
NODA EnergyView API v1

API documentation

NODA Intelligent Systems AB

2021-01-01

Contents

API Reference	4
API notes	4
HTTP headers	4
HTTP methods	4
Authentication	5
Errors	5
HTTP Status Code Summary	6
Error Types	7
Handling errors	7
Glossary	7
Domains	8
Allowed Request Rates	8
API limits	8
Client Example	9
API requests	9
CSV import	10
List CSV import integrations	10
URI parameters	10
Response example	10
Request example (cURL and Python)	10
POST <code>/domain/api/v1/csvimport/{uuid}</code>	11
URI parameters	11
Response example	11
Code example	11
Status codes	12
Nodes/Collectors	12
List all Nodes	12
Request example (cURL)	12

Sensors/Tags	14
List all sensor declarations	14
Request example (cURL)	14
Settings (Metadata)	15
Retrieve settings	15
URL parameters	15
Body parameters	16
Request example (cURL)	16
Store settings	17
URL parameters	17
Body parameters	17
Request example (cURL)	17
Time series	18
Example of a generalized reply object	18
Retrieve datapoints	19
Request example (cURL)	20
Store a single datapoint	21
Request example (cURL)	21
Store multiple datapoints	21
Request example (cURL)	22
Network manager	23
Toggle a single open scenario	23
URL attributes	23
Body attributes	24
Status codes	24
Request example (cURL)	24
Data sets	24
Create a new data set	25
Status codes	26
Request example (cURL)	26
List data sets	27
Status codes	27
Request example (cURL)	27
Get details about a data set	29
Status codes	29

Request example (cURL)	29
Get the content of a data set	30
Status codes	30
Request example (cURL)	30
Update a data set	31
Status codes	31
Request example (cURL)	32
Delete a data set	32
Status codes	32
Request example (cURL)	32

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

When performing a request two HTTP headers are required, `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.

All verbs (`PUT`, `POST`, `DELETE`) with the exception of `GET`, uses

`Content-type: application/x-www-form-urlencoded` and the `query string` is sent in the request body. With `GET` this is not allowed 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 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. In these cases, it is clearly stated in the section for that request.

Authentication

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

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 is: 0123456789abcdef0123456789abcdef
```

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	The type of error returned. One of api_connection_error, api_error,
string	authentication_error, invalid_request_error or rate_limit_error
<i>optional</i>	

Attributes

code string <i>always</i>	For some errors that could be handled programmatically, a short string indicating the error code reported.
doc_url string <i>optional</i>	A URL to more information about the error code reported.
error string <i>always</i>	A human-readable message providing more details about the error. Sometimes, these messages can be shown to your users.
param string <i>optional</i>	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 raise exceptions for many reasons, invalid parameters, authentication errors, and network unavailability. 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 as a way 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 which corresponds 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 Virtual Dynamic Energy Resource.

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 domain without access to all of the target domains. A domain is usually shown 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.

In writing, however, 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.

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 request, as these two kinds of requests affect the rate of requests in different ways.

- GET requests fetch data from the system.
- SET requests stores/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). As there can only be one process at a time accessing the underlying database table.

Therefore, doing several hundred separate store procedures at the same time (and to the same location) will not be more efficient but less efficient. As each process will stall while waiting for another process do 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 it needs.

Note: Request limits are per domain. Thus, multiple requests may be sent to different domains without causing the rate-limit to trigger.

However, the recommendation is that requests remain as few and as 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. 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**.

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

Note: a single data-point insert takes on average between 5ms and 15ms. A set bottleneck which 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 **GET** */domain/api/v1/nodes*

List all Nodes

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

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-db2d5baf27b",
      "name": "Exempel heatingsystem",
      "description": "",
      "public": true,
      "owner": false,
      "enabled": true,
      "archived": false,
```

```
    "representation": "circuit/heat",
    "device": {
      "id": 1,
      "name": "Kelp-Basic",
      "protocol_id": 1
    },
    "sensor_ids": [
      398,
      1,
      42,
      9,
      3,
      121
    ],
    "interval": 600
  }
]
```

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 only unique on a single domain.

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

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

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 component to determine if the Node should be used.

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

representation (required, string) Machine centric purpose declaration of the Node. Used by different component of the system.

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

sensor_ids (required, array of integers) A list of sensor id attached to this Node.

interval A period in seconds of how often data is expected to be stored on the Node. Used by the alert system to determine faults.

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 an JSON objects 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. In a domain with thousands of Nodes there may only still be less than a hundred sensors.

name (required, string) Declared name of the sensor, used throughout the system. Only 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}/`

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 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,
      "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*.

tag (required, string) Name of the sensor for which the data belongs.

data (required, array) A list (array) of data points as 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 parameter 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 parameter 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, which is 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=2020-01-01T00:00:00&end=2020-01-01T01:00:00'
```

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 datapoint

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, integer) Domain-unique id for the node.

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 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.

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,
  "tag": "outdoortemp",
  "value": 13.4,
  "ts": "2019-10-01T11:30:22+02"
}
```

Store multiple datapoints

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

NOTE: The Content-Type is still application/x-www-form-urlencoded even though we supply a JSON object. This is because we are supplying the JSON encoded data as a parameter.

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

```
[
  {
    "node_id": 1,
    "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 ($a \geq x$ AND $a \leq y$), 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":2.6,"ts":"2020-01-01T00:15:00"}]}' \
'https://customer.noda.se/mydomain/api/v1/timeseries'
```

Response

```
{
  "timeseries": [
    {
      "node_id": 1,
      "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. At the moment 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. 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 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/plain; 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 trying to assing invalid objects.
- 500** Internal server error. An invalid state occured.
- 201** The resource was created.

Request example (cURL)

Request

```
curl -s \
-d '{"con-
  ↪ tent": "aGVsbG8sIHdvcmxkIQ==", "format": "ini", "name": "foo", "tags": ["configuration", "ex
  ↪ service"], "thing_uuid": "54295623-c5e3-4085-80cf-542338b1bc30"}'
  ↪ \
-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",
  "check-
    ↪ sum": "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 this object was created.

updated The timestamp (RFC 3339, section 5.6) when this object was last update.

created_by A reference to the account ID (user) who created it the data set.

updated_by A reference to thge account ID (user) who performed the last update of the data set.

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 this object was created.

updated The timestamp (RFC 3339, section 5.6) when this object was last update.

created_by A reference to the account ID (user) who created it the data set.

updated_by A reference to thge account ID (user) who performed the last update of the data set.

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 this object was created.

updated The timestamp (RFC 3339, section 5.6) when this object was last update.

created_by A reference to the account ID (user) who created it the data set.

updated_by A reference to thge account ID (user) who performed the last update of the data set.

tags An optional list of tags used for filtering and identification. If not used the result is an empty list [].

Get the content of 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/plain; 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

On success the status code will be 204 with an empty reply body.

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

On success the status code will be 204 with an empty reply body.