2023-10-20

Greendata Data API

MANUAL

API VERSION: 1.0

GREENDATA | Netherlands

Contents

About the API

This Data API is primarily a RESTful HTTP JSON API.

It can be used to send and retrieve data related to meter readings.

As updates and newer versions are released, backwards compatibility within the same base version (1.X / 2.X) will be guaranteed. In other words: what works in version 1.0, will work in version 1.9. Thereby the API version is referenced only by its base version (1.2 = v1, 1.8 = v1).

API location: https://api.greendata.nl/data/v1/

Authentication

Authentication Key

Every request requires the authentication key to be supplied.

The authentication key can be acquired by obtaining an account. Contact the system admin (info@energiemanageronline.nl) to request an account, supplying an organization name.

It is imperative to keep the authentication key private, since any third party can use it to fetch, create, change and delete all data authorized for the account / key.

Supplying the key with your request

To supply the authentication key with your request, use the HTTP authentication framework (https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication).

Supply the key, base64 encoded, as: “Authorization: Basic -your-base64-key-here-“

For platforms that require HTTP authentication to be submitted as username & password, the authentication key has a username:password format, and can be split up in both values accordingly.

Temporarily also supports supplying key as: ?authentication_key=… in the url.

Request rate-limiting

The API has rate-limiting (brute force protection and performance safeguarding) in the following ways:

1. A maximum of 3 requests with an invalid authentication key from any single IP within 1 minute.
   Overusing this limit will result in a 1 minute block with response: “too many attempts”.
2. A maximum of 3 requests with an invalid authentication key, within 1 minute.
   Overusing this limit will result in a 1 minute block with response: “too many attempts”.
3. A maximum of 1000 requests for any single authentication key, within 1 minutes.
   Overusing this limit will result in a 1 minute block with result: “too many attempts”.
4. A maximum of 10 requests with an invalid uid or code, within 1 minute.
   Overusing this limit will result in a 1 minute block with result: “too many attempts”.

General Data Model

account				
property	value type	example value	default value	explanation
name	string	"Oudman Engineering" / "Syplon"	"unnamed"	Platform Name / Client Name
authentication_key	string	"d49cea2a56d6235e280b0072"	none / ""	unique hash, required for authentication with every API request

site				
property	value type	example value	default value	explanation
name	string (50)	"Site Alpha"	"unnamed"	Location Name
description	string (255)	"Street 12, 3456AB, City"	none / ""	
code	string (50)	"site_01" / "alpha"	none / ""	a unique" identifier (*for this account)
uid	string (50)	"d49cea2a56d6235e280b0072"	autogenerated	unique hash, will be autogenerated on creation, if not supplied
information	string (65k)	"{'project_start': '2020-01-01', ...}"	none / ""	Any string containing extra information, can be a JSON dataset (stringified)

device				
property	value type	example value	default value	explanation
name	string (50)	"Device Alpha"	"unnamed"	Device Name / Location Name / Utilization Name
description	string (255)	"Model 5K912.01"	none / ""	Device Name / Location Name / Utilization Name
code	string (50)	"1000024" / "alpha"	none / ""	a unique" identifier (*for this site)
uid	string (50)	"d49cea2a56d6235e280b0072"	autogenerated	unique hash, will be autogenerated on creation, if not supplied
information	string (65k)	"{'model':'5K912.01','v':'1.02', ...}"	none / ""	Any string containing extra information, can be a JSON dataset (stringified)

meter				
property	value type	example value	default value	explanation
name	string (50)	"Electricity"	"unnamed"	Meter Name / Location Name / Utilization Name
description	string (255)	"measures Wh usage per hour"	none / ""	
code	string (50)	"1000024" / "elec_1" / "port_01"	none / ""	a unique" identifier (*for this device)
uid	string (50)	"d49cea2a56d6235e280b0072"	autogenerated	unique hash, will be autogenerated on creation, if not supplied
information	string (65k)	"measures Wh usage per hour"	none / ""	Any string containing extra information, can be a JSON dataset (stringified)
meterable	string (50)	"ELEC" / "WATER" / "CO2"	"ELEC"	see table of meterables for options
meter_type	string (50)	"CONTINUOUS" / "FREE"	"FREE"	water usage meter is typically "CONTINUOUS", thermometer is typically "FREE"
cumulative	boolean	false	true	true: values can be added together. false: values can be "averaged"
reading_type	string (50)	"MOMENT" / "PERIOD"	"PERIOD	"PERIOD" reading values have a (different) from and till time
unit_type	string (50)	"Wh" / "m3"	"Wh"	
reading_factor	number	1000 / 1 / 0.001	1	the 1:X / unit_type:reading.value factor for these readings, example: mW = W:1000

reading				
property	value type	example value	default value	explanation
from_time	integer	1648144800	none / NULL	timestamp in seconds (not used for MOMENT meters)
till_time	integer	1648148400	none / NULL	timestamp in seconds (considered to be "time" for MOMENT meters)
value	big integer	1753	none / NULL	no decimals, range: 9223372036854775808 / -9223372036854775807

Meter Model – Different types of meter

The combinations of meter settings

Not all combinations of meter.meter_type, meter.reading_type and meter.cumulative are valid:

- A CONTINUOUS meter can only have MOMENT readings.
- A meter with MOMENT readings cannot be “cumulative”.

As a result:

- Setting a meter.meter_type as CONTINUOUS will autoset “reading_type” as (fixed) MOMENT.
- Setting a meter.reading_type as MOMENT will autoset “cumulative” as (fixed) “false”.

This means only the following combinations are valid:

1. CONTINUOUS meter with non-cumulative MOMENT readings. Example: Clock.
2. FREE meter with non-cumulative MOMENT readings. Example: Thermometer.
3. FREE meter with non-cumulative PERIOD readings. CO2 meter (that gives average over X time)
4. FREE meter with cumulative PERIOD readings. Electricity usage meter.

The reading factor

The ability to set the reading factor is a result of the choice to only allow reading values as integer.

Ideally the reading factor is always 1, which means the reading values will be supplied in direct units of “unit_type”. For example; a meter with Wh unit_type will have a reading value of 1000 to indicate 1000 Wh, or 1 kWh.

When a Wh meter’s reading values are supplied in units of kWh, a reading factor of 1000 needs to be set to let the system know that values are in units 1000 times the meter’s unit_type (Wh).

When a Celsius thermometer supplies values with 2 decimals, a reading factor of 0.01 needs to be set to let the system know that values are in units of 1/100 of the meter’s unit_type (Celsius).

Some examples of potential meters

meter examples									
name	description	code	uid	meterable	meter_type	cumulative	reading_type	unit_type	reading_factor
Electricity Main	main Wh usage per hour	elec_main_usage	dd570d5bdd5	ELEC	FREE	true	PERIOD	Wh	1
Electricity Main	main kWh meter	elec_main	aa199cac2e3	ELEC	CONTINUOUS	false	MOMENT	Wh	1000
Electricity	main Wh meter	elec_sub	aa199cac2e3	ELEC	CONTINUOUS	false	MOMENT	Wh	1
Thermometer	Temperature in celcius	temperature_01	5ef85873abd	TEMPERATURE	FREE	false	MOMENT	C	0.01
Thermometer	Hour average temperature	temperature_02	9d418cf2f585	TEMPERATURE	FREE	false	PERIOD	C	0.01
Water	Water meter	meter_00001	9c49d50h0b8	WATER	CONTINUOUS	false	MOMENT	m3	0.001
Water	Water meter	meter_00001	9c49d50h0b8	WATER	CONTINUOUS	false	MOMENT	L	1
CO2 meter	Measures CO2 in PPM	1000098	ef692138b3e	CO2	FREE	false	PERIOD	PPM	1

Meterables

The meterables are model representations of “anything that is meterable”. By using the appropriate meterable, the system can autoset the appropriate meter settings, and knows how the meter corresponds to other meters.

For instance; a CONTINUOUS electricity meter can have it’s values converted to a FREE electricity meter, so that the meter’s hourly throughput can be shown. Similarly the system will know how to add together the readings for a “FREE electricity import high rate” and a “FREE electricity import low rate” to get the total “FREE electricity import” readings. As well as subtracting readings, when appropriate.

When the meterable for your meter exists, it is highly recommended you supply the appropriate meterable code when creating a new meter, and don’t overwrite any of the meter’s autoset settings. When a meterable does not yet exist, the OTHER meterable can be used, and the meter’s settings can be custom set.

Overwriting a FREE meterable’s default reading_type

By default, all FREE meterables are considered to have reading_type PERIOD. Because in most cases, even “moment” readings such as temperature, are ideally an average over the previous period (to cancel out extremes). In all those cases it is preferred to have the appropriate period times (from and till time) supplied with every reading. If however, your FREE meter’s values are explicitly for a split-second moment, and not an average over time, then it is advised to set the meter’s reading_type to MOMENT, even when the appropriate meterable has default PERIOD.

All current meterables

meterables					
code	name	unit_type	meter_type	cumulative	reading_type
OTHER	Other				
TEMP_CELS	Temperature Celsius	°C	FREE	false	PERIOD
HUMID	Humidity	%	FREE	false	PERIOD
CO2	Carbon Dioxide	PPM	FREE	false	PERIOD
ATMO_PRESS	Atmospheric Pressure	bar	FREE	false	PERIOD
IRR_HO_PLANE	Irradiation In Horizontal Plane	Wh/m2	FREE	true	PERIOD

meterables - electricity					
code	name	unit_type	meter_type	cumulative	reading_type
ELEC	Electricity	Wh	FREE	true	PERIOD
ELEC_CONS	Electricity Consumption	Wh	FREE	true	PERIOD
ELEC_PROD	Electricity Production	Wh	FREE	true	PERIOD
ELEC_CONS_HIGH	Electricity Consumption High Rate	Wh	FREE	true	PERIOD
ELEC_CONS_LOW	Electricity Consumption Low Rate	Wh	FREE	true	PERIOD
ELEC_PROD_PV	Electricity Production Photovoltaic	Wh	FREE	true	PERIOD
ELEC_IMPORT	Electricity Import	Wh	FREE	true	PERIOD
ELEC_EXPORT	Electricity Export	Wh	FREE	true	PERIOD
ELEC_IMPORT_HIGH	Electricity Import High Rate	Wh	FREE	true	PERIOD
ELEC_IMPORT_LOW	Electricity Import Low Rate	Wh	FREE	true	PERIOD
ELEC_EXPORT_HIGH	Electricity Export High Rate	Wh	FREE	true	PERIOD
ELEC_EXPORT_LOW	Electricity Export Low Rate	Wh	FREE	true	PERIOD
C_ELEC	Electricity	Wh	CONTINUOUS	false	MOMENT
C_ELEC_CONS	Electricity Consumption	Wh	CONTINUOUS	false	MOMENT
C_ELEC_PROD	Electricity Production	Wh	CONTINUOUS	false	MOMENT
C_ELEC_CONS_HIGH	Electricity Consumption High Rate	Wh	CONTINUOUS	false	MOMENT
C_ELEC_CONS_LOW	Electricity Consumption Low Rate	Wh	CONTINUOUS	false	MOMENT
C_ELEC_PROD_PV	Electricity Production Photovoltaic	Wh	CONTINUOUS	false	MOMENT
C_ELEC_IMPORT	Electricity Import	Wh	CONTINUOUS	false	MOMENT
C_ELEC_EXPORT	Electricity Export	Wh	CONTINUOUS	false	MOMENT
C_ELEC_IMPORT_HIGH	Electricity Import High Rate	Wh	CONTINUOUS	false	MOMENT
C_ELEC_IMPORT_LOW	Electricity Import Low Rate	Wh	CONTINUOUS	false	MOMENT
C_ELEC_EXPORT_HIGH	Electricity Export High Rate	Wh	CONTINUOUS	false	MOMENT
C_ELEC_EXPORT_LOW	Electricity Export Low Rate	Wh	CONTINUOUS	false	MOMENT

Site: fetch

Use: Retrieve (a list of) sites

URL: https://api.greendata.nl/data/v1/site/

HTTP Method: GET

Response: JSON dataset, array with sites (or empty array)

Response example (JSON):

```
[
        {
                'name': Site Alpha',
                'code': 'alpha',
                'uid': 'd49cea2a56d6'
        },
        {
                'name': 'Site Bravo',
                'code': 'bravo',
                'uid': '235e280b0072'
        }
]
```

Options

When using the base fetch url (without any additions), all sites for the account are returned.

By using these options (and supplying the relevant additions), a single or subset of sites can be requested.

Preferred single site option: RESTful

Uid in url: https://api.greendata.nl/data/v1/site/{uid}/

Easy-use single site options:

Uid as param: https://api.greendata.nl/data/v1/site/?uid={uid}

Code as param: https://api.greendata.nl/data/v1/site/?code={code}

Preferred multifunctional option: HTTP Body Data

Using the HTTP Body Data for the instructions allows fetching multiple specific sites at once. It also has the added benefit of using the same methodology as the “create” and “change” requests.

For HTTP Body Data (GET), use the base fetch url and supply JSON as example:

```
[
        {
                'uid': 'd49cea2a56d6'
        },
        {
                'uid': '235e280b0072'
        },
        {
                'code': 'delta'
        }
]
```

We’re fetching 3 sites, with uid “d49cea2a56d6” or uid “235e280b0072” or code “delta”.

Site: create

Use: Create new sites

URL: Same as fetch request

HTTP Method: POST

Response: Same as fetch request

Options

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

Supplying these fields as POST values will save them accordingly as a single site:

- name (Optional. Default: “unnamed”)
- description (Optional. Default: “”)
- code (Optional. Default: “”)
- uid (Optional. Default: autogenerated. Supplying this is highly discouraged)
- information (Optional. Default: “”)

Preferred option: HTTP Body Data

Using the HTTP Body Data for the values, allows creating multiple sites at once.

Supplying the values as JSON in HTTP Body Data (POST) will save them accordingly, example:

```
[
        {
                'name': 'Site Alpha',
                'code': 'alpha'
        }
]
```

We’re creating a site with name “Site Alpha” and code “alpha”.

Site: change

Use: Change (existing) sites

URL: Same as fetch request

HTTP Method: PUT or POST

Response: Same as fetch request

Options

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

This option requires a single site fetch request url.

Supplying these fields as POST values will save them accordingly to the requested site:

- name (Optional. No change if not supplied)
- description (Optional. No change if not supplied)
- code (Optional. No change if not supplied)
- information (Optional. No change if not supplied)

Preferred option: HTTP Body Data

This option requires the base fetch url (without additions).

Using the HTTP Body Data for the values, allows changing multiple sites at once. It also allows both creating and updating multiple sites in a single request when using POST, since any supplied site with unknown code and uid, will be created, as per the "Create site request" instruction. Use PUT to avoid this and only change sites.

Supplying the values as JSON in HTTP Body Data (PUT or POST) will save them accordingly, example:

```
[
        {
                'name': 'Site Alpha',
                'uid': 'd49cea2a56d6'
        }
]
```

We're changing the name to Site "Alpha", for site with uid "d49cea2a56d6".

Site: delete

Use: Delete existing sites

URL: Same as fetch request

HTTP Method: DELETE

Response: Same as fetch request

Options

Any valid single or subset site fetch request with the DELETE HTTP method will delete the requested sites.

This API request is not yet supported.

Device: fetch

Use: Retrieve (a list of) devices

URL: https://api.greendata.nl/data/v1/device/

HTTP Method: GET

Response: JSON dataset, array with meters (or empty array)

Response example (JSON):

```
[
        {
                'name': 'Device Alpha',
                'code': 'alpha',
                'uid': 'd49cea2a56d6'
        },
        {
                'name': 'Device Bravo',
                'code': 'bravo',
                'uid': '235e280b0072'
        }
]
```

Alternative base fetch URL:

When using the base fetch url (without any additions), all devices for the account are returned.

To fetch devices for a site, use the RESTful site fetch request, adding the /device/ part:

https://api.greendata.nl/data/v1/site/{uid}/device/

Options

By using these options (and supplying the relevant additions), a single or subset of devices can be requested.

Preferred single device option: RESTful

Uid in url: https://api.greendata.nl/data/v1/device/{uid}/

Easy-use single device options:

Uid as param: https://api.greendata.nl/data/v1/device/?uid={uid}

Code as param: https://api.greendata.nl/data/v1/device/?code={code}

Preferred multifunctional option: HTTP Body Data

Using the HTTP Body Data for the instructions allows fetching multiple specific devices at once. It also has the added benefit of using the same methodology as the “create” and “change” requests.

For HTTP Body Data (GET), use the base fetch url and supply JSON as example:

```
[
        {
                'uid': 'd49cea2a56d6'
        },
        {
                'uid': '235e280b0072'
        },
        {
                'code': 'delta'
        }
]
```

We’re fetching 3 devices, with uid “d49cea2a56d6” or uid “235e280b0072” or code “delta”.

Device: create

Use: Create new devices

URL: Same as fetch request

HTTP Method: POST

Response: Same as fetch request

Options

When creating new devices without supplying the parent site (see alternative base fetch URL), a new site will be created as parent site for these new devices.

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

Supplying these fields as POST values will save them accordingly as a single device:

- name (Optional. Default: “unnamed”)
- description (Optional. Default: “”)
- code (Optional. Default: “”)
- uid (Optional. Default: autogenerated. Supplying this is highly discouraged)
- information (Optional. Default: “”)

Preferred option: HTTP Body Data

Using the HTTP Body Data for the values, allows creating multiple devices at once.

Supplying the values as JSON in HTTP Body Data (POST) will save them accordingly, example:

```
[
        {
                'name': 'Device Alpha',
                'code': 'alpha'
        }
]
```

We’re creating a device with name “Device Alpha” and code “alpha”.

Device: change

Use: Change (existing) devices

URL: Same as fetch request

HTTP Method: PUT or POST

Response: Same as fetch request

Options

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

This option requires a single device fetch request url.

Supplying these fields as POST values will save them accordingly to the requested device:

- name (Optional. No change if not supplied)
- description (Optional. No change if not supplied)
- code (Optional. No change if not supplied)
- information (Optional. No change if not supplied)

Preferred option: HTTP Body Data

This option requires the base fetch url.

Using the HTTP Body Data for the values, allows changing multiple devices at once. It also allows both creating and updating multiple devices in a single request when using POST, since any supplied device with unknown code and uid, will be created, as per the "Create device request" instruction. Use PUT to avoid this and only change devices.

Supplying the values as JSON in HTTP Body Data (PUT or POST) will save them accordingly, example:

```
[
        {
                'name': Device Alpha',
                'uid': 'd49cea2a56d6'
        }
]
```

We're changing the name to "Device Alpha", for device with uid "d49cea2a56d6".

Device: delete

Use: Delete existing devices

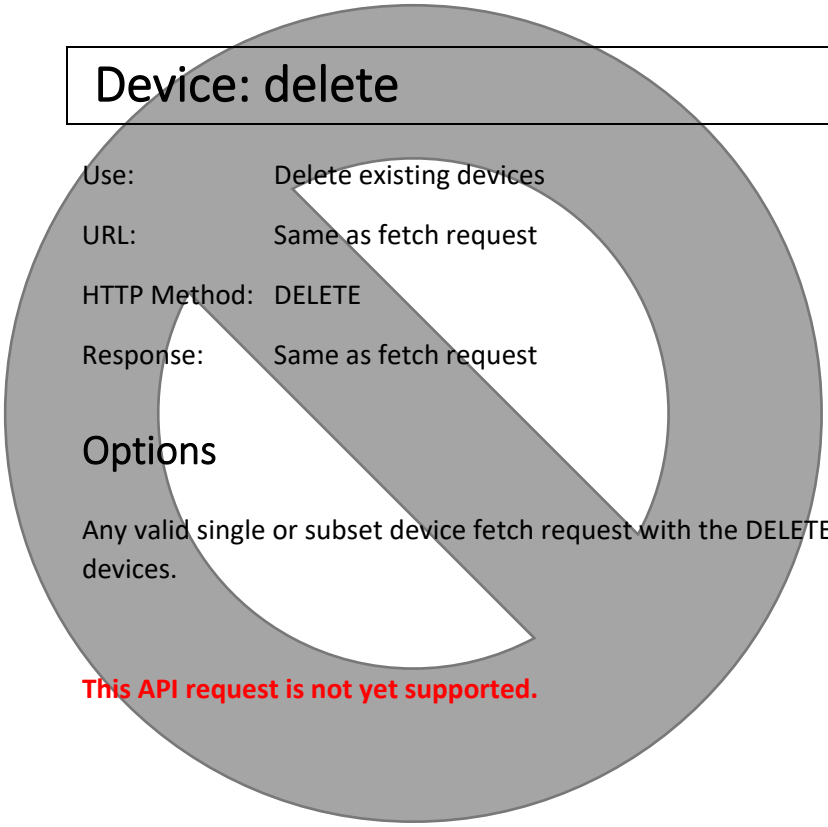
URL: Same as fetch request

HTTP Method: DELETE

Response: Same as fetch request

Options

Any valid single or subset device fetch request with the DELETE HTTP method will delete the requested devices.



This API request is not yet supported.

Meter: fetch

Use: Retrieve (a list of) meters

URL: https://api.greendata.nl/data/v1/meter/

HTTP Method: GET

Response: JSON dataset, array with meters (or empty array)

Response example (JSON):

```
[
        {
                'name': 'Temperature',
                'code': 'port_01',
                'uid': 'd49cea2a56d6'
        },
        {
                'name': 'Humidity',
                'code': 'port_02',
                'uid': '235e280b0072'
        }
]
```

Alternative base fetch URL:

When using the base fetch url (without any additions), all meters for the account are returned.

To fetch the meters for a site, use the RESTful site fetch request, adding the /meter/ part:

https://api.greendata.nl/data/v1/site/{uid}/meter/

To fetch the meters for a device, use the RESTful device fetch request, adding the /meter/ part:

https://api.greendata.nl/data/v1/device/{uid}/meter/

Options

By using these options (and supplying the relevant additions), a single or subset of meters can be requested.

Preferred single meter option: RESTful

Uid in url: https://api.greendata.nl/data/v1/meter/{uid}/

Easy-use single meter options:

Uid as param: https://api.greendata.nl/data/v1/meter/?uid={uid}

Code as param: https://api.greendata.nl/data/v1/meter/?code={code}

Preferred multifunctional option: HTTP Body Data

Using the HTTP Body Data for the instructions allows fetching multiple specific meters at once. It also has the added benefit of using the same methodology as the “create” and “change” requests.

For HTTP Body Data (GET), use the base fetch url and supply JSON as example:

```
[
        {
                'uid': 'd49cea2a56d6'
        },
        {
                'uid': '235e280b0072'
        },
        {
                'code': 'port_03'
        }
]
```

We’re fetching 3 meters, with uid “d49cea2a56d6” or uid “235e280b0072” or code “port_03”.

Meter: create

Use: Create new meters

URL: Same as fetch request

HTTP Method: POST

Response: Same as fetch request

Options

When creating new meters without supplying the parent device (see alternative base fetch URL), a new device will be created as parent device for these new meters.

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

Supplying these fields as POST values will save them accordingly as a single device:

- name (Optional. Default: “unnamed”)
- description (Optional. Default: “”)
- code (Optional. Default: “”)
- uid (Optional. Default: autogenerated. Supplying this is highly discouraged)
- information (Optional. Default: “”)
- meterable (Optional. Default: “ELEC”. If supplied; will autoset the meter settings)
- any other field as shown in the General Data Model for “meter”.

Preferred option: HTTP Body Data

Using the HTTP Body Data for the values, allows creating multiple meters at once.

Supplying the values as JSON in HTTP Body Data (POST) will save them accordingly, example:

```
[
        {
                'name': 'CO2',
                'code': 'port_03',
                'meterable': 'CO2'
        }
]
```

We’re creating a meter with name “CO2” and code “port_03” and meterable “CO2”.

Meter: change

Use: Change (existing) meters

URL: Same as fetch request

HTTP Method: PUT or POST

Response: Same as fetch request

Options

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

This option requires a single meter fetch request url.

Supplying these fields as POST values will save them accordingly to the requested meter:

- name (Optional. No change if not supplied)
- description (Optional. No change if not supplied)
- code (Optional. No change if not supplied)
- information (Optional. No change if not supplied)
- any other field as shown in the General Data Model for “meter”, except uid.

Preferred option: HTTP Body Data

This option requires the base fetch url.

Using the HTTP Body Data for the values, allows changing multiple meters at once. It also allows both creating and updating multiple meters in a single request when using POST, since any supplied meter with unknown code and uid, will be created, as per the “Create meter request” instruction. Use PUT to avoid this and only change meters.

Supplying the values as JSON in HTTP Body Data (PUT or POST) will save them accordingly, example:

```
[
        {
                'description': 'Measures average temperature per hour in degrees Celsius',
                'code': 'port_01'
        }
]
```

We’re changing the description for meter with code “port_01”.

Meter: delete

Use: Delete existing meters

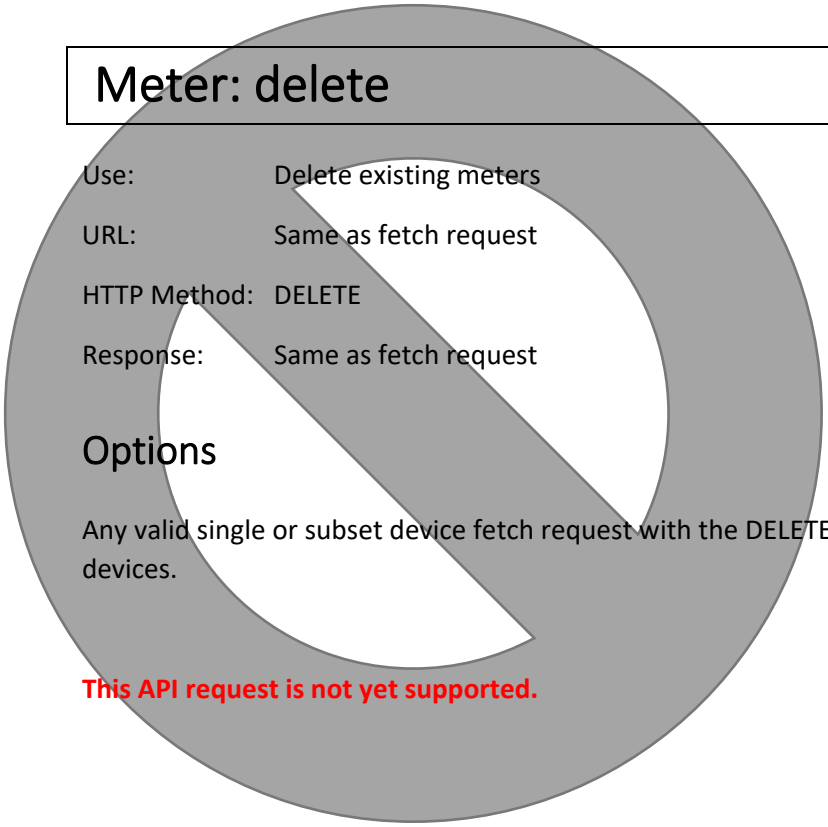
URL: Same as fetch request

HTTP Method: DELETE

Response: Same as fetch request

Options

Any valid single or subset device fetch request with the DELETE HTTP method will delete the requested devices.



This API request is not yet supported.

Reading: fetch

Use: Retrieve (a list of) readings

URL: https://api.greendata.nl/data/v1/reading/

HTTP Method: GET

Response: JSON dataset, array with readings (or empty array)

Response example (JSON):

```
[
        {
                'from_time': '1648144800',
                'till_time': '1648148400',
                'value': '1753'
        },
        {
                'from_time': '1648148400',
                'till_time': '1648152000',
                'value': '1484'
        },
]
```

Alternative base fetch URL:

When using the base fetch url (without any additions), all readings for the account are returned.

To fetch the readings for a site, use the RESTful site fetch request, adding the /reading/ part:

https://api.greendata.nl/data/v1/site/{uid}/reading/

To fetch the readings for a device, use the RESTful device fetch request, adding the /reading/ part:

https://api.greendata.nl/data/v1/device/{uid}/reading /

To fetch the readings for a meter, use the RESTful meter fetch request, adding the /reading/ part:

https://api.greendata.nl/data/v1/meter/{uid}/reading /

Period vs Moment readings

Depending on the meter setting “reading_type”, readings will have either a “time” field (for meters with reading type MOMENT) or a “from_time” and “till_time” field (for meters with reading type PERIOD).

The “time” field is in truth saved as, and thus similar, to the “till_time” and can be used as such.

Options

For the purpose of working with readings, the time (or till_time) timestamp is used as id, and only works when supplying the parent meter.

By using these options (and supplying the relevant additions), a single or subset of readings can be requested.

Preferred single reading option: RESTful

Time in url: https://api.greendata.nl/data/v1/meter/{uid}/reading/{time}/

Easy-use single meter options:

Time as param:

https://api.greendata.nl/data/v1/meter/{uid}/reading/?time={time}

Preferred multifunctional option: HTTP Body Data

Using the HTTP Body Data for the instructions allows fetching multiple specific readings at once. It also has the added benefit of using the same methodology as the “create” and “change” requests.

For HTTP Body Data (GET), use the base fetch url and supply JSON as example:

```
[
        {
                'time': '1648148400'
        },
        {
                'time': '1648152000'
        }
]
```

*We’re fetching 2 readings, with time “*1648148400*” or time “*1648152000*”.*

Limits

The maximum number of readings that can be fetched in one request is capped at 10.000. The only way to fetch more readings is to use batches (of max 10.000 readings each). This can be done by using the extra filters.

Extra filters

- Supply the “batch” GET parameter to iterate over batches. The first batch (batch 1) is the same batch as when no “batch” parameter is supplied. Use batch=2, batch=3, etc.
- Supply the “batch_from_time” and/or the “batch_till_time” to only receive readings with the appropriate “time” or “till_time”.

Reading: create

Use: Create new readings

URL: Same as fetch request

HTTP Method: POST

Response: Same as fetch request

Options

Creating new readings requires the parent meter. The parent meter can be supplied by using the appropriate alternative base fetch URL, or by supplying the required identification, per reading, in the HTTP Body Data (see below).

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

Supplying these fields as POST values will save them accordingly as a single device:

- from_time (Unix timestamp. Required for PERIOD, ignored for MOMENT)
- till_time (Unix timestamp. Required for PERIOD)
- time (Unix timestamp. Required for MOMENT. Can be NULL to indicate “autoset”)
- value (Integer. Required. Can be NULL / empty to indicate no reading for time)

Preferred option: HTTP Body Data

Using the HTTP Body Data for the values, allows creating multiple readings at once.

Supplying the values as JSON in HTTP Body Data (POST) will save them accordingly, example:

```
[
        {
                'from_time': '1648144800',
                'till_time': '1648148400',
                'value': '1753',
                'device.code': 'alpha',
                'meter.code': 'port_05',
        }
]
```

We’re creating a reading for a meter with code “port_05” for the device “alpha”.

When using a “meter.code” that is assumed to be unique, “device.code” is not required. Similarly using “meter.uid” will suffice. And alternatively, for an assumed unique “device.code”, the “device.uid” can be used.

Options for "time" value

By default, the time value is assumed to be the exact unix timestamp of the reading. However, since some reading sources might not support an internal clock, or might not have an accurate clock, there are some alternative options to supply the time.

Supplying the (optional) “call_time” GET parameter will allow you to save readings with a “time”, “from_time” and “till_time” value, other than the supplied reading time. Possible values are:

- CURRENT timestamp: Supplying the “call_time” with the (CURRENT) timestamp value of when the call request is made, allows the server to compare it to it’s own CURRENT timestamp. Any difference in time will then be used to offset the reading time values so that they will be in server clock time. This is useful when the source clock is not trusted to be accurate.
- 0: Supplying the “call_time” with a 0 value, allows you to supply each reading with a 0 time value so that they’ll receive the CURRENT server clock timestamp. Additionally supplying readings with for example a time value of -600 will have the server save the reading with a “600 seconds ago” server clock timestamp. This is useful when the source has no (timestamps) clock.

For request without the “call_time” parameter, reading times are saved “as is”. So make sure to implement it when using any of the above options.

Multi creation

This API allows the reading creation request to be used to auto create the corresponding meter, device and site. This is done by supplying the required properties in the request.

- When creating new readings for a meter that doesn’t exist yet, supplying an unique meter code and meterable will result in those readings saved under a newly created meter accordingly.
- Optional meter properties such as “name” or “description” will also be saved appropriately.
- When a “to be created” meter’s code is not unique, the device uid or code is required. Similarly as with the meter; supplying the device code (with optional parameters) will result in that device being created accordingly.
- When a “to be created” device’s code is not unique, the site uid or code is required. Similarly as with the device; supplying the site code (with optional parameters) will result in that site being created accordingly.
- When the device code is not unique and a site uid or code is not supplied, the device will be created under the default “anywhere” site.

Supplying these meter, device and site properties is done in the same way as shown in the “JSON HTTP Body Data” example above. So often the required properties (except the meter.meterable) are already supplied in the required way, and the objects will be created automatically.

Caveat: It is not possible to change an existing meter’s properties using this method.

Reading: change

Use: Change (existing) readings

URL: Same as fetch request

HTTP Method: PUT or POST

Response: Same as fetch request

Options

The API offers two different options of supplying the data to be saved:

Easy-use option: basic POST parameters

This option requires a single reading fetch request url.

Supplying these fields as POST values will save them accordingly to the requested meter:

- from_time (Unix timestamp. Required for PERIOD, ignored for MOMENT)
- till_time (Unix timestamp. Required for PERIOD)
- time (Unix timestamp. Required for MOMENT. Can be NULL to indicate “autoset”)
- value (Integer. Required. Can be NULL / empty to indicate no reading for time)

Preferred option: HTTP Body Data

This option requires the base fetch url and supplying the required identification, per reading, in the HTTP Body Data.

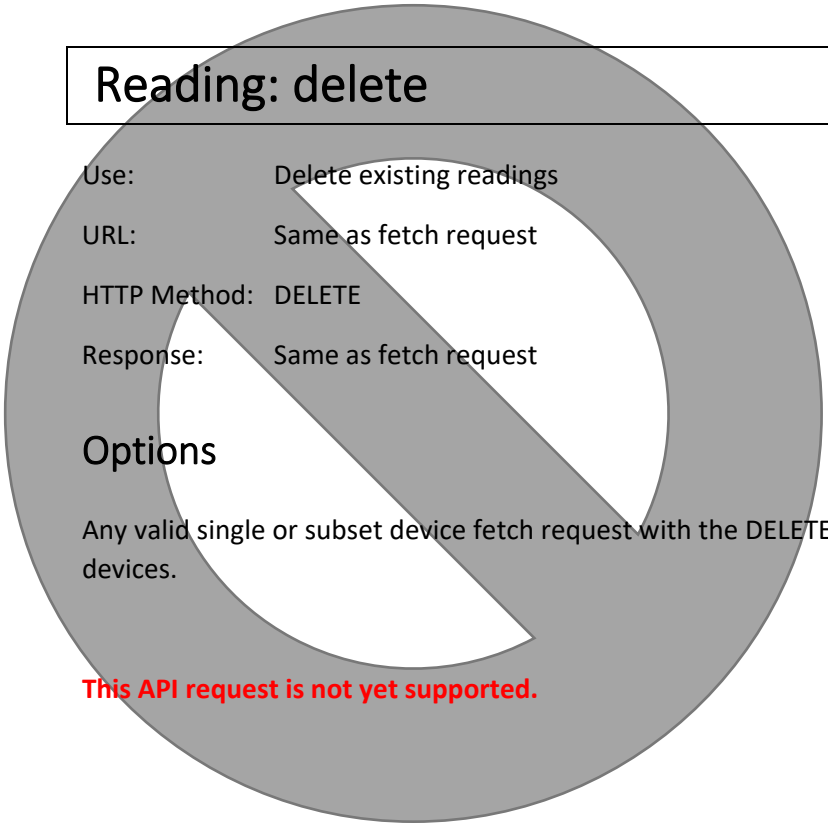
Using the HTTP Body Data for the values, allows changing multiple readings at once. It also allows both creating and updating multiple readings in a single request when using POST, since any supplied reading with a new time or till_time, will be created, as per the “Create reading request” instruction. Use PUT to avoid this and only change readings.

Supplying the values as JSON in HTTP Body Data (PUT or POST) will save them accordingly, example:

```
[
        {
                'time': '1648148400',
                'value': '1753',
                'meter.uid': 'd49cea2a56d6',
        }
]
```

*We’re saving a reading for meter with uid “*d49cea2a56d6*”. This can be a new reading, or an overwrite.*

Reading: delete



Use: Delete existing readings

URL: Same as fetch request

HTTP Method: DELETE

Response: Same as fetch request

Options

Any valid single or subset device fetch request with the DELETE HTTP method will delete the requested devices.

This API request is not yet supported.