`.
For the time being API-keys will have to be requested by e-mail.

1.2.3 HMAC

An HMAC-token can be used to provide credentials for a user linked to your API-key without sharing the user credentials or your API-key. This is useful for external services such as IFTTT for which using your API-key might be a security issue.
An HMAC authentication token can be constructed using an API-key. The HMAC can either be passed as query parameter: `hmac=<hmac>&hmac_account_id=<account_id>` or in the `Authorization` header: `Authorization: HMAC <hmac>,account_id=<account_id>`.
The `account_id` should be the account for which you wish to provide the API-key, *NOT* the `account_id` associated with the API-key.

1.2.4 Session authentication

Session authentication is mostly useful for front-end frameworks using javascript. By authenticating once and storing the result cookies the API can be queried without storing the username or password. The following example show how to request a session-token using json:

1. Construct the post body: `{ "username": <username>, "password": <password> }`
2. Send the post to: `/login/api/v1/authenticate/`
3. Store the following cookies: `2myhomesession` and `csrftoken`
4. Add the stored cookies to any following requests

It is possible to construct the post-body using the content-types: `application/json`, `text/plain` or `x-www-form-urlencoded`.

1.3 Accessing and manipulating resources

Accessing and manipulating resources follow a fixed pattern. Depending on the HTTP request method each request is handled differently. These different ways of handling are described as follows:

GET View 1 or more resource(s)
POST Create a new instance of a resource
PUT Update all or some values of the resource
DELETE Remove a resource

Multiple resources can be requested by requesting the base-url of the resources. This returns the entire list of applicable resources. A single resource can be requested by appending the resource id to the url. For example: `/products/` requests all products, while `/products/1/` requests 1 product with the id `1`.
Multiple resources can be requested by using query parameter filters. These are described per resource and vary depending on the resources. In general it's always possible to request multiple resources by id by passing de singular resource name followed by `_id` as query parameter with as value a comma-separated list of ids. For example: `/products/?product_id=1,5,8` will request the resources 1, 5 and 8.

1.3.1 Resource bulk creation

Some resources can be created in bulk. This is more efficient than using separate request because it reduce HTTP and database overhead. If a resource allows bulk creation you can send a list of resources instead of a singular resource as POST-body. Bulk creations are performed as an atomic action. If the response code is not 2XX no resources will have been created.
The response will consist of separate responses for each resource. Possible output responses are:

If the resource is created The response object will contain the status code 201 and the created resource
If an error occurs The response object will contain the relevant status code, message and original request body.

If all individual responses have the same status code, this status code will be used for the entire request status code. E.g. if all resources are created succesfully a 201 CREATED will be returned. If the response status codes are mixed, a 207 MULTISTATUS is returned.
If bulk creation is attempted for a resource that does not support bulk creation the resource will return a 422 UNPROCESSABLE ENTITY.

1.4 Common & custom request headers

1.4.1 Accept-Encoding

It is recommended to send request to the API with an `Accept-Encoding: gzip` header. This will make sure the response is compressed before it is returned to the requester. This reduces network traffic and will (for larger requests) result in a faster transfer time. API calls can also be

performed with compression on the request body. If this is desirable, gzip the request body and add a `Content-Encoding: gzip` header to the request.

1.4.2 Accept-Format

It is possible to request a different format for the response output. The default for resource lists is an object wrapped list. This is to ensure proper JSON handling in most edge cases. To allow for easier handling it is possible to just return the list.
To (explicitly) request dictonary format, pass `Accept-Format: object` as header. This is not required, as object wrapping is the default. To request a plain list with resources, pass `Accept-Format: list` as header.
If an accept-format request is invalid or unknown a 406 NOT ACCEPTABLE may be returned.

1.4.3 Range

It is possible to do pagination with the `Range` header. This API implementation a custom range-specifier called `resourceids`. If a resource accepts a `resourceids` range this will be noted in the request headers. This range specifier can be used to (efficiently) request part of an resourcelist. The semantic checking for this header is limited, so invalid header will result in empty requests or request that are not filter. The format is as follows:

```
Range: resourceids <start>-<end>/<count>
```

Both the `count` and `end` specifier are optional. When omitting the `count` specifier, do not used the `/`-separator. Valid value are:

start Any integer number or `*`
end Any integer number
count Any integer number

A valid range request will filter out all resources smaller than `start` and larger than `end`. This operation is inclusive, we both `start` and `end` will be included in the request. `count` will limit the request to at most that many resource. Any range-limited resource will always return a 206 PARTIAL CONTENT response.

1.5 Cross-site requests

The API has support for cross-site requests using JSONP-style wrapped JSON data. If the optional query parameter `jsonp=<fn_name>` is passed with a specified function name, the JSON data will be returned as data wrapped in a function with the specified name. The `Content-Type` of this data will be `application/javascript`. This can be used to load data from the API from different origins.

1.6 Errors

If a call to the API results in een error, these will be passed in a JSON object. These objects contain a textual description of of the problem and an error code. Where possible an appropriate http error code will be used. An example of an error that can be return is found below:

```
HTTP 404 NOT FOUND
{
  "error": "account not found",
  "resource": "account",
  "code": 12
}
```

1.7 Timestamps and timezone information

The API has multiple resources which accept timestamp as field or as query parameter. Handling these will always be done with respect to timezones. Anywhere a timestamp can be enter an ISO-format timezone (e.g. +0300) can be appended.
To make sure the API always returns consistent data all timestamp are return in the UTC timezone. This makes sure that that timeseries are always consistent no matter what. This also allow for the usage of multiple timezones in requests. For example it is possible to use 2 different timezones in history-entry GETs.

2 Account information

2.1 Accounts

The account resource allows for the looking up and changing of account information. Using an API-key it is also possible to create accounts.

2.1.1 URL patterns

```
/login/api/v1/accounts/
/login/api/v1/accounts/1/
```

2.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `addons`, `email`, `firstname`, `language`, `lastname`, `username`.

account_id Accepts a comma-separated list

account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

2.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.1.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Resource identifier
addons List of all active addons enabled for user
email E-mail for the account, required for password resets
firstname First name for the account
language Preferred language selected by user
lastname Last name for the account
username Username for the account, used to login. Max. 30 characters

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that account doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those accounts will be returned
- If no accounts are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "addons": [],
    "email": "no-reply@benext.eu",
    "firstname": "Demo",
    "language": "nl",
    "lastname": "Account",
    "username": "demo"
}

ResourceList = { "accounts": [
    {
        "account": 1,
        "addons": [],
        "email": "no-reply@benext.eu",
        "firstname": "Demo",
        "language": "nl",
        "lastname": "Account",
        "username": "demo"
    }
] }
```

2.1.2.2 POST

A POST request will create a new account. The account will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.

Required fields `username`, `password`, `email`, `firstname`, `lastname`
Optional fields `language`

Creating a account is done by passing the resource as JSON data with POST. An example of the POST body is

```
  {
    "username": "jdoe",
    "lastname": "Doe",
    "email": "johndoe@example.com",
    "firstname": "John"
    "password": "password123",
  }
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed. Usernames must be unique system-wide. If an account is created with a duplicate username a 409 CONFLICT http code will be returned. A username may be a maximum of 30 characters. If a username is submitted which is longer than 30 characters a 400 Bad Request will be returned.

2.1.2.3 PUT

A PUT request is used to update fields of a account. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `email`, `firstname`, `lastname`, `language`

Updating a account is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
  {
```

```
    "email": "new_email@gmail.com",
    "firstname": "New_firstname",
    "lastname": "New_lastname"
  }
```

A PUT request with filtering query parameters or without an account id *WILL* update multiple accounts. It is *NOT* recommend to send a PUT request without an account id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

2.1.2.4 DELETE

A DELETE request is used to remove an account. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned. After a delete request the account is marked as `inactive` and will not be accessible. After a grace period of 30 days the account will be permanently deleted. It is currently *NOT* possible to restore the account using the API.
A DELETE without an account id is *NOT* allowed and will result in a 400 Bad Request.

2.2 Addresses

Describes the address info for an account, if available. All values except `account` and `address` *MAY* be empty (`""`) or `null`.

2.2.1 URL patterns

```
/login/api/v1/addresses/
/login/api/v1/addresses/229/
/login/api/v1/accounts/682/addresses/
/login/api/v1/accounts/682/addresses/229/
```

2.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `address`, `city`, `country`, `postal_code`, `street_address`.
address_id Accepts a comma-separated list
address_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

2.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.2.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

account Parent account identifier
address Resource identifier
city City
country Country name
postal_code Postal Code
street_address Street address

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that address doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those addresses will be returned
- If no addresses are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 682,
    "address": 229,
    "city": "Amsterdam",
    "country": "Netherlands",
    "postal_code": "1093JX",
```

```
    "street_address": "Ter Gouwstraat 3"
}

ResourceList = { "addresses": [
    {
        "account": 682,
        "address": 229,
        "city": "Amsterdam",
        "country": "Netherlands",
        "postal_code": "1093JX",
        "street_address": "Ter Gouwstraat 3"
    }
] }
```

2.3 Projects

The project resource allows grouping of different accounts into projects.

2.3.1 URL patterns

```
/login/api/v1/projects/
/login/api/v1/projects/6/
```

2.3.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `accounts`, `name`, `organizations`, `project`.

project_id Accepts a comma-separated list

project_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and `_` symbol. The `%`-symbol match any character for any amount. The `_`-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

2.3.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.3.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

accounts Lists accounts linked to this project

name Name of the project

organizations List of organizations linked to this project

project Resource identifier

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that project doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those projects will be returned
- If no projects are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": [
        1,
        5,
        6
    ],
    "name": "Testproject",
    "organizations": [
        1
```

```
    ],
    "project": 6
}

ResourceList = { "projects": [
    {
        "accounts": [
            1,
            5,
            6
        ],
        "name": "Testproject",
        "organizations": [
            1
        ],
        "project": 6
    },
    {
        "accounts": [
            5
        ],
        "name": "DemoNomProject",
        "organizations": [
            2
        ],
        "project": 5
    }
] }
```

2.3.2.2 PUT

A PUT request is used to update fields of a project. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `name`

Updating a project is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
{
  "name": "new name"
}
```

A PUT request with filtering query parameters or without an project id *WILL* update multiple projects. It is *NOT* recommend to send a PUT request without an project id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

2.4 Organizations

The organization resource provides contact information for companies and organizations associated with projects.

2.4.1 URL patterns

`/login/api/v1/organizations/`
`/login/api/v1/organizations/1/`

2.4.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `city`, `email`, `logo`, `name`, `organization`, `phone`, `street`, `website`.
organization_id Accepts a comma-separated list

2.4.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.4.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

city City where organization is located, *MAY* be `null`
email Contact e-mail address for general inquiry, *MAY* be `null`
logo Logo associated with organization, *MAY* be `null`
name Printable name of the organization, *MAY* contain UTF-8 and/or special characters
organization Resource identifier
phone Customer service phone number, *MAY* be `null`
street Street address where organization is located, *MAY* be `null`
website Website of the organization, *MAY* be `null`

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that organization doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those organizations will be returned
- If no organizations are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "city": "Amsterdam",
    "email": "info@benext.eu",
    "logo": null,
    "name": "BeNext B.V.",
    "organization": 1,
    "phone": null,
    "street": "Ter Gouwstraat 3",
    "website": "https://www.benext.eu"
```

```
}

ResourceList = { "organizations": [
    {
        "city": "Amsterdam",
        "email": "info@benext.eu",
        "logo": null,
        "name": "BeNext B.V.",
        "organization": 1,
        "phone": null,
        "street": "Ter Gouwstraat 3",
        "website": "https://www.benext.eu"
    }
] }
```

2.5 Lifestyles

The lifestyle resource lists the available lifestyles for an account and denotes which is the currently active lifestyle.
The id field will always be a number between 1 and 10 inclusive indicating the internal id of the lifestyle. These map 1-to-1 on lifestyle names, although it is possible that lifestyle names change. It is possible (and likely) that only some of the possible lifestyles are actually available.
Possible lifestyles include:

Home 1
Home 2 2
Home 3 3
Home 4 4
Away 5
Away 2 6
Away 3 7
Away 4 8
Sleep 9
Party 10

2.5.1 URL patterns

```
/login/api/v1/lifestyles/
/login/api/v1/lifestyles/2/
/login/api/v1/accounts/1/lifestyles/
/login/api/v1/accounts/1/lifestyles/2/
```

2.5.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `active`, `id`, `lifestyle`, `name`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
lifestyle_id Accepts a comma-separated list
active `true` or `false`
id number between 1 and 10, inclusive

2.5.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.5.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier
active Marks the currently active lifestyle
id Account-level unique id for lifestyle

lifestyle Resource identifier
name Name of lifestyle

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that lifestyle doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those lifestyles will be returned
- If no lifestyles are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "active": false,
    "id": 5,
    "lifestyle": 2,
    "name": "away"
}

ResourceList = { "lifestyles": [
    {
        "account": 1,
        "active": true,
        "id": 1,
        "lifestyle": 1,
        "name": "home"
    },
    {
        "account": 1,
        "active": false,
        "id": 5,
        "lifestyle": 2,
        "name": "away"
    },
    {
        "account": 1,
        "active": false,
        "id": 9,
        "lifestyle": 3,
        "name": "sleep"
    }
] }
```

2.5.2.2 PUT

A PUT request is used to update fields of a lifestyle. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `active`

Updating a lifestyle is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
{
  "active": true
}
```

- The *ONLY* valid value for `active` is `true`.
- It is *NOT* possible to deactivate a lifestyle.
- If the value `false` is passed a 400 Bad Request will be returned.
- It is *NOT* possible to activate multiple lifestyles at the same time. A PUT request on multiple resources will result in a 400 Bad Request.

A PUT request with filtering query parameters or without an lifestyle id *WILL* update multiple lifestyles. It is *NOT* recommend to send a PUT request without an lifestyle id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

2.6 Scenes

The scene resource describes the required data to send local-api requests to trigger scene and lists the available scenes by name.
It is currently not possible to request what products are changed by a scene but this is on the feature list for addition to the scene resource.

2.6.1 URL patterns

```
/login/api/v1/scenes/
/login/api/v1/scenes/3/
/login/api/v1/accounts/1/scenes/
/login/api/v1/accounts/1/scenes/3/
```

2.6.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `code`, `name`, `scene`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
scene_id Accepts a comma-separated list
scene_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

2.6.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.6.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier
code Code to trigger scene on the local API of the Gateway
name Name of scene
scene Resource identifier

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that scene doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those scenes will be returned
- If no scenes are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "code": 75,
    "name": "LightsOn",
    "scene": 3
}

ResourceList = { "scenes": [
    {
        "account": 1,
        "code": 75,
        "name": "LightsOn",
        "scene": 3
    },
    {
        "account": 1,
        "code": 76,
        "name": "Film",
        "scene": 4
    }
] }
```

2.6.2.2 PUT

A PUT request is used to update fields of a scene. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `name`, `trigger`

Updating a scene is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
  {
    "name": "TestScene"
  }
```

A PUT request with filtering query parameters or without an scene id *WILL* update multiple scenes. It is *NOT* recommend to send a PUT request without an scene id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

2.6.2.3 DELETE

A DELETE request is used to remove a scene. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an scene id *WILL* delete multiple scenes. It is *NOT* recommend send a DELETE request without an scene id or filtering query parameters.

2.7 Files

Describes the contents and location of various files linked to an account. A title and description provide some context to about the file. Additional tags allow for filtering and classification. Examples of tags may be: `manual`, `promo`, `tandc`.

2.7.1 URL patterns

```
/login/api/v1/files/
/login/api/v1/accounts/1/files/
```

2.7.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `desc`, `location`, `mimetype`, `tags`, `title`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
tags_contains Accepts a comma-separated list

2.7.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

2.7.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

account Parent account identifier
desc Description of the file,
location URL pointing to the file
mimetype Mime-type of the file
tags Tags associated with the image
title Title of file, intended for printing and display

If filter queryparams are passed:

- A `ResourceList` containing those files will be returned
- If no files are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "files": [
    [
        {
            "account": 1,
            "description": "Manual for explaining our NOM-service",
            "location": "https://s3-eu-west-1.amazonaws.com/cdn-benext/static/nom_files/manuals/Be
            "tags": [
                "manual",
                "nom"
```

```
            ],
            "title": "NOM/EPV handleiding",
            "type": "application/pdf"
        },
        {
            "account": 1,
            "description": "The BeNext Smart Home website",
            "location": "https://www.benext.eu/",
            "tags": [
                "promo"
            ],
            "title": "Homepage",
            "type": "text/html"
        }
    ]
] }
```

3 General product information

3.1 Producttypes

The producttype resource describes the type of a product and the associated image. They also contain information on the full name of the product and metadata about how the product should be used in the myBeNext interface.
If possible the correct producttype should be chosen over installing a product as a generic device.

3.1.1 URL patterns

```
/login/api/v1/producttypes/
/login/api/v1/producttypes/42/
```

3.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `addable`, `appearance`, `image`, `installable`, `manufacturer`, `name`, `producttype`, `type`.

addable Can be `true` or `false`. List only products which can also be added using the myBeNext user interface. If a product is NOT in this list it CAN still be added, but this action wouldn't normally be supported.

producttype_id Accepts a comma-separated list

3.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.1.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

addable Indicates if this product can be added under normal circumstances

appearance Lists the possibile appearance options for the "producttype, always includes the producttype itself

image Image of producttype

installable Indicates if this product can be installed using the API

manufacturer Gives the name of manufacturer, if available/applicable. May be `null`

name Name of producttype

producttype Resource identifier

type Specificies the type of product, `P` for physical product (the devices), `I` for images (e.g. Boiler, PC, lamp)

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that producttype doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those producttypes will be returned
- If no producttypes are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "appearance": [
        42, 48, 143, 140, 82, 83, 53, 56,
        55, 144, 142, 141, 47, 57, 61, 115,
        58, 54, 51, 52, 60, 111, 112
    ],
    "image": "/static/uploads/peripheral_class_images/EnergySwitch.png",
    "manufacturer": "BeNext",
    "name": "Energy Switch",
    "producttype": 42,
    "type": "P"
}

ResourceList = { "producttypes": [
    {
        "appearance": [
            42, 48, 143, 140, 82, 83, 53, 56,
            55, 144, 142, 141, 47, 57, 61, 115,
            58, 54, 51, 52, 60, 111, 112
        ],
        "image": "/static/uploads/peripheral_class_images/EnergySwitch.png",
        "manufacturer": "BeNext",
        "name": "Energy Switch",
        "producttype": 42,
        "type": "P"
    }
] }
```

3.2 Products

The product resource lists the virtual products linked to an account. Each virtual product may be linked to *one* physical product. These links are based on Manufacturer specific information listed by the device.
Products also have subresources: properties. These can be used to determine functionality of a device. The appearance of a product is linked to a producttype which contains the image for the icon.

3.2.1 URL patterns

```
/login/api/v1/products/
/login/api/v1/products/29/
/login/api/v1/accounts/1/products/
/login/api/v1/accounts/1/products/29/
```

3.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `appearance`, `name`, `product`, `producttype`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
appearance_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_name Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
producttype_id Accepts a comma-separated list

3.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.2.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier
appearance Producttype resource id, describes the associated image
name Name of product, may be an empty string
product Resource identifier
producttype Producttype resource id, describes the physical product

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.

- A 404 NOT FOUND will be returned if that product doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those products will be returned
- If no products are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "appearance": 1,
    "name": "Internet Gateway",
    "product": 35,
    "producttype": 1
}

ResourceList = { "products": [
    {
        "account": 1,
        "appearance": 73,
        "name": "Electricity Meter",
        "product": 36,
        "producttype": 73
    },
    {
        "account": 1,
        "appearance": 48,
        "name": "Boiler",
        "product": 41,
        "producttype": 42
    }
] }
```

3.2.2.2 POST

A POST request will create a new product. The product will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.

Required fields `producttype`, `account`
Optional fields `name`, `appearance`

Creating a product is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "producttype": 1,
  "account": 1
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

3.2.2.3 PUT

A PUT request is used to update fields of a product. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `name`, `appearance`

Updating a product is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
{
  "name": "Boiler",
  "appearance": 53
}
```

A PUT request with filtering query parameters or without an product id *WILL* update multiple products. It is *NOT* recommend to send a PUT request without an product id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

3.2.2.4 DELETE

A DELETE request is used to remove a product. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned. After a delete request the product is marked as `delete` and will not be accessible. After a grace period of 7 days the product will be permanently deleted. It is currently *NOT* possible to restore the product using the API.
A DELETE with filtering query parameters or without an product id *WILL* delete multiple products. It is *NOT* recommend send a DELETE request without an product id or filtering query parameters.

3.3 Datatypes

The datatype resource describes how values should be displayed and handled. The functionality of a property *CAN* and *SHOULD* be determined by looking at the datatype of the property.
If a property value has a specific suffix, this should be used to correctly display the value. The property resource is described by the following value types:

Type	Meaning
`boolean`	A boolean value will have 2 choices, described by `min` and `max`. The respective `min_conv` and `max_conv` values should be used to display which one is active. As a rule, the `min` value will be an inactive state.
`choice`	The value can be 1 of the keys listed in the `choices` field. Values should be displayed using the values from the `choices` field.
`integer`	An integer value with a `min`, `max` and `step` argument for usage in slider and spinboxes. Does *NOT* allow decimal values.
`float`	An floating point value with a `min`, `max` and `step` argument for usage in slider and spinboxes. Does allow decimal values.
`composite`	These values *MAY* be composed of 2 separate integer values, divided by a comma (e.g. `3,FF`). The first part will be an integer, the second part a hexadecimal encoded integer.
`custom`	Parsing and handling this type of property requires custom code. For implementation and support contact the Api-maintainer.

3.3.1 URL patterns

```
/login/api/v1/datatypes/
/login/api/v1/datatypes/12/
```

3.3.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `datatype`, `name`, `suffix`, `value`.
datatype_id Accepts a comma-separated list

3.3.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.3.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

datatype Resource identifier
name Name of datatype
suffix Describes the suffix for the value, if applicable
value value description

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.

- A 404 NOT FOUND will be returned if that datatype doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those datatypes will be returned
- If no datatypes are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "datatype": 15,
    "name": "Contact",
    "suffix": "",
    "value": {
        "cmp": [
            "eq"
        ],
        "max": 255,
        "max_conv": "Opened",
        "min": 0,
        "min_conv": "Closed",
        "type": "boolean"
    }
}

ResourceList = { "datatypes": [
    {
        "datatype": 25,
        "name": "Mode",
        "suffix": "",
        "value": {
            "choice": {
                "1": "Alarm",
                "2": "Error",
                "3": "Walk in",
                "4": "Alert",
                "5": "Wake up",
                "6": "Doorbell"
            },
            "cmp": [
                "eq"
            ],
            "type": "choice"
        }
    },
    {
        "datatype": 27,
        "name": "Duration",
        "suffix": " sec",
        "value": {
            "cmp": [
                "lt",
                "eq",
                "gt"
            ],
            "max": 15,
            "min": 0,
            "step": 1,
            "suffix": " sec",
            "type": "integer"
```

```
        }
    }
] }
```

3.4 Properties

The property resource describes a single measurement source for a product. Any values received for the same property always refer to the same sensor on the same product. Examples include temperature and energy measurements. It's possible for two properties from one product to have the same datatype. This simply means the correspond to two different sensors. e.g. inside temperature and outside temperature

3.4.1 URL patterns

```
/login/api/v1/properties/
/login/api/v1/properties/222/
/login/api/v1/products/36/properties/
/login/api/v1/products/36/properties/222/
/login/api/v1/account/1/properties/
/login/api/v1/account/1/properties/222/
/login/api/v1/accounts/1/products/44/properties/
/login/api/v1/accounts/1/products/44/properties/222/
```

3.4.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `datatype`, `name`, `product`, `property`, `receiving`, `sending`, `updated`, `value`.

account_id Accepts a comma-separated list

account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

appearance_id Accepts a comma-separated list

datatype_id Accepts a comma-separated list

product_id Accepts a comma-separated list

product_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

producttype_id Accepts a comma-separated list

property_id Accepts a comma-separated list

property_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

updated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

updated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

value_like Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

3.4.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.4.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

datatype Datatype resource id, describes the value conversion
name Name of property, may *NOT* be empty
product Parent product identifier
property Resource identifier
receiving Indicates if a product can receive messages
sending Indicates if a product can send messages
updated Timestamp at which the latest value was received in isoformat, may also be `null` if no value is ever received
value Current value

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that property doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those properties will be returned
- If no properties are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "datatype": 8,
    "name": "Dimmer",
    "product": 44,
    "property": 222,
    "receiving": true,
    "sending": true,
    "updated": "2018-04-20T14:53:00Z",
    "value": "80"
}

ResourceList = { "properties": [
    {
        "datatype": 8,
        "name": "Dimmer",
        "product": 44,
        "property": 222,
        "receiving": true,
        "sending": true,
        "updated": "2018-04-20T14:53:00Z",
        "value": "80"
    },
    {
        "datatype": 18,
        "name": "Energy",
        "product": 44,
        "property": 223,
        "receiving": false,
        "sending": true,
        "updated": "2018-04-20T14:53:00Z",
        "value": "150"
    },
```

```
    {
        "datatype": 39,
        "name": "kWh",
        "product": 44,
        "property": 224,
        "receiving": false,
        "sending": true,
        "updated": "2018-04-20T14:53:00Z",
        "value": "348.92"
    }
] }
```

3.4.2.2 POST

A POST request will create a new property. The property will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error. This resource allows the bulk creation. To create multiple resources in 1 request, send all resources in a list. Read the introduction chapter for more information on this feature.

Required fields `product`, `name`, `datatype`, `receiving`, `sending`
Optional fields –

Creating a property is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "product": ,
  "name": ,
  "datatype": ,
  "receiving": ,
  "sending":
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

3.4.2.3 PUT

A PUT request is used to update fields of a property. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `name`, `datatype`, `receiving`, `sending`, `value`

Updating a property is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
{
  "name": ,
  "datatype": ,
  "receiving": ,
  "sending": ,
  "value":
}
```

A PUT request with filtering query parameters or without an property id *WILL* update multiple properties. It is *NOT* recommend to send a PUT request without an property id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

3.4.2.4 DELETE

A DELETE request is used to remove a property. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an property id *WILL* delete multiple properties. It is *NOT* recommend send a DELETE request without an property id or filtering query parameters.

3.5 Mainmeters

The mainmeter resource describes which properties combine to form the mainmeter for an account. Note that these property *MAY* be spread amongst multiple physical products.

3.5.1 URL patterns

```
/api/v1/mainmeters/
/api/v1/mainmeters/331/
/api/v1/accounts/682/mainmeters/
/api/v1/accounts/682/mainmeters/331/
```

3.5.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `mainmeter`, `property`, `tariff`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
mainmeter_id Accepts a comma-separated list

3.5.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.5.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

account Parent account identifier
mainmeter Resource identifier
property Parent property identifier
tariff Tariff identifier

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that mainmeter doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those mainmeters will be returned
- If no mainmeters are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 682,
    "mainmeter": 1,
    "property": 3763,
```

```
    "tariff": 331
}

ResourceList = { "mainmeters": [
    {
        "account": 682,
        "mainmeter": 1,
        "property": 3763,
        "tariff": 331
    }
] }
```

3.5.2.2 POST

A POST request will create a new mainmeter. The mainmeter will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.

Required fields `account`, `property`, `tariff`
Optional fields –

Creating a mainmeter is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "account": 682,
  "property": 3763,
  "tariff": 331
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

3.5.2.3 DELETE

A DELETE request is used to remove a mainmeter. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an mainmeter id *WILL* delete multiple mainmeters. It is *NOT* recommend send a DELETE request without an mainmeter id or filtering query parameters.

3.6 Tariffs

The tariff resource provides information about the configured tariffs for the mainmeter. These can be used to convert measured values into monetary values uniformly throughout the system. Note that the symbols are purely graphic and provide no form of conversion.

3.6.1 URL patterns

```
/login/api/v1/tariffs/
/login/api/v1/tariffs/331/
/login/api/v1/accounts/682/tariffs/
/login/api/v1/accounts/682/tariffs/331/
```

3.6.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `symbol`, `tariff`, `type`, `value`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
tariff_id Accepts a comma-separated list
type_str Accepts a comma-separated list

3.6.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

3.6.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

account Parent account identifier
symbol Symbol for tariff
tariff Resource identifier
type Tariff type, possible options:
value Tariff per unit

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that tariff doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those tariffs will be returned
- If no tariffs are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 682,
    "symbol": "\\u20ac",
    "tariff": 331,
    "type": "energy_normal",
    "value": 0.12
}

ResourceList = { "tariffs": [
    {
        "account": 682,
        "symbol": "\\u20ac",
        "tariff": 331,
        "type": "energy_normal",
        "value": 0.22
    },
    {
        "account": 682,
        "symbol": "\\u20ac",
        "tariff": 335,
        "type": "gas",
        "value": 0.65
    },
    {
        "account": 682,
        "symbol": "\\u20ac",
        "tariff": 336,
        "type": "water",
        "value": 0.006
    }
] }
```

3.6.2.2 PUT

A PUT request is used to update fields of a tariff. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `value`, `symbol`

Updating a tariff is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
  {
    "value": 0.22,
    "symbol": "$"
  }
```

A PUT request with filtering query parameters or without an tariff id *WILL* update multiple tariffs. It is *NOT* recommend to send a PUT request without an tariff id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

4 Sensor data & availability

4.1 Availabilities

Provide information about when products were available and unavailable.

4.1.1 URL patterns

`/login/api/v1/availability/`

4.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `product`, `status`, `timestamp`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
begin ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
end ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
product_id Accepts a comma-separated list

4.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

4.1.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

product Product for which the status message is received
status Availabilty status report: available, retrying and unreachable
timestamp Time at which the status code was received, in isoformat

If filter queryparams are passed:

- A `ResourceList` containing those availabilities will be returned
- If no availabilities are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "availabilities": [
    {
        "product": 35670,
        "status": "product_retry",
        "timestamp": "2017-08-16T12:34:40.159Z"
    },
```

```
    {
        "product": 35670,
        "status": "product_avail",
        "timestamp": "2017-08-16T12:34:50.039Z"
    }
] }
```

4.2 Historyentries

History entries are the way raw data is stored in the myBeNext environment. They are linked to a property and contain a timestamp and a "raw" value. This value is stored as a string because it may contain any form of data including, but not limited to: comma-separated values, floating points, strings and Z-wave specific metadata.

4.2.1 URL patterns

```
/login/api/v1/historyentries/<datetime>/<datetime>/
/login/api/v1/properties/historyentries/<datetime>/<datetime>/
/login/api/v1/properties/222/historyentries/<datetime>/<datetime>/
/login/api/v1/products/historyentries/<datetime>/<datetime>/
/login/api/v1/products/29/historyentries/<datetime>/<datetime>/
```

The `<datetime>` part of the URL consists of an ISO8601 extended timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
The first timestamp describes the start of the query (inclusive), the second timestamp describes the end of query (exclusive).

4.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `property`, `timestamp`, `value`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
appearance_id Accepts a comma-separated list
datatype_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
producttype_id Accepts a comma-separated list
property_id Accepts a comma-separated list
property_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
updated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
updated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
value_like Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

4.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

4.2.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

property Parent property identifier
timestamp Timestamp for entry in isoformat
value Historic value at timestamp

If the `end` timestamp is smaller than or equal to the `begin` timestamp a 400 Bad Request error is returned. If more than 30 days of data is requested a 400 Bad Request is returned.
If filter queryparams are passed:

- A `ResourceList` containing those historyentries will be returned
- If no historyentries are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "historyentries": [
    {
        "property": 172,
        "timestamp": "2015-03-30T00:00:00Z",
        "value": 0
    }
] }
```

4.3 Energyentries

Energy entries are the way aggregated data is stored in the myBeNext environment. They are linked to a property and contain a timestamp and a floating point value. All values are normalized to 15 minute interval values. These are *NOT* cummulative and can be summed to gain a total over a period of time (e.g. sum all data from 2015-05-01 to 2015-05-05 to gain the total energy used over this period).
These values can be aggregated a different resolution as listed below.

4.3.1 URL patterns

```
/login/api/v1/energyentries/<aggregate>/<datetime>/<datetime>/
/login/api/v1/properties/energyentries/<aggregate>/<datetime>/<datetime>/
/login/api/v1/properties/222/energyentries/<aggregate>/<datetime>/<datetime>/
/login/api/v1/products/energyentries/<aggregate>/<datetime>/<datetime>/
/login/api/v1/products/29/energyentries/<aggregate>/<datetime>/<datetime>/
```

The `<aggregate>` part of the URL indicates the resolution at which to aggregate. Possible options are: `minute`, `hour`, `day`, `week`, `month`, `quarter`, `year`
The `<datetime>` part of the URL consists of an ISO8601 extended timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
The first timestamp describes the start of the query (inclusive), the second timestamp describes the end of query (exclusive).

4.3.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `property`, `timestamp`, `value`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
appearance_id Accepts a comma-separated list
datatype_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
producttype_id Accepts a comma-separated list
property_id Accepts a comma-separated list
property_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
updated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
updated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
value_like Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

4.3.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

4.3.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

property Parent property identifier
timestamp Timestamp for entry in isoformat
value Aggregated value

If the `end` timestamp is smaller than or equal to the `begin` timestamp a 400 Bad Request error is returned.
If filter queryparams are passed:

- A `ResourceList` containing those energyentries will be returned
- If no energyentries are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "energyentries": [
    {
        "property": 248,
        "timestamp": "2015-03-30T00:00:00Z",
        "value": 99.24999999999993
    },
    {
        "property": 249,
        "timestamp": "2015-03-30T00:00:00Z",
        "value": 68.18848888888905
    },
    {
        "property": 250,
        "timestamp": "2015-03-30T00:00:00Z",
        "value": 0.0030000000000001137
    }
] }
```

4.4 Energyentrytotals

Energy entries are the way aggregated data is stored in the myBeNext environment. They are linked to a property and contain a timestamp and a floating point value. All values are normalized to 15 minute interval values. These are *NOT* cummulative and can be summed to gain a total over a period of time (e.g. sum all data from 2015-05-01 to 2015-05-05 to gain the total energy used over this period).
These values can be aggregated a different resolution as listed below.

4.4.1 URL patterns

```
/login/api/v1/energyentries/total/<datetime>/<datetime>/
/login/api/v1/properties/energyentries/total/<datetime>/<datetime>/
/login/api/v1/properties/222/energyentries/total/<datetime>/<datetime>/
/login/api/v1/products/energyentries/total/<datetime>/<datetime>/
/login/api/v1/products/29/energyentries/total/<datetime>/<datetime>/
```

Note that this url is similar to the normal energyentry resource. The main difference between the two is the output format, which for energyentrytotals does *NOT* include a timestamp.
The `<datetime>` part of the URL consists of an ISO8601 extended timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
The first timestamp describes the start of the query (inclusive), the second timestamp describes the end of query (exclusive).

4.4.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `property`, `value`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
appearance_id Accepts a comma-separated list
datatype_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
producttype_id Accepts a comma-separated list
property_id Accepts a comma-separated list
property_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
updated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
updated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
value_like Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

4.4.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

4.4.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

property Parent property identifier
value Aggregated value

If filter queryparams are passed:

- A `ResourceList` containing those energyentrytotals will be returned
- If no energyentrytotals are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "energyentrytotals": [
    {
        "property": 1103,
        "value": 0.316
    },
    {
        "property": 914,
        "value": 11.6200000000092
    },
    {
        "property": 1102,
        "value": 0.168000000000001
    }
] }
```

5 Device linking & configuration

5.1 Gateways

The Gateway resource describes the metadata associated with the physical Gateway connected to the account. Gateways can be installed using the serialnumber, a public ip adress of the network the Gateway is on or a mac address that is listed on the back of the Gateway.

5.1.1 URL patterns

```
/login/api/v1/gateways/
/login/api/v1/gateways/1/
```

5.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `firmware`, `gateway`, `public_ip`, `serial`, `status`, `status_time`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
gateway_id Accepts a comma-separated list
public_ip List gateway which have this ip is external ip address, may return multiple Gateways
serial filter on a serial number of a gateway

5.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.1.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier
firmware Describes whether the firmware can or should be updated
gateway Resource identifier
public_ip Public ip associated with the Gateway
serial Internal serial number of Gateway
status Gateway availability status
status_time Status time

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that gateway doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those gateways will be returned
- If no gateways are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "firmware": "newest",
    "gateway": 1,
    "public_ip": "4.4.4.4",
    "serial": "000042"
}

ResourceList = { "gateways": [
    {
        "account": 1,
        "firmware": "newest",
        "gateway": 1,
        "public_ip": "4.4.4.4",
        "serial": "000042"
    }
] }
```

5.1.2.2 POST

A POST request will create a new gateway. The gateway will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.
When creating a gateway either a `serial` or `public_ip` is *REQUIRED*. If either one is *NOT* present a 400 BAD REQUEST will be returned with a message explaining that 1 of them is required. In the case that both are present the `serial` *WILL* take precedence over `public_ip`.
If the specified `account` doesn't exist a 404 NOT FOUND will be raised, indicating the account doesn't exist. If the specified `account` already has a gateway installed, a 409 CONFLICT will be returned.
The `exists` value specifies if the server should check whether the gateway specified already exists. This is useful if you want to want to verify if the gateway is connected to the server. If the `exists` value is set to `true` the API will verify that *1* gateway is present for the provided `serial` or `public_ip`. If no Gateway is connected with the server a 404 NOT FOUND http code will be returned. If multiple gateways are found a 400 BAD REQUEST is returned with a message explained multiple gateways were found.

Required fields `account`, (`serial` or `public_ip`)
Optional fields `serial`, `public_ip`, `exists`

Creating a gateway is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "account": 1,
  "serial": "000042"
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed. If the provided `serial` is already in use, the api will return a 409 CONFLICT indicating the serial is already in use.

5.1.2.3 DELETE

A DELETE request is used to remove a gateway. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an gateway id *WILL* delete multiple gateways. It is *NOT* recommend send a DELETE request without an gateway id or filtering query parameters.

5.2 Settings

The settings subresource is lists the possible settings you can send to a device. These are the same settings that can be sent using the myBeNext-interface. The setting resource does *NOT* have a unique identifier. Rather, the resource is a combination of the product resource and the name of the setting.
The resource has a special `value` field which describes the possible values you can send to the resource.
The contents of the `value`-field depends on the `type` that is passed in the value. The following types are currently supported:

Type	Meaning
`choice`	1 of the choices listed in the `choices` field may be passed in the `value` field.
`multiplechoice`	1 or more of the choices listed in the `choices` field may be passed in the `value` field. These values should be passed as comma-separated list
`char`	A character string may be passed in the `value` field. Optional `min` and/or `max` fields may describe the minimum and/or maximum length that may be passed.
`integer`	An integer may be passed in the `value` field. Optional `min`, `max` and `step` field may describe the minimum and maximum values. The `step` field describes the possible step from the minimum value up. The maximum value will always be a valid value.
`float`	A float may be passed in the `value` field. Optional `min`, `max` and `step` field may describe the minimum and maximum values. The `step` field describes the possible step from the minimum value up. The maximum value will always be a valid value.
`boolean`	A boolean `true` or `false` may be passed in the `value` field.
`null`	A special type reserved for future support. Ignore any fields with this type.

5.2.1 URL patterns

```
/login/api/v1/settings/
/login/api/v1/products/108/settings/
```

5.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `name`, `product`, `value`.
product_id Accepts a comma-separated list

5.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.2.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

name Name of setting
product Parent product identifier
value value description

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that setting doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those settings will be returned
- If no settings are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = null

ResourceList = { "settings": [
    {
        "name": "latitude",
        "product": 108,
        "value": {
            "max": 90,
            "min": -90,
            "type": "float"
        }
    },
    {
        "name": "longtitude",
        "product": 108,
        "value": {
            "max": 180,
            "min": -180,
            "type": "float"
        }
    },
    {
        "name": "timezone",
        "product": 108,
        "value": {
            "choices": [
                -12, -11, -10, -9,
                -8, -7, -6, -5,
                -4, -3, -2, -1,
                0, 1, 2, 3,
                4, 5, 6, 7,
                8, 9, 10,11, 12
            ],
            "type": "choice"
        }
    },
    {
        "name": "dst",
        "product": 108,
        "value": {
            "type": "boolean"
        }
    }
] }
```

5.2.2.2 POST

A POST request will send the new setting to the specified product. Any query parameters not required for creation will be ignored. All arguments are required. If the passed `value` does not

corrispond to the listed `value` type a 400 Bad Request will be raised. If the passed `name` is not a valid setting a 400 Bad Request will be passed.

Required fields `product`, `name`, `value`
Optional fields –

Creating a setting is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "product": 108,
  "name": "latitude",
  "value": 52.37
}
```

If the settings is sent succesfully a 204 No Content http code will be returned.

5.3 Nodes

The node resource lists the nodes that the Gateway has registered.
If a node has a linked product the product field contains the resource identifier for that product. Otherwise the product field *MAY* be `null`.
If a node has relevant version and/or serial number information these fields will contain the relevant info. Otherwise they *MAY* be `null`.

5.3.1 URL patterns

```
/login/api/v1/nodes/
/login/api/v1/nodes/13/
/login/api/v1/gateways/2/nodes/
/login/api/v1/gateways/2/nodes/13/
```

5.3.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `gateway`, `node`, `product`, `protocol`, `serial`, `status`, `version`, `zwave_id`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
gateway_id Accepts a comma-separated list
node_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_null Accepts a boolean value: `true` or `false`
protocol_id Accepts a comma-separated list
status_id Accepts a comma-separated list

5.3.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.3.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

gateway Parent gateway identifier
node Resource identifier
product Linked product identifier
protocol Node protocol
serial Serial of the node, *MAY* be `null`
status Z-wave availability status
version Version of the node, *MAY* be `null`
zwave_id Z-wave node id

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that node doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those nodes will be returned
- If no nodes are relevant or selected, an empty list will be returned.

The status of the node is indicated by the `status` field. This field has multiple possible values:

0: Node is available
1: Node has an unstable connection
2: Node has no connection

The protocol of the node is indicated by the `protocol` field. This field has multiple possible values:
`mygate`: : This the the representative node for the Gateway

`zwave`: These nodes communicate using the Z-wave protocol
`p1`: These nodes communicate using the P1 or W-MBus protocol
`webcam`: These nodes are webcams

Output JSON

```
Resource = {
    "gateway": 2,
    "node": 13,
    "product": 13,
    "protocol": "zwave",
    "serial": null,
    "status": 2,
    "version": null,
    "zwave_id": 3
}

ResourceList = { "nodes": [
    {
        "gateway": 2,
        "node": 13,
        "product": 13,
        "protocol": "zwave",
        "serial": null,
        "status": 2,
        "version": null,
        "zwave_id": 3
    },
    {
        "gateway": 2,
        "node": 14,
        "product": 17,
        "protocol": "zwave",
        "serial": null,
        "status": 2,
        "version": null,
        "zwave_id": 8
    }
] }
```

5.3.2.2 POST

A POST request will request a nodeinfo from the Gateway for the relevant `zwave_id`. If a node with the specified `zwave_id` exists it will be returned be the Gateway and created with all relevant information. This can be used to discover what nodes are available in a Z-wave network.

Required fields `zwave_id`, `gateway`
Optional fields –

Requesting a nodeinfo is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "zwave_id": 26,
  "gateway": 2
}
```

If the resource is requested succesfully a 202 ACCEPTED http code will be returned.

5.4 Propertymappings

The propertymapping resource describes the way properties are linked to the physical parameters of a node. This is based on the Z-wave command class model. For more information on this subject, please contact benext at support@benext.eu

5.4.1 URL patterns

```
/login/api/v1/propertymappings/
/login/api/v1/nodes/13/propertymappings/
/login/api/v1/gateways/2/propertymappings/
```

5.4.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `channel`, `command_class`, `node`, `parameter`, `property`, `propertymapping`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the % and _ symbol. The %-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
channel_id Accepts a comma-separated list
command_class_id Accepts a comma-separated list
gateway_id Accepts a comma-separated list
node_id Accepts a comma-separated list
parameter_id Accepts a comma-separated list
product_id Accepts a comma-separated list
product_null Accepts a boolean value: `true` or `false`
property_id Accepts a comma-separated list
propertymapping_id Accepts a comma-separated list
protocol_id Accepts a comma-separated list
status_id Accepts a comma-separated list

5.4.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.4.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

channel Z-wave command class channel
command_class Z-wave command class ID
node Parent gateway identifier
parameter Z-wave scale/option identifier
property Linked property identifier
propertymapping Resource identifier

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that propertymapping doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those propertymappings will be returned
- If no propertymappings are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "channel": 1,
    "command_class": 50,
    "node": 211,
    "parameter": 0,
    "property": 1100,
    "propertymapping": 681
}

ResourceList = { "propertymappings": [
    {
        "channel": 1,
        "command_class": 50,
        "node": 211,
        "parameter": 0,
        "property": 1100,
        "propertymapping": 681
    },
    {
        "channel": 2,
        "command_class": 50,
        "node": 211,
        "parameter": 0,
        "property": 1101,
        "propertymapping": 682
    },
    {
        "channel": 3,
        "command_class": 50,
        "node": 211,
        "parameter": 0,
        "property": 1102,
        "propertymapping": 683
    }
] }
```

5.4.2.2 POST

A POST request will create a new propertymapping. The propertymapping will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.

Required fields `node`, `command_class`, `property`
Optional fields `channel`, `parameter`

Creating a propertymapping is done by passing the resource as JSON data with POST. An example of the POST body is

```
  {
    "node": 13,
```

```
    "command_class": 50,
    "property": 298
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

5.4.2.3 DELETE

A DELETE request is used to remove a propertymapping. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an propertymapping id *WILL* delete multiple propertymappings. It is *NOT* recommend send a DELETE request without an propertymapping id or filtering query parameters.

5.5 Installation

The installation resource is used to install or "unpair" physical Z-wave products onto virtual products.This resource is a wrapper for the installation procedure and doesn't directly map to any database resources.
The installation API can be used to start and stop the installation process and it will report back feedback on the current state of the installation process. Possible `type` statuses are:

Type	Meaning
`install_start`	Install request succesfully sent to Gateway
`install_accepted`	Install request received by Gateway
`install_searching`	Gateway ready for nodeinfo
`install_found`	Gateway received nodeinfo
`install_configuring`	Gateway is configuring node
`install_conf_success`	Node configuration success
`install_success`	Node is succesfully installed on product
`install_unknown_dev`	Installed node is an unknown product
`install_wrong_dev`	Installed node did not match type of product
`install_node_mapped`	Node was already mapped to a product
`install_aborted`	Installation was aborted
`install_rwu_fail`	Node went in sleep-mode to fast
`install_conf_fail_w`	Node went unreachable during configuration (Wakeup)
`install_conf_fail_l`	Node went unreachable during configuration (Listening)
`install_internal`	Internal Gateway error
`install_sec_hs_fail`	Secure handshake failure
`install_no_sis`	No SIS available
`install_busy`	Driver timeout
`install_no_space`	No more space in Gateway
`install_proto_fail`	Protocol failure
`install_timeout`	Installation timed out

5.5.1 URL patterns

`/login/api/v1/installation/`

5.5.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `product`, `status`, `timestamp`.

account_id Accepts a comma-separated list

account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

begin ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

end ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

product_id Accepts a comma-separated list

5.5.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.5.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

product Product for which the status code was sent
status Status identifier
timestamp Time at which the status code was received, in isoformat

If filter queryparams are passed:

- A `ResourceList` containing those installation will be returned
- If no installation are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "installation": [
    {
        "product": 41,
        "status": "install_received",
        "timestamp": "2015-03-30T10:23:05Z"
    }
] }
```

5.5.2.2 POST

A POST request will start the installation process. Any query parameters not required for creation will be ignored.

- If no Gateway is linked to the account for the requested product, a 400 Bad Request will be returned.
- If the requested product does not exist, a 404 Not Found will be returned.
- If no connection could be setup to the Gateway or an error is received, a 502 Bad Gateway error will be returned
- If the installation is requested as non-secured, but the product requires secure installation, a 400 Bad Request will be returned
- If the resource is created successfully, a 202 ACCEPTED http code will be returned. A GET to the installation resource, with an optional product filter, can be performed to list the historic and current installation state.

Required fields `product`
Optional fields `secure`, `timeout`

Starting an installation is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "product": 41
}
```

5.5.2.3 DELETE

A DELETE request is used to abort an installation. If the installation abort message has been sent a 204 No Content is returned. To verify the gateway has received the abort request, perform a GET to the resourcelist. If successful it will contain an installation resource with status `install_aborted`.

5.6 Deinstallation

The deinstallation resource is used to deinstall or "unpair" physical Z-wave products from virtual products. This resource is a wrapper for the deinstallation procedure and doesn't directly map to any database resources.
The deinstallation API can be used to start and stop the deinstallation process and it will report back feedback on the current state of the deinstallation process.
The product field is required, but it is possible to pass `null` as value to allow deinstallation of a physical product which is unknown in the network. Possible `type` statuses are:

Type	Meaning
`deinst_start`	Deinstall request succesfully sent to Gateway
`deinst_accepted`	Deinstall request received by Gateway
`deinst_searching`	Gateway ready for nodeinfo
`deinst_found`	Gateway received nodeinfo
`deinst_success`	Deinstallation successful
`deinst_wrong_dev`	Wrong product was deinstalled
`deinst_not_modified`	Node was removed from a different network or not installed
`deinst_aborted`	Deinstallation was aborted
`deinst_internal`	Internal Gateway error
`deinst_busy`	Driver timeout
`deinst_proto_fail`	Protocol failure
`deinst_timeout`	Installation timed out

5.6.1 URL patterns

`/login/api/v1/deinstallation/`

5.6.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `product`, `status`, `timestamp`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
begin ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
end ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
product_id Accepts a comma-separated list

5.6.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.6.2.1 GET

A GET returns the relevant `ResourceList`. The resource contains the following fields:

product Product for which the status code was sent
status Status identifier
timestamp Time at which the status code was received, in isoformat

If filter queryparams are passed:

- A `ResourceList` containing those deinstallation will be returned
- If no deinstallation are relevant or selected, an empty list will be returned.

Output JSON

```
ResourceList = { "deinstallation": [
    {
        "product": null,
        "status": "deinst_start",
        "timestamp": "2015-08-18T08:42:50.315Z"
    },
    {
        "product": null,
        "status": "deinst_accepted",
        "timestamp": "2015-08-18T08:42:50.534Z"
    },
    {
        "product": null,
        "status": "deinst_searching",
        "timestamp": "2015-08-18T08:42:50.581Z"
    },
    {
        "product": null,
        "status": "deinst_found",
        "timestamp": "2015-08-18T08:42:57.377Z"
    },
    {
        "product": null,
        "status": "deinst_not_modified",
        "timestamp": "2015-08-18T08:42:59.377Z"
    }
] }
```

5.6.2.2 POST

A POST request will start the deinstallation process. Any query parameters not required for creation will be ignored. The product field is required, but it is possible to pass `null` as value to allow deinstallation of a physical product which is unknown in the network.

Required fields `product`
Optional fields `timeout`

Starting a deinstallation is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "product": 41
}
```

5.6.2.3 DELETE

A DELETE request is used to abort an installation. If the installation abort message has been sent a 204 No Content is returned. To verify the gateway has received the abort request, perform a GET to the resourcelist. If successful it will contain an installation resource with status `install_aborted`.

5.7 Synchronize

The synchronize resource allows for the synchronize of rules for 1 or more gateways. Synchronization should always be performed after installing products to ensure the Gateway is configured correctly. Possible statuses for synchronization are:

Status	Meaning
`uninitialized`	The Gateway has never been synchronized
`accepted`	The synchronization request has been accepted
`sending`	Update is being sent to the Gateway
`verifying`	Update is being verified by the Gateway
`programming`	Update is being installed in the Gateway
`success`	Update successful
`error`	Update failed

5.7.1 URL patterns

```
/login/api/v1/synchronize/
/login/api/v1/gateways/1/synchronize/
```

5.7.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `gateway`, `status`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
gateway_id Accepts a comma-separated list

5.7.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.7.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

gateway Parent gateway identifier
status Synchronization status

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that sychronization doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those synchronize will be returned
- If no synchronize are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "gateway": 1,
    "status": "success"
}

ResourceList = { "synchronize": [
    {
        "gateway": 1,
        "status": "accepted"
    }
] }
```

5.7.2.2 POST

A POST request will synchronize the rules with the Gateway. This will ensure the Gateway is up-to-date.

Required fields –
Optional fields –

Starting a sychronization is done by a POST to the resource. This object has no required fields. Anything added to the POST body will be ignored.
If the Gateway synchronization is started succesfully a 202 ACCEPTED http code will be returned. A Location header containing a canonical url to the status resource will also be passed.

5.8 Pendingdatas

Pending data resources allows tracking of when and if values are sent to a specific product.

5.8.1 URL patterns

```
/login/api/v1/pendingdata/
/login/api/v1/pendingdata/16599/
/login/api/v1/gateways/65/pendingdata/
/login/api/v1/gateways/65/pendingdata/16599/
```

5.8.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `channel`, `command_class`, `eta`, `gateway`, `node`, `parameter`, `pendingdata`, `protocol`, `status`, `timestamp`, `value`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
begin ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
command_class_id Accepts a comma-separated list
end ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
gateway_id Accepts a comma-separated list
node_id Accepts a comma-separated list
pendingdata_id Accepts a comma-separated list

5.8.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

5.8.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. The resource contains the following fields:

channel Channel ID
command_class Command Class
eta Estimated time at which the value will be sent
gateway Parent account identifier
node Node ID
parameter Parameter
pendingdata Resource identifier
protocol Protocol identifier
status Status
timestamp Time at which the data was sent to the gateway
value The encoded value to be sent

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that pendingdata doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those pendingdatas will be returned
- If no pendingdatas are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "channel": 0,
    "command_class": 132,
    "eta": "2015-08-19T15:39:33Z",
    "gateway": 65,
    "node": 2,
    "parameter": 0,
    "pendingdata": 16599,
    "protocol": "zwave",
    "status": "wakeup",
    "timestamp": "2015-08-19T15:39:33Z",
    "value": "100E00000100000000000000"
}

ResourceList = { "pendingdatas": [
    {
        "channel": 0,
        "command_class": 132,
        "eta": "2015-08-19T15:39:33Z",
        "gateway": 65,
        "node": 2,
        "parameter": 0,
        "pendingdata": 16599,
        "protocol": "zwave",
        "status": "wakeup",
        "timestamp": "2015-08-19T15:39:33Z",
        "value": "100E00000100000000000000"
    },
    {
        "channel": 0,
        "command_class": 132,
        "eta": "2016-11-16T10:24:46Z",
        "gateway": 65,
        "node": 5,
        "parameter": 0,
        "pendingdata": 17568,
        "protocol": "zwave",
        "status": "wakeup",
        "timestamp": "2016-11-16T10:17:40Z",
        "value": "201C00000100000000000000"
    }
] }
```

5.8.2.2 POST

A POST request will create a new pendingdata. The pendingdata will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error.

Required fields `gateway`, `protocol`, `node`, `command_class`

Optional fields `channel`, `parameter`, `value`

Creating a pendingdata is done by passing the resource as JSON data with POST. An example of the POST body is

```
{
  "gateway": 65,
  "protocol": "zwave",
  "node": 5,
  "command_class": 112
}
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

5.8.2.3 DELETE

A DELETE request is used to remove a pendingdata. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an pendingdata id *WILL* delete multiple pendingdatas. It is *NOT* recommend send a DELETE request without an pendingdata id or filtering query parameters.

6 Energy asset reporting

6.1 Energyassetcategories

6.1.1 URL patterns

```
/login/api/v1/energyassetcategories/
/login/api/v1/energyassetcategories/7/
```

6.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `energyassetcategory`, `name`.

energyassetcategory_id Accepts a comma-separated list

6.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

6.1.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

energyassetcategory Resource identifier

name Category name

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that energyassetcategory doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those energyassetcategories will be returned
- If no energyassetcategories are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "energyassetcategory": 7,
    "name": "building_related_energy"
}

ResourceList = { "energyassetcategories": [
    {
        "energyassetcategory": 6,
        "name": "usage"
    },
    {
        "energyassetcategory": 7,
```

```
        "name": "building_related_energy"
    },
    {
        "energyassetcategory": 8,
        "name": "live_energy_generating"
    }
] }
```

6.2 Energyassets

6.2.1 URL patterns

```
/login/api/v1/energyassets/
/login/api/v1/energyassets/1/
/login/api/v1/accounts/5/energyassets/
/login/api/v1/accounts/5/energyassets/1/
```

6.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `bundles`, `energyasset`, `validated`.

account_id Accepts a comma-separated list

account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

energyasset_id Accepts a comma-separated list

validated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

validated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

validated_null Accepts a boolean value: `true` or `false`

6.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

6.2.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier

bundles Lists the various measurement categories, estimated by month

energyasset Resource identifier

validated Date on which energyasset was officially done, may be `null` if the asset is not finished yet

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that energyasset doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those energyassets will be returned
- If no energyassets are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 5,
    "bundles": {
        "10": [92, 209, 368, 635, 709, 750, 713, 619, 407, 252, 131, 109],
        "6": [469, 402, 409, 349, 335, 314, 310, 318, 335, 375, 411, 467],
        "7": [348, 292, 284, 225, 204, 187, 177, 186, 209, 248, 290, 346],
        "9": [4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
    },
    "energyasset": 1,
    "validated": null
}

ResourceList = { "energyassets": [
    {
        "account": 5,
        "bundles": {
            "10": [92, 209, 368, 635, 709, 750, 713, 619, 407, 252, 131, 109],
            "6": [469, 402, 409, 349, 335, 314, 310, 318, 335, 375, 411, 467],
            "7": [348, 292, 284, 225, 204, 187, 177, 186, 209, 248, 290, 346],
            "9": [4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
        },
        "energyasset": 1,
        "validated": "2018-01-01T00:00:00Z"
    }
] }
```

6.3 Energyassetproperties

6.3.1 URL patterns

```
/login/api/v1/energyassetproperties/
/login/api/v1/energyassetproperties/2/
/login/api/v1/energyassets/1/energyassetproperties/
/login/api/v1/energyassets/1/energyassetproperties/2/
/login/api/v1/accounts/5/energyassetproperties/
/login/api/v1/accounts/5/energyassetproperties/2/
/login/api/v1/accounts/5/energyassets/1/energyassetproperties/
/login/api/v1/accounts/5/energyassets/1/energyassetproperties/2/
```

6.3.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `energyasset`, `energyassetcategory`, `energyassetproperty`, `property`.
account_id Accepts a comma-separated list
account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.
energyasset_id Accepts a comma-separated list
energyassetcategory_id Accepts a comma-separated list
energyassetproperty_id Accepts a comma-separated list
validated_after ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
validated_before ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.
validated_null Accepts a boolean value: `true` or `false`

6.3.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

6.3.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

energyasset Parent energyasset identifier
energyassetcategory Parent energyassetcategory identifier
energyassetproperty Resource identifier
property Parent property identifier

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that energyassetproperty doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those energyassetproperties will be returned
- If no energyassetproperties are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "energyasset": 1,
    "energyassetcategory": 7,
    "energyassetproperty": 2,
    "property": 922
}

ResourceList = { "energyassetproperties": [
    {
        "energyasset": 1,
        "energyassetcategory": 6,
        "energyassetproperty": 1,
        "property": 918
    },
    {
        "energyasset": 1,
        "energyassetcategory": 9,
        "energyassetproperty": 2,
        "property": 919
    }
] }
```

7 Problem detection and resolution

7.1 Failuretypes

Provides extra information about the Failures which occured and what type of resource they are linked to.

7.1.1 URL patterns

```
/login/api/v1/failuretypes/
/login/api/v1/failuretypes/product/gateway/unavailable/
```

7.1.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `failuretype`, `impact`, `resourcetype`.

failuretype_id Accepts a comma-separated list

failuretype_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

7.1.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

7.1.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

failuretype Resource identifier

impact Impact

resourcetype Resource type

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that failuretype doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those failuretypes will be returned
- If no failuretypes are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "failuretype": "product/gateway/unavailable",
    "resource_type": "gateway"
}
```

```
ResourceList = { "failuretypes": [
    {
        "failuretype": "product/gateway/unavailable",
        "resource_type": "gateway"
    },
    {
        "failuretype": "product/general/unavailable",
        "resource_type": "peripheral"
    }
] }
```

7.2 Failures

Provides a list of all failures associated with an account. Failures will be automatically be removed after 6 months. To get an overview of the latest status of all products use the `latest=true` query parameter.

7.2.1 URL patterns

```
/login/api/v1/failures/
/login/api/v1/failures/130/
/login/api/v1/accounts/1/failures/
/login/api/v1/accounts/1/failures/130/
```

7.2.1.1 Query parameters

Query parameters are extra resource list filters that can be passed with a GET request. These parameters are always optional. If after filtering no resources remain, an empty resource list will be returned.

fields Comma-separated list, if this parameter is used only the selected fields will be show. Options are: `account`, `failure`, `failuretype`, `resource`, `status`, `timestamp`.

account_id Accepts a comma-separated list

account_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

account_search Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

begin ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

end ISO-8601 timestamp. The following formats are possible: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm:ss`, `YYYY-MM-DDTHH:mm:ss(Z|+-13:00)`.

failure_id Accepts a comma-separated list

failuretype_name Lookups are *NOT* case sensitive. Complex lookups are possible using the `%` and _ symbol. The `%`-symbol match any character for any amount. The _-symbol matches any character once. This syntax matches normal `LIKE` lookups in SQL.

impact Accepts a comma-separated list

latest Accepts a boolean value: `true` or `false`

resource_id Accepts a comma-separated list

status_str Accepts a comma-separated list

7.2.2 Usable HTTP methods

The following section describes what HTTP methods are available for this resource. It also describes possible side-effect and return codes for API-calls.

7.2.2.1 GET

A GET returns the relevant `Resource` or `ResourceList`. This resources allows for pagination using the `Range` header. See the introduction chapter for more information on the implementation. The resource contains the following fields:

account Parent account identifier

failure Resource identifier

failuretype Parent failuretype identifier

resource Resource type identifier pk

status Status of failure, options are: `no_failure`, `failed`, `accepted`

timestamp Timestamp for failure isoformat

If the URL is passed without id:

- The `ResourceList` will be returned

If an id is passed in the URL:

- A `Resource` will be returned.
- A 404 NOT FOUND will be returned if that failure doesn't exist

If filter queryparams are passed:

- A `ResourceList` containing those failures will be returned
- If no failures are relevant or selected, an empty list will be returned.

Output JSON

```
Resource = {
    "account": 1,
    "failure": 128,
    "failuretype": "product/gateway/unavailable",
    "resource": 1,
    "status": "no_failure",
    "timestamp": "2018-05-07T00:00:00Z"
}

ResourceList = { "failures": [
    {
        "account": 1,
        "failure": 128,
        "failuretype": "product/gateway/unavailable",
        "resource": 1,
        "status": "no_failure",
        "timestamp": "2018-05-07T00:00:00Z"
    },
    {
        "account": 1,
        "failure": 130,
        "failuretype": "product/general/unavailable",
        "resource": 5,
        "status": "failed",
        "timestamp": "2018-05-14T07:18:13.988Z"
    }
] }
```

7.2.2.2 POST

A POST request will create a new failure. The failure will automatically be linked to the API key. Any query parameters not required for creation will be ignored. A POST request with an id will result in a 404 NOT FOUND error. This resource allows the bulk creation. To create multiple resources in 1 request, send all resources in a list. Read the introduction chapter for more information on this feature.

Required fields `failuretype`, `account`, `status`, `timestamp`
Optional fields `resource`

Creating a failure is done by passing the resource as JSON data with POST. An example of the POST body is

```
  {
    "failuretype": "product/gateway/unavailable",
    "account": 1,
    "status": "failed",
    "timestamp": "2018-05-14T07:18:13.988Z"
  }
```

If the resource is created succesfully a 201 CREATED http code will be returned, along with the id of the created resource. A Location header containing a canonical url to the created resource will also be passed.

7.2.2.3 PUT

A PUT request is used to update fields of a failure. If a field is passed that cannot be updated, a 400 Bad Request error will be returned. If a field is passed that is not recognized it will be ignored.

Updatable fields `failuretype`, `account`, `status`, `resource`

Updating a failure is done by passing the values to be updated as JSON data with a PUT. An example of the PUT body is

```
{
  "failuretype": "product/general/unavailable",
  "account": 5,
  "status": "no_failure",
  "resource": null
}
```

A PUT request with filtering query parameters or without an failure id *WILL* update multiple failures. It is *NOT* recommend to send a PUT request without an failure id or filtering query parameters.
A successful PUT request will return a 200 OK http code with an empty response body.

7.2.2.4 DELETE

A DELETE request is used to remove a failure. If the resource does not exist a 404 NOT FOUND is returned. If the resource is succesfully deleted a 204 NO CONTENT is returned.
A DELETE with filtering query parameters or without an failure id *WILL* delete multiple failures. It is *NOT* recommend send a DELETE request without an failure id or filtering query parameters.