NODA EnergyView API v1

API documentation

NODA Intelligent Systems AB

2022-01-14

Contents

API Reference

The API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

API notes

HTTP headers

Two HTTP headers are **required** when performing a request, `Authorization` and `Accept`.

The `Authorization` header is explained in the section Authentication.

The `Accept` header should always be `application/json`. Otherwise, all error messages will be returned in `HTML` and not `JSON`.

HTTP methods

The HTTP verbs `GET`, `PUT`, `POST` and `DELETE` are used for `retrieving`, `changing`, `creating` and `deleting` content.

Unless stated otherwise, all verbs (`PUT`, `POST`, `DELETE`) except `GET` use `Content-type: application/x-www-form-urlencoded` and the `query string` is sent in the request body. This is not allowed with `GET`, and the `query string` is therefore sent in the `URL`.

```
GET /domain/api/v1/example?user_name=helge HTTP/1.0
Accept: application/json
Authorization: Key ...

AND

DELETE /domain/api/v1/example HTTP/1.0
Accept: application/json
Authorization: Key ...
Content-type: application/x-www-form-urlencoded

user_name=helge
```

In cases where a JSON object is used as a payload to a `POST` or `PUT` request. The `Content-type` is usually `application/json` if the request requires no arguments and the only payload is the JSON object itself. It is clearly stated in the section for that request in these cases.

Authentication

The API uses API keys to authenticate requests. You can view and manage API keys in the account page if you have administrative access.

Your API keys carry many privileges, so be sure to keep them secure.

Do not share your secret API keys in publicly accessible areas such as social media, client-side code, and so forth.

Authentication to the API is performed via HTTP Key Auth. Provide your API key as the auth key value. You do not need to provide a password. For example;

> Authorization: Key c4bbcb1fbec99d65...9afa73bd4e39a8a

All API requests must be made over `HTTPS`. *API requests without authentication will fail.*

> The API key for your account can be found here

Errors

NODA uses conventional HTTP response codes to indicate the success or failure of an API request.

In general: Codes in the `2xx` range indicate success. Codes in the `4xx` range indicate an error that failed given the information provided (e.g., a required parameter was omitted, a request failed, etc.). Codes in the `5xx` range indicate an error with NODA's servers (these are rare).

Some `4xx` errors that could be handled programmatically can include an error code that briefly explains the error reported.

Attributes	
type string *optional*	The type of error returned. One of api_connection_error, api_error, authentication_error, invalid_request_error or rate_limit_error
code string *always*	For some errors that could be handled programmatically, a short string indicating the error code reported.

Attributes	
doc_url string *optional*	A URL to more information about the error code reported.
error string *always*	A human-readable message providing more details about the error. Sometimes, these messages can be shown to your users.
param string *optional*	If the error is parameter-specific, the parameter related to the error. For example, you can use this to display a message near the correct form field.

HTTP Status Code Summary

200 - OK	Everything worked as expected.
400 - Bad Request	The request was unacceptable, often due to missing a required parameter.
401 - Unauthorized	No valid API key provided.
402 - Request Failed	The parameters were valid but the request failed.
403 - Forbidden	The API key doesn't have permissions to perform the request.
404 - Not Found	The requested resource doesn't exist.
409 - Conflict	The request conflicts with another request.
429 - Too Many Requests	Too many requests hit the API too quickly. We recommend an exponential backoff of your requests.
500, 502, 503, 504 - Server Errors	Something went wrong on NODA's end. (These are rare.)

Error Types

api_connection_error	Failure to connect to NODA's API.
api_error	API errors cover any other type of problem (e.g., a temporary **problem with NODA's servers), and are extremely uncommon.
authentication_error	Failure to properly authenticate yourself in the request.
invalid_request_error	Invalid request errors arise when your request has invalid parameters.
rate_limit_error	Too many requests hit the API too quickly.

Handling errors

Our system raises exceptions for many reasons, invalid parameters, authentication errors, and network unavailability. Therefore, we recommend writing code that gracefully handles all possible API exceptions.

Glossary

`Cluster` A.k.a `group`. A free-formed collection of nodes. For which the purpose depends on the context.

`Device` A sub-set of the possible sensors declared in a `protocol`. Used to distinguish between entities with the same `protocol`. For example *indoor climate sensors* from different manufacturers.

`Domain` A segregated storage location. See Domains.

`Node` A.k.a `collector`. An entity with a collection of tags (sensors), defined by the `device` type. Typically a single control point, a weather forecast (for a location) or a single indoor climate point.

`Protocol` A set of possible (but not always used) sensors as a *named* set.

`Scout` Legacy (yet not deprecated), name for the `protocol` corresponding with an *indoor climate sensor*. Scout was the name of Nodas' own indoor temperature sensors, for which manufacturing ended in 2014.

`Tag` A.k.a sensor (name). A stored time-series measurement. Tag names are required to be distinct within a `protocol`, and as a result, also `device` and `node`.

`vDER` **V**irtual **D**ynamic **E**nergy **R**esource.

Domains

Storage is separated into domains. Typically a domain refers to a location, a smaller customer/organization or a business division.

It all depends on circumstance and preference.

As each domain acts as a ”walled of garden”, data can not be cross queried between domains without access to all target domains. A domain name usually begins with a ~ (tilde) character in front. For example, `~mydomain`, or in a URL; `http://customer.noda.se/~mydomain`.

> As of **May 2020** this is not enforced any longer, and you can use URLs with or without the ~ (tilde) character.

However, in writing, it is still recommended to use the ~ (tilde) to emphasize that the token you are referring to is a domain. As this documentation, most of the time refers to a domain in the context of an example URL. We have chosen to skip the ~ (tilde) character. — pagetitle: “Request rates” icon: tachometer-alt —

Allowed Request Rates

TL;DR

To minimise the load on the API, you are **not allowed** to perform more than **10 requests per second** on a specific `domain`.

API limits

We need to separate `GET` and `SET` requests, as these two kinds of requests affect the rate in different ways.

- `GET` requests fetch data from the system.
- `SET` requests store/modify data in the system.

The difference between these two requests has to do with `locking`. While one request (req. A) is storing data, another request (req. B) can not store data at the same location (i.e. measurement). This is because there can only be one process at a time accessing the underlying database table.

Therefore, doing several hundred separate store procedures simultaneously (and to the same location) will not be more efficient but less efficient. Each process will stall while waiting for another process to finish its task.

Fetching data (`GET`) is not bound by this restriction, and a client may (if available resources allow it) make as many simultaneous requests as it needs.

Note: Request limits are per `domain`. Thus, one may send multiple requests to different domains without causing the rate limit to trigger.

However, the recommendation is that requests remain as few and small as possible. As fetching all values from 1000+ nodes at a time will likely cause either `memory exhaustion` (500 error) or `take too long` to execute. Therefore, splitting such a large request into several smaller requests with a subset of nodes per request is highly recommended, and **will likely be an enforced requirement in the future**.

Each request may contain data for more than one node, so the possible throughput depends on the clients' behaviour.

Note: a single data-point insert takes on average between 5ms and 15ms. A set bottleneck that can not be improved by any form of batching requests over the REST API.

The error returned when a limit has been reached is `429 Too many requests`.

Client Example

A client which POSTs data to `/v1/timeseries` using a JSON object with a single `node_id`, a single `tag`, and a single `data` point will manage `10 data-points per second`.

While a client which POSTs data to `/v1/timeseries` using a JSON object with five different `node_id`, the same `tag` for all of them (or different tags) and with a single `data` point for each item. Will manage `50 data points per second`.

API requests

- CSV import
- Nodes/Collectors
- Sensors/Tags
- Settings (Metadata)
- Time series
- Network manager
- vDer control plan

CSV import

Push data to EnergyView using CSV files.

The structure of the CSV file is managed via the integration page.

List CSV import integrations

> **GET** */domain*/api/v1/csvimport

List all existing import definitions.

URI parameters

There are no specific parameters for this request.

Response example

```
{
    "integrations": [
        {
            "uuid": "945978fd-bd61-4d83-a1e4-02ef89d60987",
            "title": "My CSV importer"
        },
        {
            "uuid": "53a13531-6430-4014-8155-f8d05df12e68",
            "title": "Price forecast importer"
        }
    ]
}
```

`integrations` **(required, array)** Named wrapper attribute. A list (array) of objects.
`uuid` **(required, string)** Unique identifier for the integration.
`title` **(required, string)** Name of the integration.

Request example (cURL and Python)

```
# GET /domain/api/v1/csvimport
# Using cURL CLI
curl -s \
```

```
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/csvimport'

# GET /domain/api/v1/csvimport
# Using Python and Requests
import requests
url = 'https://customer.noda.se/domain/api/v1/csvimport'
hdr = {
  "accept": "application/json",
  "authorization": "Key 0123456789abcdef0123456789abcdef"
}
r = requests.get(url, headers=hdr)
print(r.json())
```

POST */domain*/api/v1/csvimport/{uuid}

Upload a CSV file and import the parsed data into the time series database.

URI parameters

`{uuid}` is the id of the CSV import integration.

> The `Content-Type` for this request must be: `multipart/form-data`.

Response example

```
OK
```

Code example

```
# POST /domain/api/v1/csvimport/{uuid}
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X POST \
-F 'file=@example_data.csv'\
'https://customer.noda.se/mydomain/api/v1/csvimport/53a13531-6430-4014-8155-f8d05df12e68'
```

Status codes

`400` Invalid request.
`404` No such integration (invalid UUID).
`500` Internal server error. An invalid state occurred.
`200` OK. With the response content.

Nodes/Collectors

List information about all nodes/collectors.

> **Endpoints**
> **POST** */domain*/api/v1/node
> **GET** */domain*/api/v1/nodes
> **GET** */domain*/api/v1/node/`{id}`
> **PUT** */domain*/api/v1/node/`{id}`
> **DELETE** */domain*/api/v1/node/`{id}`
> **PUT** */domain*/api/v1/node/`{id}`/provision

Create a new Node

Perform a `POST` request with `Content-Type: application/json` and the following content to the endpoint `/*domain*/api/v1/node`.

```
{
    "name": "My testnode",
    "device": "Kelp-Basic",
    "designation": "building",
    "description": "Node for testing",
    "dataif": "ca:fe:c0:c0:a0:00",
    "tags": [
        "outdoortemp",
        "returntemp_sec",
        "supplytemp_sec"
    ],
    "parent": "c7a386c4-3f7c-4784-aad6-ffd04aca0410"
}
```

`uuid` **(optional string)** Use a provided UUID when creating a new Node.
`name` **(required string)** Name of the new Node

`device` **(required string)** Device type of the Node. Determines which tags can be assigned to it. Options depend on how the environment is configured.

`designation` **(required string)** Purpose of the node, allowed options are; `building`, `building/residential`, `building/commercial`, `building/industrial`, `energymeter`, `energymeter/heat`, `energymeter/cool`, `circuit`, `circuit/heat`, `circuit/cool`, `sensor`, `sensor/indoor`, `sensor/outdoor`, `network/production`, `network/station` and `data`. *These options may be tailored for the environment. The list given here is the default items.*

`description` **(optional string)** A description of the Node. Useful to distinguish between similarly named Nodes.

`dataif` **(optional string)** Key used to identify a data source in a foreign system.

`tags` **(optional array of strings)** The list of tags assigned to the Node.

`parent` **(optional array of strings)** Either an integer Node-id or a string Node UUID. Creates a child -> parent relationship between Nodes. Cyclic dependencies are not allowed.

Status codes

`400` Invalid user request.
`409` Conflict. A Node with the same UUID already exists.
`500` Internal server error. An invalid state occurred.
`201` Created.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X POST \
-d '{"name": My testnode", "device": "Kelp-Basic", "designation": "building"}'
'https://customer.noda.se/mydomain/api/v1/node'
```

Response

```
Content-Type: application/json
```

```
{
    "id": 60,
```

```
    "uuid": "ccb286c5-d13f-4083-9e02-db2d5bafe27b"
}
```

`id` **(required integer)** Id of the newly created Node.
`uuid` **(required string)** UUID of the newly created Node.

List all Nodes

Perform a `GET` request to the endpoint `/*domain*/api/v1/nodes`.

Status codes

`400` Invalid user request.
`404` No such Node.
`500` Internal server error. An invalid state occurred.
`200` OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/nodes'
```

Response

```
Content-Type: application/json
```

```
{
    "nodes": [{
        "id": 60,
        "uuid": "ccb286c5-d13f-4083-9e02-db2d5bafe27b",
        "name": "Exempel heatingsystem",
        "description": "",
        "dataif": "00:3f:dd:ab:33:b1:7f:44:23:56",
        "public": true,
        "owner": false,
        "enabled": true,
        "archived": false,
```

```
        "designation": "circuit/heat",
        "device": "Kelp-Basic",
        "tags": [
            "outdoortemp",
            "returntemp_sec",
            "supplytemp_sec"
        ],
        "interval": 600,
        "parent": {
            "id": null,
            "uuid": null
        },
        "children": []
    }]
}
```


`nodes` **(required array)** Array of objects

`id` **(required, integer)** Domain-unique id of the Node.

`uuid` **(required, string)** An UUIDv4 ID which *should* be unique across domains. Since `id` is unique on a single domain.

`name` **(required, string)** Human-readable name of the Node.

`description` **(required, string)** Human-readable description of the Node.

`dataif` **(required, string)** Data interface declaration. Used for tracking references to external sources.

`public` **(required, boolean)** Boolean to declare if this node is public to all accounts in the domain.

`owner` **(required, boolean)** Boolean to show if the account performing the request is the owner of the Node or not.

`enabled` **(required, boolean)** State used by several different components to determine if computations should use the Node.

`archived` **(required, boolean)** An alternative to deleting the node and losing everything.

`designation` **(required, string)** Machine centric purpose declaration of the Node. Used by several different components of the system.

`device` **(required, string)** Legacy purpose declaration of the Node. Needed as it is a central part of the design. A device is a sub-set of a Protocol.

`tags` **(required, array of strings)** A list of tags assigned to this Node.

`interval` **(required, integer)** A period in seconds of how often data is expected to be stored on the Node. Used by (for example) the alert system to determine faults.

`parent` **(required, object)** An object with the two keys, `id` and `uuid` pointing to one Node as a parent. Both keys will have the value `null` if no parent has been assigned.

`children` **(required, array of objects)** A list of objects with the keys `id` and `uuid` pointing to a Node as a child. If the Node has no children the result is an empty list `[]`.

Get details about a single Node

Perform a `GET` request to the endpoint `/*domain*/api/v1/node/{id}`.

NOTE: The `{id}` can be either the `integer` identifier for the Node or the `UUID` identifier.

Status codes

`400` Invalid user request.
`404` No such Node.
`500` Internal server error. An invalid state occurred.
`200` OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/node/60'
```

Response

```
Content-Type: application/json
```


```
{
    "id": 60,
    "uuid": "ccb286c5-d13f-4083-9e02-db2d5bafe27b",
    "name": "Exempel heatingsystem",
    "description": "",
    "dataif": "00:3f:dd:ab:33:b1:7f:44:23:56",
    "public": true,
    "owner": false,
    "enabled": true,
    "archived": false,
    "designation": "circuit/heat",
    "device": "Kelp-Basic",
    "tags": [
```

```
        "outdoortemp",
        "returntemp_sec",
        "supplytemp_sec"
    ],
    "interval": 600
}
```

`id` **(required, integer)** Domain-unique id of the Node.

`uuid` **(required, string)** An UUIDv4 ID which *should* be unique across domains. Since `id` is unique on a single domain.

`name` **(required, string)** Human-readable name of the Node.

`description` **(required, string)** Human-readable description of the Node.

`dataif` **(required, string)** Data interface declaration. Used for tracking references to external sources.

`public` **(required, boolean)** Boolean to declare if this node is public to all accounts in the domain.

`owner` **(required, boolean)** Boolean to show if the account performing the request is the owner of the Node or not.

`enabled` **(required, boolean)** State used by several different components to determine if computations should use the Node.

`archived` **(required, boolean)** An alternative to deleting the node and losing everything.

`designation` **(required, string)** Machine centric purpose declaration of the Node. Used by several different components of the system.

`device` **(required, string)** Legacy purpose declaration of the Node. Needed as it is a central part of the design. A device is a sub-set of a Protocol.

`tags` **(required, array of strings)** A list of tags assigned to this Node.

`interval` **(required, integer)** A period in seconds of how often data is expected to be stored on the Node. Used by (for example) the alert system to determine faults.

`parent` **(required, object)** An object with the two keys, `id` and `uuid` pointing to one Node as a parent. Both keys will have the value `null` if no parent has been assigned.

`children` **(required, array of objects)** A list of objects with the keys `id` and `uuid` pointing to a Node as a child. If the Node has no children the result is an empty list `[]`.

Update a single Node

Perform a `PUT` request with `Content-Type: application/json` and the following content to the endpoint `/*domain*/api/v1/node/{id}`.

NOTE: The `{id}` can be either the `integer` identifier for the Node or the `UUID` identifier.

```
{
    "name": "My testnode",
    "description": "Node for testing",
    "designation": "building",
    "dataif": "ca:fe:c0:c0:a0:00",
    "public": true,
    "enabled": true,
    "interval": "01:00:00",
    "deployment_date": "2022-02-13T12:50:39+01:00",
    "tags": [
        "outdoortemp",
        "returntemp_sec",
        "supplytemp_sec"
    ],
    "parent": "c7a386c4-3f7c-4784-aad6-ffd04aca0410"
}
```


`name` **(optional string)** Name of the Node.

`description` **(optional string)** A description of the Node. Useful to distinguish between similarly named Nodes.

`designation` **(optional string)** Purpose of the node, allowed options are; `building`, `building/residential`, `building/commercial`, `building/industrial`, `energymeter`, `energymeter/heat`, `energymeter/cool`, `circuit`, `circuit/heat`, `circuit/cool`, `sensor`, `sensor/indoor`, `sensor/outdoor`, `network/production`, `network/station` and `data`. *These options may be tailored for the environment. The list given here is the default items.*

`dataif` **(optional string)** Key used to identify a data source in a foreign system.

`public` **(required, boolean)** Boolean to declare if this node is public to all accounts in the domain.

`enabled` **(optional boolean)** Shows if the Node is enabled (active) in the system. A disabled Node may still store data, but no control actions will occur unless they are "forced" via the API.

`interval` **(optional string)** Update interval on the format `HH:MM:SS`. The value may exceed 99h. Depending on the integration solution, one may use this to perform data retrieval or identify missing data.

`deployment_date` **(optional string)** RFC3339 string of when the Node is "deployed". Used to keep track of deployment dates.

`tags` **(optional array of strings)** The list of tags assigned to the Node. **NOTE** This change will replace the current assigned list of tags. Ensure all wanted tags are included in this request.

`parent` **(optional array of strings)** Either an integer Node-id or a string Node UUID. Creates a child -> parent relationship between Nodes. Cyclic dependencies are not allowed.

Status codes

`400` Invalid user request.
`404` No such Node.
`500` Internal server error. An invalid state occurred.
`204` No Content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X PUT \
-d '{"designation": "building/industrial", "tags": ["outdoortemp", "supplytemp_sec",
↪  "returntemp_sec"]}'
'https://customer.noda.se/mydomain/api/v1/node/ccb286c5-d13f-4083-9e02-db2d5bafe27b'
```

Response

There will be no content, only a `204` status response on success.

Delete a single Node

Perform a `DELETE` request to the endpoint `/*domain*/api/v1/node/{id}`.

NOTE: The `{id}` can be either the `integer` identifier for the Node or the `UUID` identifier.

Status codes

`400` Invalid user request.
`404` No such Node.
`500` Internal server error. An invalid state occurred.
`204` No Content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X DELETE
'https://customer.noda.se/mydomain/api/v1/node/ccb286c5-d13f-4083-9e02-db2d5bafe27b'
```

Response

There will be no content, only a `204` status response on success.

Provision a Node to enable steering service

Perform a `PUT` request with `Content-Type: application/json` and the following content to the endpoint `/*domain*/api/v1/node/{id}/provision`.

NOTE: The `{id}` can be either the `integer` identifier for the Node or the `UUID` identifier.

```
{
    "active": true,
    "balance_temperature": 17.0,
    "idt_wanted": 21.0,
    "idt_min": 19.0
}
```

`active` **(optional boolean)** State of the steering service. False = disabled, True = enabled. Defaults to `true`.

`balance_temperature` **(optional number)** The temperature at which the circulation pump is switched off. Defaults to `17.0`.

`idt_wanted` **(optional number)** The wanted average indoor temperature. Defaults to `22.0`.

`idt_min` **(optional number)** The lowest allowed indoor temperature from any indoor sensor. Defaults to `17.0`.

To provision a Node the following prerequisites are necessary;

- The Node is designated as either `building` or any sub-type (`building/%`), or as `circuit` or any sub-type (`circuit/%`).
- Indoor sensors are assigned as `children` to the control point. These Nodes must be designated as `sensor/indoor`.

Status codes

400 Invalid user request.
404 No such Node.
500 Internal server error. An invalid state occurred.
204 No Content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X PUT \
-d '{"balance_temperature": 16.0, "idt_min": 18.0}' \
'https://customer.noda.se/mydomain/api/v1/node/ccb286c5-d13f-4083-9e02-db2d5bafe27b/provision'
```

Response

There will be no content, only a 204 status response on success.

Sensors/Tags

List information about all sensors/tags.

> **Endpoints GET** */domain*/api/v1/tags

List all sensor declarations

Perform a GET request to the endpoint /*domain*/api/v1/tags.

The response is a JSON object on the format;

```
{
    "sensors": [
        {
            "id": 1,
            "name": "outdoortemp",
            "description": "Outdoor temperature sensor",
```

```
            "postfix": "C",
            "protocol_id": 1
        },
        ...
    ]
}
```

`sensors` **(required, array)** A list of objects.

`id` **(required, integer)** Domain-unique id of the sensor. A sensor (declaration) is usually assigned to multiple nodes. There may only be less than a hundred sensors in a domain with thousands of Nodes.

`name` **(required, string)** Declared name of the sensor used throughout the system. Unique in combination with `protocol_id`.

`description` **(required, string)** Human readable declaration of the sensor.

`postfix` **(required, string)** May serve as the unit.

`protocol_id` **(required, integer)** Reference to a protocol. A protocol is a set of sensors acting as a declaration of ability.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/tags'
```

Response

`sensors` will contain as many sensors as declared by the system.

```
{
    "sensors": [
        {
            "id": 1,
            "name": "outdoortemp",
            "description": "Outdoor temperature sensor",
            "postfix": "C",
            "protocol_id": 1
        }
    ]
}
```

Settings (Metadata)

Get and set values related to settings or metadata.

> **Endpoints GET** */domain*/api/v1/settings/`{type}`/`{id}`
> **PUT** */domain*/api/v1/settings/`{type}`/`{id}`

Retrieve settings

Perform a `GET` request and the following parameters to the endpoint `/*domain*/api/v1/settings/{type}/{id`

URL parameters

`type` One of;

- cluster
- gridmgr *deprecated*
- node
- report
- netmgr

`id` The numeric id associated with the resource.

Body parameters

`path` **(optional, string)** Filter using the specified path. Separator is '.' (dot). For example: `coco.default`.

`extract` **(optional, integer)** Unique identifier for the integration. When set to `1`. Everything beneath `path` will be extracted and returned. For example;

```
extract=0 and path=coco.default
Result:
{"coco": {"default": {"hello": "world"}}}
Compared to:
extract=1 and path=coco.default
Result:
{"hello": "world"}
```

```
Or:
extract=1 and path=coco.default.hello
Result:
"world"
```

Request example (cURL)

Request

```
# GET /domain/api/v1/settings/{type}/{id}
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/settings/node/1?path=coco.default'
```

Response

```
{
    "coco": {
        "default": {
            "control": {
                "balance_temperature": 17,
                "bias_hour_cap_charge": -30,
                "bias_hour_cap_discharge": 32,
                "bias_hour_recovery_time": 24
            }
        }
    }
}
```

Store settings

Perform a `PUT` request and the following parameters to the endpoint `/*domain*/api/v1/settings/{type}/{id}`

URL parameters

`type` One of;

- cluster
- gridmgr *deprecated*

- node
- report
- netmgr

`id` The numeric id associated with the resource.

Body parameters

`path` **(required, string)** Target path. Separator is '.' (dot). For example: `coco.default`.

`value` **(required, string (JSON))** Any JSON compatible value. Objects are not supported and will be converted to a string.

`force` **(optional, integer)** Force the new value to be written. Even if the type (boolean, string, integer, float) is different from the old value. The only allowed value is: `1`.

Request example (cURL)

Request

```
# PUT /domain/api/v1/settings/{type}/{id}
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-d 'path=coco.default.control.balance_temperature&value=12' \
-X PUT \
'https://customer.noda.se/mydomain/api/v1/settings/node/1'
```

Response

```
{
    "value": 12
}
```

Time series

Store and retrieve time series data.

> **Endpoints**

> **GET** */domain*/api/v1/timeseries
> **POST** */domain*/api/v1/timeseries

Example of a generalized reply object


```
{
    "timeseries": [
        {
            "node_id": 1,
            "node_uuid": "2e949492-9008-4f81-a360-9059a769b195",
            "tag": "outdoortemp",
            "data": [
                {
                    "v": 2.6,
                    "ts": "2020-01-01T00:05:57+01:00"
                },
                {
                    "v": 2.6,
                    "ts": "2020-01-01T00:15:37+01:00"
                }
            ]
        }
    ]
}
```


`timeseries` **(required, array)** Named wrapper attribute. A list (array) of objects.
`node_id` **(required, integer)** Domain-unique identifier for the *node*.
`node_uuid` **(required, integer)** UUID identifier for the *node*.
`tag` **(required, string)** Name of the sensor for which the data belongs.
`data` **(required, array)** A list (array) of data points as an object.
`v` **(required, numeric)** Value of the data point.
`ts` **(required, string)** Date time string on the format `YYYY-MM-DDThh:mm:ss±hh:mm`.

Retrieve datapoints

Perform a `GET` request with the following attributes to the endpoint `/*domain*/api/v1/timeseries`.

`node_id` **(optional, integer)** Filter on the unique identifier for a specific node.

`node_ids` **(optional, array of integers)** Filter on several unique node identifiers. Can not be used together with `node_id`. The paramater needs to be `JSON` encoded. **Example:**

`node_ids=[1, 67, 995]`.

`tag` **(optional, string)** Filter on a sensor name.

`tags` **(optional, array of strings)** Filter on several sensor names. Can not be used together with `tag`. The paramater needs to be `JSON` encoded. **Example:** `tags=["outdoortemp", "supplytemp_sec", "returntemp_sec"]`.

`start` **(optional, string)** The "from" date-time string on the format `YYYY-MM-DDThh:mm:ss±hh:mm`. Without timezone information, the API will fall back to the time zone configured for the domain. Defaults to **now**.

`end` **(optional, string)** The "to" date-time string on the format `YYYY-MM-DDThh:mm:ss±hh:mm`. Without timezone information, the API will fall back to the time zone configured for the domain. Defaults to **now**.

`resolution` **(optional, string)** Truncates all timestamps to any of (options), then computes the average for all points within the truncated period.

- second
- minute
- 5minute
- 10minute
- 15minute
- 20minute
- 30minute
- hour
- day
- month
- year
- decade
- century
- millennia

`aggregate` **(optional, string)** When using `resolution`. Select this aggregate function instead of the default `avg` when computing the result.

- avg
- min
- max
- sum
- count

`epoch` **(optional, integer)** When enabled (set to 1), the `ts` field will be in Unix timestamp format (numeric) instead of a string. This is the number of seconds that have elapsed since the `Unix epoch`, the time `00:00:00 UTC on 1 January 1970`.

Request example (cURL)

Request

```
# GET /domain/api/v1/timeseries
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/timeseries?node_id=1&tag=outdoortemp&start=2020-01-01T00:0
```

Response

```
{
    "timeseries": [
        {
            "node_id": 1,
            "tag": "outdoortemp",
            "data": [
                {
                    "v": 2.6,
                    "ts": "2020-01-01T00:05:57+01:00"
                },
                {
                    "v": 2.6,
                    "ts": "2020-01-01T00:15:37+01:00"
                }
            ]
        }
    ]
}
```

Store a single data point

Perform a `POST` request with `Content-Type: application/x-www-form-urlencoded` and the following attributes to the endpoint `/*domain*/api/v1/timeseries`.

`node_id` **(required|optional, integer)** Domain-unique id for the node. Required unless `node_uuid` is used.

`node_uuid` **(required|optional, integer)** UUID for the node. Required unless `node_id` is used.

`tag` **(required, string)** The name of the tag/sensor as declared in EnergyView. Such as `outdoortemp`, `indoortemp`, `supplytemp_pri` etc.

`val` **(required, numeric)** The value to store.

`ts` **(required, string)** The date-time of the data point as a string on the format RFC3339 (`YYYY-MM-DDThh:mm:ss±hh:mm`). Without timezone information, the API will fall back to the time zone configured for the domain.

Request example (cURL)

Request

```
# POST /domain/api/v1/timeseries
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-d 'node_id=1&tag=outdoortemp&val=13.4&ts=2019-10-01T11:30:22%2B02:00' \
'https://customer.noda.se/mydomain/api/v1/timeseries'
```

Response

```
[
  {
    "node_id": 1,
    "node_uuid": "2e949492-9008-4f81-a360-9059a769b195",
    "tag": "outdoortemp",
    "data": [{
      "v": 13.4,
      "ts": "2019-10-01T11:30:22+02"
    }]
  }
]
```

Store multiple datapoints

There are two different formats for storing multiple datapoints in one request.

One is using `Content-Type: application/x-www-form-urlencoded` and the other `Content-Type: application/json`.

Perform a `POST` request with `Content-Type: application/x-www-form-urlencoded` and the following attributes to the endpoint `/*domain*/api/v1/timeseries`.

`timeseries` **(required, string (JSON))** An array of timeseries objects.

```
[
    {
        "node_id": 1,
        "node_uuid": "2e949492-9008-4f81-a360-9059a769b195",
        "tag": "outdoortemp",
        "data": [{
            "v": 2.6,
            "ts": "2020-01-01T00:00:00"
        },{
            "v": 2.6,
            "ts": "2020-01-01T00:15:00"
        }]
    }
]
```

`overwrite` **(optional, string)** Possible values are: `replace_window` : This first deletes all datapoints between the `lowest` and `highest` ts *(a >= x AND a <= y)*, for each `node_id` and corresponding `tag`. Then inserts all the new datapoints.

`silent` **(optional, boolean)** Possible values are `true` or `false`. When set to `true`, a call will only reply with status code `201 Created` and an empty reply instead of `200 Success` and the inserted rows.

OR

Perform a `POST` request with `Content-Type: application/json` and the JSON data as the body content. Parameters are sent as query string arguments.

```
[
    {
        "node_id": 1,
        "node_uuid": "2e949492-9008-4f81-a360-9059a769b195",
        "tag": "outdoortemp",
        "data": [{
            "v": 2.6,
            "ts": "2020-01-01T00:00:00"
        },{
            "v": 2.6,
            "ts": "2020-01-01T00:15:00"
        }]
```

```
    }
]
```

`overwrite` **(optional, string)** Possible values are: `replace_window` : This first deletes all datapoints between the `lowest` and `highest` ts *(a >= x AND a <= y)*, for each `node_id` and corresponding `tag`. Then inserts all the new datapoints.

`silent` **(optional, boolean)** Possible values are `true` or `false`. When set to `true`, a call will only reply with status code `201 Created` and an empty reply instead of `200 Success` and the inserted rows.

Request example (cURL)

Request

```
# POST /domain/api/v1/timeseries
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-d
↪  'timeseries=[{"node_id":1,"tag":"outdoortemp","data":[{"v":2.6,"ts":"2020-01-01T00:00:00"},{"v":1
↪  \
'https://customer.noda.se/mydomain/api/v1/timeseries'
```

Response

```
{
    "timeseries": [
        {
            "node_id": 1,
            "node_uuid": "2e949492-9008-4f81-a360-9059a769b195",
            "tag": "outdoortemp",
            "data": [{
                    "v": 2.6,
                    "ts": "2020-01-01T00:00:00"
                },{
                    "v": 2.6,
                    "ts": "2020-01-01T00:15:00"
                }]
            }
    ]
}
```

Network manager

A network manager is a controller for several vDERs. Currently, the only exposed interface via API is access to `open scenarios`.

The `{id}` part of the URL refers to the network manager id—a unique id for each network manager in a domain. The default id (first) is: `0`.

> **Endpoints GET** */domain*/api/v1/network_manager/{id}/scenarios
> **PUT** */domain*/api/v1/network_manager/{id}/scenarios

Toggle a single open scenario

Perform a `PUT` request with `Content-Type: application/x-www-form-urlencoded` and the following attributes to the endpoint `/*domain*/api/v1/network_manager/{id}/scenarios`.

URL attributes

`id` Supplied as part of the URL. The ID of the target network manager.

Body attributes

The following attributes are supplied in the body.

`scenario_id` **(required, integer)** Target scenario id, in the interval 1..X. Where *X* is the highest numbered scenario.

`state` **(required, boolean)** State; either `true` or `false`.

`ts` **(optional, string)** The date-time of the data point as a string on the format `YYYY-MM-DDThh:mm:ss±hh:mm`. Without timezone information, the API will fall back to the time zone configured for the domain. Defaults to **now**.

Status codes

`400` Invalid input. Ensure that the supplied parameters are correctly formatted.

`500` Internal server error. An invalid state occurred.

`200` OK. With the response content.

Request example (cURL)

Request

```
# PUT /domain/api/v1/network_manager/{id}/scenarios
# Using cURL CLI
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
-X PUT \
-d 'scenario_id=15&state=false' \
'https://customer.noda.se/mydomain/api/v1/network_manager/0/scenarios'
```

Response

```
{
    "scenario_id": 1,
    "state": false,
    "ts": 1578653498.03568
}
```

Data sets

Get and set objects (files) stored in these formats;

- CSV (text/csv)
- INI (text/plan; charset=utf-8)
- JSON (application/json)
- TOML (application/toml)
- XML (application/xml)
- YAML (application/yaml)
- Binary (application/octet-stream)

> **Endpoints POST** */domain*/api/v1/dataset/
> **GET** */domain*/api/v1/dataset/
> **GET** */domain*/api/v1/dataset/{id}
> **GET** */domain*/api/v1/dataset/{id}/raw
> **PUT** */domain*/api/v1/dataset/{id}
> **DELETE** */domain*/api/v1/dataset/{id}

Create a new data set

Perform a `POST` request with `Content-Type: application/json` and the following JSON structure to the endpoint `/*domain*/api/v1/dataset`.

```
{
    "content": "aGVsbG8sIHdvcmxkIQ==",
    "format": "ini",
    "name": "foo",
    "tags": [
        "configuration",
        "external-service"
    ],
    "thing_uuid": "54295623-c5e3-4085-80cf-542338b1bc30"
}
```

`content` The base64 encoded content of the object/file.
`format` One of csv, ini, json, misc, toml, xml, yaml.
`name` Name of the data set object.
`tags` Optional element used for filtering and identification. A list/array of strings.
`thing_uuid` Optional element used to bind a data set to a Node/Thing. Uses the globally unique identifier (UUID) of a Node/Thing.

Status codes

`400` Invalid input. Ensure that the supplied content is correctly formatted and that you are not assigning invalid objects.
`500` Internal server error. An invalid state occurred.
`201` The resource was created.

Request example (cURL)

Request

```
curl -s \
-d
↪  '{"content":"aGVsbG8sIHdvcmxkIQ==","format":"ini","name":"foo","tags":["configuration","external
↪  \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset'
```

Response


```
{
    "uuid": "8258755c-f817-4e87-a8fc-94cae7126fbe",
    "name": "foo",
    "format": "ini",
    "checksum":"68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728",
    "size": 13,
    "thing_uuid": "54295623-c5e3-4085-80cf-542338b1bc30",
    "created":"2022-01-12 10:25:17.065338+01",
    "updated":"2022-01-12 10:25:17.065338+01",
    "created_by": 1,
    "updated_by": 1,
    "tags": [
        "configuration",
        "external-service"
    ]
}
```


`uuid` This is the unique identifier for this particular data set.
`name` The name (human-readable) of the data set.
`format` The format of the stored content.
`checksum` The SHA256 checksum of the content.
`size` The size of the (decoded) content in bytes.
`thing_uuid` The optional reference to a Node/Thing. If not used, this value will be `null`.
`created` The timestamp (RFC 3339, section 5.6) when someone created this object.
`updated` The timestamp (RFC 3339, section 5.6) when someone updated this object last.
`created_by` A reference to the account ID (user) who created the data set.
`updated_by` A reference to the account ID (user) who performed the last data set update.
`tags` An optional list of tags used for filtering and identification. If not used, the result is an empty list `[]`.

List data sets

Perform a `GET` request with to the endpoint `/*domain*/api/v1/dataset` with query arguments;

`offset` An integer with the lowest value of 0. Used to skew (offset) the result range.
`limit` An integer with the lowest value of 1 and the highest value of 100. Used to limit the amount of results returned for each “page”.

Status codes

400 Invalid input. Ensure that the supplied parameters are correctly formatted.

500 Internal server error. An invalid state occurred.

200 OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset?offset=0&limit=10'
```

Response


```
[
  {
    "uuid": "11eff124-fdd1-4b0e-9d1a-52b9fe8497cb",
    "name": "Dataset #1",
    "format": "yaml",
    "size": 41,
    "checksum": "eb488679e0c0de6ac2e8446be252767b18fedea0c0a404b9bf12a530ca79c199",
    "thing_uuid": null,
    "created": "2021-10-01 12:26:26.42555+02",
    "created_by": 1,
    "updated": "2021-10-04 09:30:28+02",
    "updated_by": 1,
    "tags": []
  },
  {
    "uuid": "1271f951-29ff-4f2e-acb9-1410fbfe1c21",
    "name": "Dataset #2",
    "format": "toml",
    "size": 40,
    "checksum": "7df1475dd2847c094003f4d1a56bc420781b74543a0b5ec8205f16ff07e9e1c3",
    "thing_uuid": null,
    "created": "2021-10-01 12:25:15.246015+02",
    "created_by": 1,
    "updated": "2021-10-04 08:48:24+02",
    "updated_by": 1,
    "tags": []
```

```
  }
]
```

The response (JSON) is a list of objects;

`uuid` This is the unique identifier for this particular data set.
`name` The name (human-readable) of the data set.
`format` The format of the stored content.
`checksum` The SHA256 checksum of the content.
`size` The size of the (decoded) content in bytes.
`thing_uuid` The optional reference to a Node/Thing. If not used, this value will be `null`.
`created` The timestamp (RFC 3339, section 5.6) when someone created this object.
`updated` The timestamp (RFC 3339, section 5.6) when this object was last updated.
`created_by` A reference to the account ID (user) who created the data set.
`updated_by` A reference to the account ID (user) who performed the last data set update.
`tags` An optional list of tags used for filtering and identification. If not used, the result is an empty list `[]`.

Get details about a data set

Perform a `GET` request with to the endpoint `/*domain*/api/v1/dataset/{id}`;

`id` The UUID of the data set.

Status codes

`500` Internal server error. An invalid state occurred.
`200` OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset/11eff124-fdd1-4b0e-9d1a-52b9fe8497cb'
```

Response

```
{
    "uuid": "11eff124-fdd1-4b0e-9d1a-52b9fe8497cb",
    "name": "Dataset #1",
    "format": "yaml",
    "size": 41,
    "checksum": "eb488679e0c0de6ac2e8446be252767b18fedea0c0a404b9bf12a530ca79c199",
    "thing_uuid": null,
    "created": "2021-10-01 12:26:26.42555+02",
    "created_by": 1,
    "updated": "2021-10-04 09:30:28+02",
    "updated_by": 1,
    "tags": []
}
```


A (JSON) object with the following elements;

`uuid` This is the unique identifier for this particular data set.
`name` The name (human-readable) of the data set.
`format` The format of the stored content.
`checksum` The SHA256 checksum of the content.
`size` The size of the (decoded) content in bytes.
`thing_uuid` The optional reference to a Node/Thing. If not used, this value will be `null`.
`created` The timestamp (RFC 3339, section 5.6) when someone created this object.
`updated` The timestamp (RFC 3339, section 5.6) when this object was last updated.
`created_by` A reference to the account ID (user) who created the data set.
`updated_by` A reference to the account ID (user) who performed the last data set update.
`tags` An optional list of tags used for filtering and identification. If not used, the result is an empty list `[]`.

Get the content in a data set

Perform a `GET` request with to the endpoint `/*domain*/api/v1/dataset/{id}/raw`;

`id` The UUID of the data set.

Status codes

500 Internal server error. An invalid state occurred.
200 OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-H 'Accept: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset/11eff124-fdd1-4b0e-9d1a-52b9fe8497cb/raw'
```

Response

The `Content-Type` header will be one of;

- `text/csv`: csv
- `text/plan; charset=utf-8`: ini
- `application/json`: json
- `application/toml`: toml
- `application/xml`: xml
- `application/yaml`: yaml
- `application/octet-stream`: misc

Depending on the `format` of the data set.

```
hello: world
list:
 - item
 - item10
```

Update a data set

Perform a `PUT` request with `Content-Type: application/json` and the following JSON structure to the endpoint `/*domain*/api/v1/dataset/{id}`;

`id` The UUID of the data set.

```
{
    "content": "QXl5IGNhcHRpJ24K",
    "format": "ini",
    "name": "bar",
    "tags": [
        "privateer"
    ],
    "thing_uuid": null
}
```

`content` (Optional) the base64 encoded content of the object/file.
`format` (Optional) one of csv, ini, json, misc, toml, xml, yaml.
`name` (Optional) name of the data set object.
`tags` (Optional) element used for filtering and identification. A list/array of strings.
`thing_uuid` (Optional) element used to bind a data set to a Node/Thing. Uses the globally unique identifier (UUID) of a Node/Thing.

Status codes

400 Invalid input. Ensure that the supplied parameters are correctly formatted.
500 Internal server error. An invalid state occurred.
200 OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-X 'PUT' \
-d '{"content":"TGV0J20gd2FsayB5ZXIgcGxhbmsK"}' \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset/11eff124-fdd1-4b0e-9d1a-52b9fe8497cb'
```

Response

The status code will be 204 with an empty reply body on success.

Delete a data set

Perform a DELETE request to the endpoint `/*domain*/api/v1/dataset/{id}`.

Status codes

500 Internal server error. An invalid state occurred.
200 OK. With the response content.

Request example (cURL)

Request

```
curl -s \
-X 'DELETE' \
-H 'Content-Type: application/json' \
-H 'Authorization: Key 0123456789abcdef0123456789abcdef' \
'https://customer.noda.se/mydomain/api/v1/dataset/11eff124-fdd1-4b0e-9d1a-52b9fe8497cb'
```

Response

The status code will be 204 with an empty reply body on success.