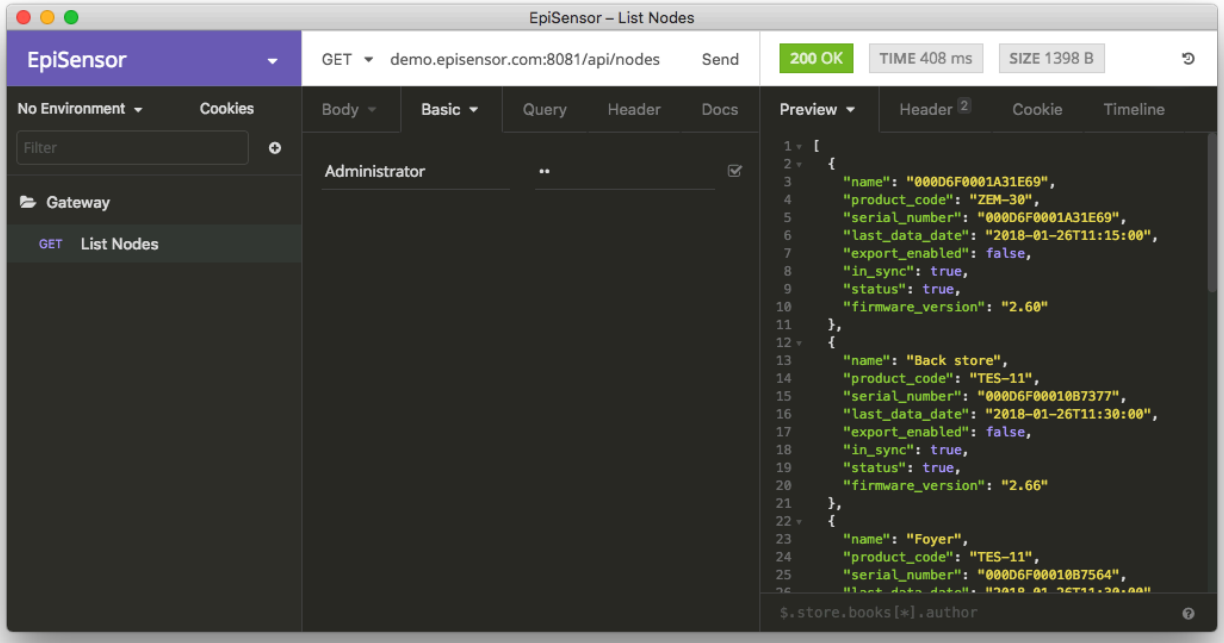


User Guide

EpiSensor Gateway API

Applies to: NGR-30-3, NGR-30-5, ZGW-10

EPI-009-10

© EpiSensor

Table of Contents

Introduction **3**

Getting Started **3**

- Basic Information 3
- Authentication 3
- Initialisation 4
- Versioning 4
- Timestamps 4
- Pagination 4
- Usage Restrictions 5
- HTTP Error Codes 5
- Content Type 5
- Error & Status Messages 5
- Management & Test Tools 6

API Resources **6**

- Gateway Status 7
- Gateway Commands 7
- All Nodes 8
- Interacting at the Node level 9
- Requesting all Sensors from one Node 12
- Requesting one Sensor from one Node 13
- Requesting Multiple Data Points 14
- Requesting Live Calibration Data 16
- Changing Node Properties 17
- Change a property of one Sensor 24
- Using Set Sensor Value 28
- Send a node level command 28

API Request Workflow **30**

Introduction

EpiSensor's Gateway API is designed around REST, and provides a way to interact with the Gateway and wireless nodes programmatically. This means that changes can be made in bulk and systems can be more easily monitored and maintained remotely.

The resources of the Gateway are arranged around easy-to-understand URLs that reflect the structure of the Gateway's web interface as closely as possible.

The Gateway uses standard HTTP features to make the API as easy to integrate with as possible, and all responses are returned in JSON format.

Getting Started

This section has information on how to authenticate with the API, how data will be returned, and information on tools and additional resources that may be helpful in working with the API. For additional information, please contact support@episensor.com

Basic Information

From version V03.00.00.00 of the Gateway software the API will be enabled by default. The Gateway's web server runs on TCP port 8081, and the API will run on the same port. However, this is configurable from the 'Engineer' settings page of the Gateway.

In a future software release, it will be possible to enable/disable the API from the Gateway's user interface, on the Settings -> API page. The API will run at an endpoint like this (assuming the web server port is set to the default 8081):

http://<gateway_ip>:8081/api/

Authentication

The API will support HTTP basic authentication in version 1, using the credentials of the "Administrator" user account on the Gateway Web Interface. The password for this user account can be set from the Settings -> Password page when logged in to the Gateway Web Interface.

Please note that if the password is changed from the Gateway Web Interface, it will only take effect in the API after a reboot of the Gateway.

Important Note	
	Authentication on the web interface of the Gateway will accept variations of the Administrator username like “Admin” and “admin”, but the API will only accept “Administrator”. On Gateway software version V04.01.00.00 and above, the API will also accept “Admin” and “admin”.

The code examples below include this basic authentication header with the default user account info encoded.

Initialisation

After startup or reboot of the EpiSensor Gateway, the first request made to the API will initialise the API servlet, and could take up to 60 seconds to complete. A workflow to initialise the API should be built into applications that depend on fast response times.

Versioning

There will be URL versioning support included with the initial release, so future versions of the the API can add new functionality, and an application matched to a particular release of the API. If no version number is provided in the URL, the latest version of the API will be used.

Timestamps

The timestamp format will be the same as is used on EpiSensor’s existing JSON and CSV data exports, which is a subset of RFC 3339, for example:

2015-09-27T16:26:00

The timestamp doesn’t include time zone or locale information, but it will be possible to query this from the settings section of the API.

Pagination

If there is a large number of nodes or sensors on the Gateway, the API will paginate the results - but this won't be supported in the initial release. This is likely to work by passing the page number as a query parameter within an HTTP GET to an endpoint that has multiple pages, for example:

/api/v1/nodes?page=2

The total number of pages (and the sequence of the page that has been returned) would be included in the JSON response so the client can request each page in the set sequentially.

Usage Restrictions

Limitations on concurrent sessions and requests per hour will also be implemented in future versions of the API to keep the resource usage of the Gateway system within acceptable limits.

HTTP Error Codes

The table below lists all supported HTTP response codes, and a description of when each code will be returned. There may also be additional information returned in the body of the response.

Code	Description
200	Success
202	POST has been accepted for processing
304	Not modified. Used when the POST has not resulted in any changes.
404	Not found. Used when the specified resource is not found.
400	Bad Request. Used when the JSON body is malformed.
406	Not Acceptable. Used specifically when the node or the sensor id is not correctly specified.
500	Internal Server Error. Used when the gateway is unable to process the request.

Content Type

HTTP response headers will have a content type set to “application/json”.

Error & Status Messages

Apart from the HTTP status code, a human-readable message will be included in the body of the response to an HTTP POST in JSON format. This will provide useful information for error logging and user feedback.

```
{
  "status": 200,
```

```
  "reason": "OK",
  "message": "Node 000D6F00010B52B4 updated"
}
```

Management & Test Tools

We would like to encourage our partners to consider open-sourcing any management tools that are developed for interacting with the EpiSensor Gateway API, so others can benefit from the features developed. EpiSensor will host a code repository and issue tracker for these projects on GitHub.

For testing the API, two useful tools are POSTMAN which can be downloaded at the following link: https://www.postman.com and Insomnia, available at this link: https://insomnia.rest/

Please contact EpiSensor support for sample requests for the EpiSensor Gateway API.

API Resources

The structure of the API will be hierarchical and will reflect the Gateway's user interface as closely as possible, with the top level returning Gateway status information, similar to what you would see on the Gateway's home page.

/api/status
/api/command
/api/nodes
/api/nodes/<node_serial>
/api/nodes/<node_serial>/sensors
/api/nodes/<node_serial>/sensors/<sensor_id>
/api/nodes/<node_serial>/sensors/<sensor_id>/data
/api/nodes/<node_serial>/sensors/<sensor_id>/calibration/data
/api/nodes/<node_serial>/command/<command_type>

There will be three main API endpoints - "status", “command" and "status". It’s also possible to interact with individual nodes and sensors using the hierarchy above.

The **/api/nodes/<node_serial>/sensors/<sensor_id>/data** endpoint returns the recent data points saved on the Gateway for a particular sensor, which by default stores the last 96 values received for every sensor.

The **/api/nodes/<node_serial>/sensors/<sensor_id>/calibration/data** endpoint returns the calibration data points saved on the Gateway for a particular sensor.

Gateway Status

An example of the status information returned by accessing the /api/status endpoint is as follows:

HTTP GET

/api/status


```
{
        "serial_number": "000D6F000C5770EC",
        "name": "EpiSensor Gateway",
        "status": "OK",
        "software_version": "V04.01.00.03",
        "current_time": "2019-03-29T12:03:02",
        "time_zone": "UTC",
        "start_time": "2019-03-28T09:10:25",
        "up_time": " 1 day, 2 hours, 52 minutes and 36 seconds",
        "logging_level": "INFO",
        "start_up_progress": "Started",
        "zap_connection": "SOCKET",
        "number_of_nodes": 2,
        "number_of_active_nodes": 0,
        "number_of_reporting_sensors": 22,
        "number_of_exporting_sensors": 4,
        "allow_join_enabled": false,
        "export_type": "EpiSensor JSON stored locally",
        "last_export": "2019-03-29T12:00:37",
        "export_interval": 30,
        "network_connection": "Ethernet",
        "internal_ip": "10.10.11.64",
        "external_ip": "212.17.63.154"
}
```


The Gateway status information above is read-only, and approximately represents the information available on the ‘Home’ page of the Gateway’s web interface.

Gateway Commands

Commands may be sent to the Gateway using an HTTP POST to the /api/command endpoint:

HTTP POST

/api/command


```
{
        "command": “allow_join",
        "parameters": {
            "interval": 15
```

```
        }
}
```

The currently supported commands are :

command	Required parameters	Description
allow_join	The allow join interval (in seconds) as an integer, for example : `"interval": 15`	The Zigbee network will be opened for new nodes to join for the interval specified. An interval of 0 will result in the network closing. An interval of 65535 will result in the network remaining open permanently.
set_gateway_name	The new name of the Gateway as a string, for example : `"name": “TEST name”`	This will change the name of the Gateway displayed on the About > Overview page on the Gateway’s web interface and API.
set_gateway_description	The new name of the Gateway as a string, for example : `"description": “TEST description”`	This will change the description of the Gateway displayed on the About > Overview page on the Gateway’s web interface and API.
restart_hardware	None	The Gateway hardware will be restarted with immediate effect.

All Nodes

The ‘nodes’ endpoint will return a list of nodes and their status at the following endpoint:

HTTP GET

/api/v1/nodes

```
[
  {
    "name ": "000D6F0001A30FB6",
    "product_code": "HTS-10",
    "serial_number": "000D6F0001A30FB6",
    "last_data_date": "2015-12-17T13:00:00",
    "export_enabled": true,
    "in_sync": true,
    "status": true,
    "firmware_version": "3.02"
  },
```

```
  {
    "name ": "000D6F00030516C4",
    "product_code": "TES-32",
    "serial_number": "000D6F00030516C4",
    "export_enabled": true,
    "in_sync": true,
    "status": false,
    "firmware_version": "2.84"
  }
]
```


Interacting at the Node level

You can access more detailed information about a particular node by sending an authenticated HTTP GET request to, for example:

HTTP GET

/api/nodes/<node_serial>

Where <node_serial> is the serial number of that node, for example: /api/nodes/000D6F0001A30FB6

An example of the JSON response is as follows, and it will contain a list of sensor names and ID's and a list of the neighbour and children nodes of this node in the Zigbee network.


```
{
  "serial_number": "000D6F0001A30FB6",
  "name ": "000D6F0001A30FB6",
  "description": "",
  "status": true,
  "in_sync": true,
  "date_added": "2015-11-30T13:12:47",
  "last_communication_date": "2015-12-17T13:03:17",
  "last_data_date": "2015-12-17T13:00:00",
  "product_code": "HTS-10",
  "firmware_version": "3.02",
  "has_power_amp": false,
  "sensor_list": [
    {
      "name": "Temperature T1",
      "units": "C",
      "last_data_value": 20.1,
      "last_data_date": "2015-12-17T13:00:00",
      "id": 350,
      "in_sync": true,
      "export_enabled": true,
      "reporting_enabled": true
    },
    {
```

```
      "name": "Relative Humidity",
      "units": "%",
      "last_data_value": 68.3,
      "last_data_date": "2015-12-17T13:00:00",
      "id": 358,
      "in_sync": true,
      "export_enabled": true,
      "reporting_enabled": true
    },
    {
      "name": "Battery Level",
      "units": "mV",
      "last_data_value": 3604,
      "last_data_date": "2015-12-17T00:00:00",
      "id": 4096,
      "in_sync": true,
      "export_enabled": false,
      "reporting_enabled": true
    }
  ],
  "parent": {
      "name": "000D6F00010B768E",
      "serial_number": "000D6F00010B768E",
      "export_enabled": false,
      "in_sync": true,
      "status": true,
      "firmware_version": "2.84"
   },
  "neighbour_list": [
    {
      "serial_number": "000D6F00010B768E",
      "name": "000D6F00010B768E",
      "lqi": 255,
      "in_cost": 1,
      "out_cost": 1,
      "aging_periods": 2
    },
    {
      "serial_number": "000D6F00030516C4",
      "name": "Router",
      "lqi": 255,
      "in_cost": 2,
      "out_cost": 2,
      "aging_periods": 3
    }
 ],
 "child_list": [
    {
      "serial_number": "000D6F00010B768F",
      "name": "Temperature T2"
    },
    {
      "serial_number": "000D6F00030516C4",
```

```
      "name": "Temperature T3"
    }
 ]
}
```

In the EpiSensor network, Neighbour Nodes are powered devices that have the ability to route packets in the ZigBee network. Child Nodes are 'end devices' and are usually battery powered, so they become sleepy end devices. They do not route packets in the network and have no concept of neighbours themselves and cannot have child nodes. For each neighbour the following information is provided:

Option	Description
serial_number	The neighbour node serial number
name	The neighbour node name
lqi	LQI Incoming to the node from the neighbour.
in_cost	The inbound cost between the node and the neighbour. This value is computed from the average LQI. Values will be in the range [1, 7] with 7 being the worst link quality.
out_cost	The outbound cost between the node and the neighbour. This value is computed from the average LQI. Values will be in the range [0, 7] with 7 being the worst link quality. A value of 0 means no messages have been exchanged.
aging_periods	The number of aging periods which have elapsed since the last message exchange between the node and this neighbour. An aging period is 16 seconds. Any value greater than 3 (48 seconds) is stale.

For each child node, just the serial_number and name are provided. The additional information is not available for child nodes.

Code Example (curl):

```
curl -X GET \
  http://172.31.255.1:8081/api/nodes/000D6F000C8140EE \
  -H 'authorization: Basic QWRtaW5pc3RyYXRvcjpBMQ==' \
  -H 'cache-control: no-cache'
```

Code Example (Java / OK HTTP):

```
OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
  .url("http://172.31.255.1:8081/api/nodes/000D6F000C8140EE")
  .get()
  .addHeader("authorization", "Basic QWRtaW5pc3RyYXRvcjpBMQ==")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

Requesting all Sensors from one Node

More detailed information on all sensors of a node can be accessed using an endpoint which references the sensors, for example:

HTTP GET

/api/nodes/<node_serial>/sensors

Where <node_serial> is the serial number of the node, for example: /api/nodes/000D6F0001A30FB6/sensors

An example of the JSON response is as follows:


```
[
  {
    "name": "Temperature T1",
    "id": 350,
    "units": "C",
    "export_enabled": true,
    "export_identifier": "000D6F0001A30FB6_350",
    "in_sync": true,
    "last_data_date": "2015-12-17T13:15:00",
    "reporting_mode": "SNAP_TO_CLOCK",
    "reporting_interval": 15,
    "logging_mode": "ON",
    "reporting_delta": 0,
    "last_data_value": 20.1
  },
  {
    "name": "Relative Humidity",
    "id": 358,
    "units": "%",
    "export_enabled": true,
    "export_identifier": "000D6F0001A30FB6_358",
    "in_sync": true,
    "last_data_date": "2015-12-17T13:15:00",
```

```
    "reporting_mode": "SNAP_TO_CLOCK",
    "reporting_interval": 15,
    "logging_mode": "ON",
    "reporting_delta": 0,
    "last_data_value": 68.4
  },

  {
    "name": "Battery Level",
    "id": 4096,
    "units": "mV",
    "export_enabled": false,
    "export_identifier": "000D6F0001A30FB6_4096",
    "in_sync": true,
    "last_data_date": "2015-12-17T00:00:00",
    "reporting_mode": "SNAP_TO_CLOCK",
    "reporting_interval": 1440,
    "logging_mode": "ON",
    "reporting_delta": 0,
    "last_data_value": 3604
  }
]
```


Requesting one Sensor from one Node

More detailed information on an individual sensor would be accessed using an endpoint which references the sensor ID, for example:

HTTP GET

/api/nodes/<node_serial>/sensors/<sensor_ID>

Where <node_serial> is the serial number of the node, and <sensor_ID> is the ID of the sensor, for example: /api/nodes/000D6F0001A30FB6/sensors/350

An example of the JSON response is as follows:


```
{
  "name": "Temperature T1",
  "id": 350,
  "units": "C",
  "export_enabled": true,
  "export_identifier": "000D6F0001A30FB6_350",
  "in_sync": true,
  "last_data_date": "2015-12-17T13:00:00",
  "reporting_mode": "SNAP_TO_CLOCK",
  "reporting_interval": 15,
  "logging_mode": "ON",
```

```
  "reporting_delta": 0,
  "last_data_value": 20.1
}
```

Code Example (curl):

```
curl -X GET \
  http://172.31.255.1:8081/api/nodes/000D6F0001A30FB6/sensors/350 \
  -H 'authorization: Basic QWRtaW5pc3RyYXRvcjpBMQ==' \
  -H 'cache-control: no-cache'
```

Code Example (Java / OK HTTP):

```
OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
  .url("http://172.31.255.1:8081/api/nodes/000D6F0001A30FB6/sensors/350")
  .get()
  .addHeader("authorization", "Basic QWRtaW5pc3RyYXRvcjpBMQ==")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

Requesting Multiple Data Points

By default, the Gateway will store the last 96 data points received from every sensors in memory on the Gateway. To request multiple data points, use the following endpoint:

HTTP GET

/api/nodes/<node_serial>/sensors/<sensor_ID>/data

Where <node_serial> is the serial number of the node, and <sensor_ID> is the ID of the sensor, for example: /api/nodes/000D6F0001A30FB6/sensors/350/data.

An example of the JSON response is as follows, where there are three data points stored on the gateway for this sensor:

```
{
  "data": [
```

```
      {
        "period": "2017-08-30T13:13:00",
        "value": 25.36
      },
      {
        "period": "2017-08-30T13:14:00",
        "value": 25.34
      },
      {
        "period": "2017-08-30T13:15:00",
        "value": 25.24
      },
  ]
}
```


To request a specified number of data points from a sensor, use the following endpoint (available in Version V04.01.00.01 of the EpiSensor Gateway):

HTTP GET

/api/nodes/<node_serial>/sensors/<sensor_ID>/data/<numberOfDataPoints>

Where <node_serial> is the serial number of the node, <sensor_ID> is the ID of the sensor and <numberOfDataPoints> is the number of data points you wish to retrieve, for example: /api/nodes/000D6F0001A30FB6/sensors/350/data/2.

An example of the JSON response is as follows, where the last two data points stored on the gateway for this sensor are returned:


```
{
  "data": [
      {
        "period": "2017-08-30T13:14:00",
        "value": 25.34
      },
      {
        "period": "2017-08-30T13:15:00",
        "value": 25.24
      },
  ]
}
```


To request the last data point for all sensors of a node, use the following endpoint (available in Version V04.01.00.01 of the EpiSensor Gateway):

HTTP GET

/api/nodes/<node_serial>/sensors/lastData

Where <node_serial> is the serial number of the node, for example: /api/nodes/000D6F0001A30FB6/sensors/lastData.

An example of the JSON response is as follows, where there are two sensors of node 000D6F0001A30FB6 which have a last data point stored on the gateway:


```
[
     {
        "id": 358
        "lastData": {
            "period": "2017-08-30T13:13:00",
            "value": 49
         }
      }
       {
        "id": 380
        "lastData": {
            "period": "2017-08-30T13:10:00",
            "value": 26.25
         }
      }
]
```


Important Note
This data will be cleared when the Gateway restarts.

Requesting Live Calibration Data

If calibration mode has been enabled for a particular node, a stream of data will be available at the following endpoint which can be used by 3rd party apps to confirm that the sensor temperature has reached a steady state.

HTTP GET

/api/nodes/<node_serial>/sensors/<sensor_ID>/calibration/data

Where <node_serial> is the serial number of the node, and <sensor_ID> is the ID of the sensor, for example: /api/nodes/000D6F0001A30FB6/sensors/350/calibration/data

An example of the JSON response is as follows, where there are ten calibration data points stored on the gateway for this sensor:


```
{
    "calibrationData": [
        {
            "period": "2017-09-13T10:49:28",
            "value": 25.58
        },
        {
            "period": "2017-09-13T10:49:34",
            "value": 25.59
        },
        {
            "period": "2017-09-13T10:49:39",
            "value": 25.58
        },
        {
            "period": "2017-09-13T10:49:45",
            "value": 25.56
        },
        {
            "period": "2017-09-13T10:49:50",
            "value": 25.54
        },
        {
            "period": "2017-09-13T10:49:56",
            "value": 25.53
        },
        {
            "period": "2017-09-13T10:50:01",
            "value": 25.51
        },
        {
            "period": "2017-09-13T10:50:07",
            "value": 25.51
        },
        {
            "period": "2017-09-13T10:50:12",
            "value": 25.5
        },
        {
            "period": "2017-09-13T10:50:18",
            "value": 25.51
        }
    ]
}
```


Changing Node Properties

Some EpiSensor nodes have configurable properties, for example the “Units of Measure” for the TES or ZHT node. Changes to node-level properties are done with an HTTP POST to the node’s endpoint as follows:

HTTP POST

/api/nodes/<node_serial>

Where <node_serial> is the serial number of that node, for example: /api/nodes/000D6F0001A30FB6

```
{
    "name": "Area 1-T",
    "description": "Area-1 Temperature",
    "enable_export_all": false,
    "temperature_properties": {
        "units_of_measure": "celsius"
    }

}
```

Here is another example for the TES node where only the units of measure are changed:

```
{
    "temperature_properties": {
        "units_of_measure": "fahrenheit"
    }

}
```

Options on the temperature property for the TES and ZHT product range for “units_of_measure” are “fahrenheit” and “celsius”, or ‘f’ or ‘c’, or any of those in uppercase i.e. the character case is not checked so F or FAHRENHEIT will also be accepted.

Code Example (curl):

```
curl -X POST \
  http://172.31.255.1:8081/api/nodes/000D6F0001A30FB6 \
  -H 'authorization: Basic Z2F0ZXdheTplcGlzZW5zb3I=' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "temperature_properties": {
        "units_of_measure": "fahrenheit"
    }

}'
```

Code Example (Java / OK HTTP):

```
OkHttpClient client = new OkHttpClient();

MediaType mediaType = MediaType.parse("application/json");
RequestBody body = RequestBody.create(mediaType, "{\n    \"temperature_properties\": {\n
\"units_of_measure\": \"fahrenheit\"\n    }\n    \n}");
Request request = new Request.Builder()
  .url("http://172.31.255.1:8081/api/nodes/000D6F0001A30FB6")
  .post(body)
  .addHeader("authorization", "Basic Z2F0ZXdheTplcGlzZW5zb3I=")
  .addHeader("content-type", "application/json")
  .addHeader("cache-control", "no-cache")
  .build();

Response response = client.newCall(request).execute();
```

Power Amp and Radio Power Properties

The presence or absence of a power amplifier in the node is indicated by the has_power_amp field in the GET node payload.

The value of the radio power property is also reported in the GET node payload and may be modified for nodes which support this property. Values in the range [+8, -43] are accepted. If the node has a power amplifier (e.g. RTO-23, RTO-20) it is possible to increase the radio power levels beyond the standard radio power compliance levels. The following table explains the range of supported values for the radio power property. For nodes with power amplifiers, all power levels greater than -7 dBm are outside compliance levels for FCC/IC and EC.

Power Setting (dBm)	Description
-7	Radio Power Level meets FCC and IC Compliance Level for nodes with a power amp.
-17	Radio Power Level meets European Compliance Level for nodes with a power amp.

In the API node POST method the radio power may be modified by specifying an integer in the range [8, -43] as shown in the example below:

```
{
    "radio_power": -17
}
```

Meter Configuration Property

The meter configuration options for ZEM-6X node are as follows:

Option	Description
NINE_S	9S/16S 4-wire Wye
FIVE_S	5S/13S 3-wire Delta
SIX_S	6S/14S 4-wire Wye
EIGHT_S	8S/15S 4-wire Delta

The current and voltage transformer ratio properties (“vt_ratio” and “ct_ratio”) are 16 bit unsigned integers and must be presented as unsigned integers in the range [1, 65535]. (0 is not a valid value).

The “ct_phase_angle” and “vt_phase_angle” properties are signed 16 bit integers representing the phase angle in minutes (60 minutes equals 1 degree phase angle). They can have a maximum value of +21600 and a minimum value of -21600.

ZDR Specific Properties

In this section, configuration of the newer 4 Relay and the older Single Relay variations of ZDR product are described. Here is an example of configuring properties for a 4 Relay ZDR node :

```
{
    "zdr_properties": {
        "remotely_armed": true,
        "pre_event_log_time": 5
        "auto_exit_event": false,
        "minimum_event_time": 30,
        "event_reset_time": 10,
        "event_averaging_count": 10,
        "ct_direction": "A_FWD_B_FWD_C_FWD"
        "frequency_analysis": "POWER_FACTOR"
        "voltage_to_current_datapath": "A_B_C"
        "ct_phase_angle": 60,
        "vt_phase_angle": -60,
        "ct_ratio": 2,
        "vt_ratio": 1,
        "nominal_frequency": 50,
        "configuration": "NINE_S"
    }
}
```

Here is an example of configuring properties for a Single Relay ZDR node :

```
{
    "zdr_properties": {
        "event_sensor_id": 342,
        "enter_event_value": 49700,
        "exit_event_value": 49800,
        "auto_exit_event": false,
        "minimum_event_time": 30,
        "event_reset_time": 10,
        "event_averaging_count": 10,
        "ct_ratio": 2,
        "vt_ratio": 1,
        "nominal_frequency": 50,
        "configuration": "NINE_S"
    }
}
```


The “remotely_armed“ property can take the boolean value true or false. The “remotely_armed” property is only present on the 4 Relay ZDR Product.

The “pre_event_log_time“ property is an unsigned integer in the range [1, 29] and is in units of seconds. The “pre_event_log_time” property is only present on the 4 Relay ZDR Product.

The “event_sensor_id“ field is a 16 bit unsigned integer. At the moment the values integer values 0 or 342 and the string values “None“ or “Line Frequency“ are permitted. This property exists on the Single Relay ZDR product only.

Note that for the Single Relay ZDR, “enter_event_value” and “exit_event_value” property value types depend on the “event_sensor_id“ that is selected. For example for Line Frequency (342), the “enter_event_value“ and “exit_event_value“ properties are unsigned 32 bit integers. For the 4 Relay ZDR, the event enter and exit values are sensor level properties, so these node level enter and exit event level properties are only present on the Single Relay ZDR. See below for more details.

The “auto_exit_event“ property can take the boolean value true or false.

The “minimum_event_time“ and “event_reset_time“ properties are 16 bit unsigned integers and are in units of seconds.

The “event_averaging_count” defaults to 10 and has a minimum value of 1 and a maximum value of 100.

The “ct_direction” properties can be configured according to the following table:

Option	Description
A_FWD_B_FWD_C_FWD	All CTs are forwards, that is installed with correctly with the arrow pointing towards the load.

A_FWD_B_FWD_C_REV	Phase A and B CTs are forwards. Phase C CT is reversed.
A_FWD_B_REV_C_FWD	Phase A and C CTs are forwards. Phase B CT is reversed.
A_FWD_B_REV_C_REV	Phase A CT is forwards. Phase B and C CTs are reversed.
A_REV_B_FWD_C_FWD	Phase A CT is reversed. Phase B and C CTs are forwards.
A_REV_B_FWD_C_REV	Phase A and C CTs are reversed. Phase B CT is forwards.
A_REV_B_REV_C_FWD	Phase A and B CTs are reversed. Phase C CT is forwards.
A_REV_B_REV_C_REV	All CTs are reversed.

The “ct_direction” property is only present on the 4 Relay ZDR Product.

The “frequency_analysis” properties can be configured according to the following table:

Option	Description
POWER_FACTOR	Frequency Analysis operation will be Power Factor.
VOLTAGE_TO_CURRENT_PHASE_ANGLE	Frequency Analysis operation will be Voltage to Current Phase Angle.
VOLTAGE_TO_VOLTAGE_PHASE_ANGLE	Frequency Analysis operation will be Voltage to Voltage Phase Angle.
CURRENT_TO_CURRENT_PHASE_ANGLE	Frequency Analysis operation will be Current to Current Phase Angle.

The “frequency_analysis” property is only present on the 4 Relay ZDR Product.

The “voltage_to_current_datapath” properties can be configured according to the following table:

Option	Description
A_B_C	Phase A Voltage can be used with Phase A Current to calculate Phase A Watts. Phase B Voltage can be used with Phase B Current to calculate Phase B Watts. Phase C Voltage can be used with Phase C Current to calculate Phase C Watts.
A_C_B	Phase A Voltage can be used with Phase A Current to calculate Phase A Watts. Phase C Voltage can be used with Phase B Current to calculate Phase B Watts. Phase B Voltage can be used with Phase C Current to calculate Phase C Watts.
B_A_C	Phase B Voltage can be used with Phase A Current to calculate Phase A Watts.

	Phase A Voltage can be used with Phase B Current to calculate Phase B Watts. Phase C Voltage can be used with Phase C Current to calculate Phase C Watts.
B_C_A	Phase B Voltage can be used with Phase A Current to calculate Phase A Watts. Phase C Voltage can be used with Phase B Current to calculate Phase B Watts. Phase A Voltage can be used with Phase C Current to calculate Phase C Watts.
C_A_B	Phase C Voltage can be used with Phase A Current to calculate Phase A Watts. Phase A Voltage can be used with Phase B Current to calculate Phase B Watts. Phase B Voltage can be used with Phase C Current to calculate Phase C Watts.
C_B_A	Phase C Voltage can be used with Phase A Current to calculate Phase A Watts. Phase B Voltage can be used with Phase B Current to calculate Phase B Watts. Phase A Voltage can be used with Phase C Current to calculate Phase C Watts.

The “voltage_to_current_datapath” property is only present on the 4 Relay ZDR Product.

The “ct_phase_angle” and “vt_phase_angle” properties are signed 16 bit integers representing the phase angle in minutes (60 minutes equals 1 degree phase angle). They can have a maximum value of +21600 and a minimum value of -21600. The “ct_phase_angle” and “vt_phase_angle” properties are the same as for the ZEM-6X.

The “ct_ratio” and “vt_ratio” property options are also the same as for the ZEM-6X.

The “nominal_frequency” property can be set to 50 or 60. This is the nominal mains electricity frequency in Hertz.

The “configuration“ property options are the same as for the “configuration“ property on the ZEM-6X node.

ZHM Specific Properties

Here is an example of configuring properties for a ZHM node:

```
{
       "zhm21_properties": {

            "enable_user_interface": true,
            "config_mbus_device_address": 253,
            "config_mbus_device_baud": 0
    }
}
```

The “enable_user_interface” property may be set to true or false. The “config_mbus_device_address” property is an integer in the range [0, 255]. The “config_mbus_device_baud” maybe configured according to the following table.

Value	Description
0	Indicates scanning on the three approved rates of 300, 2400, 9600
300	Set Baud Rate 300
600	Set Baud Rate 600
1200	Set Baud Rate 1200
2400	Set Baud Rate 2400
4800	Set Baud Rate 4800
9600	Set Baud Rate 9600
19200	Set Baud Rate 19200
38400	Set Baud Rate 38400

To get a list of available properties for a particular node type (i.e. TES, ZEM or ZDR) send a GET request to the endpoint above. This will return information on the properties available for that node and the expected data types.

Many node level properties are read-only. To check if a sensor or node property is writable for a particular EpiSensor product, please consult the product documentation. If a client attempts to change a read-only property, an error will be returned.

Note that there is latency with node-level properties. For more information, see the “Command Workflow” section below.

Change a property of one Sensor

To update the property of a particular sensor, for example, to change a reporting interval of a sensor from 5 minutes to 60 minutes, an HTTP request should be sent to the relevant endpoint for that sensor that includes the properties to be updated:

HTTP POST

/api/nodes/<node_serial>/sensors/<sensor_ID>

With the following JSON payload:

```
{
   "reporting_interval":60
```

```
}
```

For this reason, the Gateway will respond to say a command has been queued, rather than executed. If a sensor has a command pending in a queue, the "in-sync" flag will be set to false.

The reporting interval may be set to values ranging from 1 minute to 1440 minutes (24 hours). Only integer values are accepted.

To change the reporting mode of a particular sensor, make the following HTTP request:

HTTP POST

/api/nodes/<node_serial>/sensors/<sensor_ID>

With the following JSON payload:

```
{
   "reporting_mode": "SNAP_TO_CLOCK"
}
```

The options for the Reporting Mode property are as follows:

Option	Description
OFF	Do not send any data from this sensor
DELTA	Send data when the measurement changes by a value greater than or equal to the “reporting delta”
INTERVAL	Send data every time the “reporting interval” is reached. Data points will have a ‘seconds’ value.
INTERVAL_AND_DELTA	Send data every time the “reporting interval” is reached AND every time the measurement changes by a value greater than or equal to the “reporting delta”. The reporting interval timer is not reset if a delta is triggered.
INTERVAL_OR_DELTA	Send data every time the “reporting interval” is reached AND every time the measurement changes by a value greater than or equal to the “reporting delta”. The reporting interval timer is reset if a delta is triggered within the reporting interval.
SNAP_TO_CLOCK	Send data “on the top of the minute” e.g. at 12:00:00, 12:05:00
LIVE_STREAM	Live data is streamed from the node at a rate of 1 data point per second.

To change the reporting delta value of a particular sensor, make the following HTTP request:

HTTP POST

/api/nodes/<node_serial>/sensors/<sensor_ID>

With the following JSON payload:

```
{
   "reporting_delta": 10.0
}
```

The value specified in the JSON payload must be greater than 0.

Important Note
To make these 'node level' changes, a message has to be sent to (and acknowledged by) the node over the wireless sensor network, so there will be latency in executing the command.

ZDR Specific Sensor Properties

For the 4 Relay ZDR Product, the enter and exit event value properties are properties of the relay state sensors 71, 72, 73 and 74. As shown in the following JSON which is returned from a GET request to sensor 71 of a 4 Relay ZDR Node.

```
{
    "name": "Relay 1",
    "id": 71,
    "units": "state",
    "export_enabled": false,
    "export_identifier": "000D6F000D7EEDA6_71",
    "in_sync": true,
    "reporting_mode": "INTERVAL_AND_DELTA",
    "reporting_interval": 240,
    "logging_mode": "ON",
    "reporting_delta": 1,
    "relay_state_properties": {
        "high_enter_event_value": 50.3,
        "high_exit_event_value": 50.2,
        "low_exit_event_value": 49.8,
        "low_enter_event_value": 49.7
    }
}
```

The sensor level enter and exit event value properties may be modified by a POST request to the sensor as follows:

HTTP POST

/api/nodes/<node_serial>/sensors/<sensor_ID>

With the following JSON payload:

```
{
    "relay_state_properties": {
        "high_enter_event_value": 50.3,
        "high_exit_event_value": 50.2,
        "low_exit_event_value": 49.8,
        "low_enter_event_value": 49.7
    }
}
```

The enter and exit even value properties can be an integer or a double value and are in units of Hertz.

Using Set Sensor Value

Set Sensor Value is used for setting calibration constants on a sensor, for resetting cumulative registers (for example a kWh register on an electricity meter) and for remote control of switches. To set a sensor value using the API, which mirrors the feature available on the Gateway's web interface, an HTTP request should be sent to the relevant endpoint for that sensor that includes the value to be set.

HTTP POST

/api/nodes/<node_serial>/sensors/<sensor_ID>

With the following JSON payload:

```
{
    "sensor_value": 150.0
}
```

Note that the 'sensor' in question must have write permissions and also must have reported at least one data point to the Gateway.

To make this change, a message has to be sent to (and acknowledged by) the node over the wireless sensor network, so there will be latency in executing the command.

For this reason, the Gateway will respond to say a command has been queued, rather than executed. If a sensor has a command pending in a queue, the "in-sync" flag will be set to false.

If the command has been returned successfully, the Gateway should return HTTP status code 202.

Send a node level command

Commands can be sent to individual nodes via the API to take actions, similar to the 'action' drop-down menu on the nodes list of the Gateway's web interface.

Nodes can be reset or factory reset with this endpoint, and node-specific actions taken like opting a ZDR out of a demand response event.

HTTP POST

/api/nodes/<node_serial>/command/<command_type>

The body of the request should be empty. If the command has been queued for processing, the Gateway will return HTTP status code 202.

The list of available node level commands is as follows:

Command	Description	Example
startCalibration	Start calibration mode on the node.	/api/nodes/000D6F00030516C4/command/startCalibration
stopCalibration	Stop calibration mode on the node.	/api/nodes/000D6F00030516C4/command/stopCalibration
deleteCalibration	Delete the calibration table on the node (and stored calibration points on the gateway)	/api/nodes/000D6F00030516C4/command/deleteCalibration
optOut	ZDR Specific (see below)	/api/nodes/000D6F00030516C4/command/optOut
refreshNeighbourList	Refresh the neighbour list properties for this node. This includes the child list property.	/api/nodes/000D6F00030516C4/command/refreshNeighbourList
sync		/api/nodes/000D6F00030516C4/command/sync
factoryReset	Factory reset the node.	/api/nodes/000D6F00030516C4/command/factoryReset
restart	Restart the node.	/api/nodes/000D6F00030516C4/command/restart
eraseDataLog	Erase logged data on the node.	/api/nodes/000D6F00030516C4/command/eraseDataLog
exportAll	Enabled export on all sensors of the node.	/api/nodes/000D6F00030516C4/command/exportAll
exportNone	Disable export on all sensors of the node.	/api/nodes/000D6F00030516C4/command/exportNone
enableInjectionTest	ZDR Specific. On receipt of this command the node will schedule a frequency injection test to start at the next 15 minutes past the hour. The test will last for 35 minutes.	/api/nodes/000D6F00030516C4/command/enableInjectionTest
disableInjectionTest	ZDR Specific. On receipt	/api/nodes/000D6F00030516C4/command/disableInjectionTest

	of this command the node will cancel the ongoing or next scheduled frequency injection test.	

An example to opt a ZDR node out of an active demand response event would be an HTTP POST to the following endpoint:

/api/nodes/<node_serial>/command/optOut

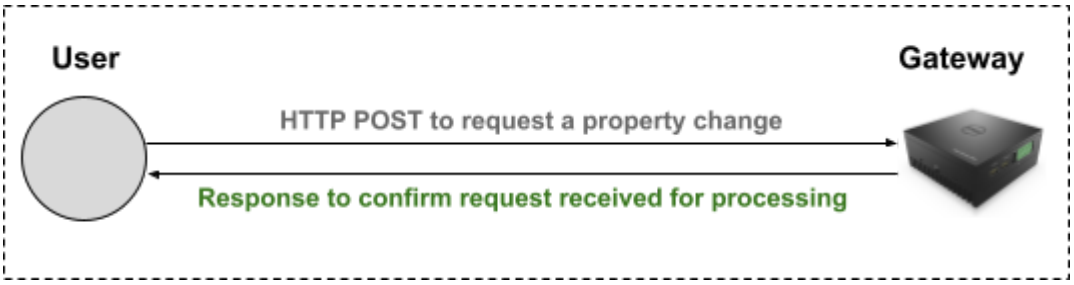
To confirm that the ZDR in question has in fact been opted out, send a GET request to **/api/nodes/<node_serial>/sensors/<sensor_ID>** (using sensor ID 26 or 27 for the ZDR) to confirm the current event status is zero.

API Request Workflow

When sending requests to one or more nodes / sensors via the API, the client / server workflow should be as follows:

Step 1: Request

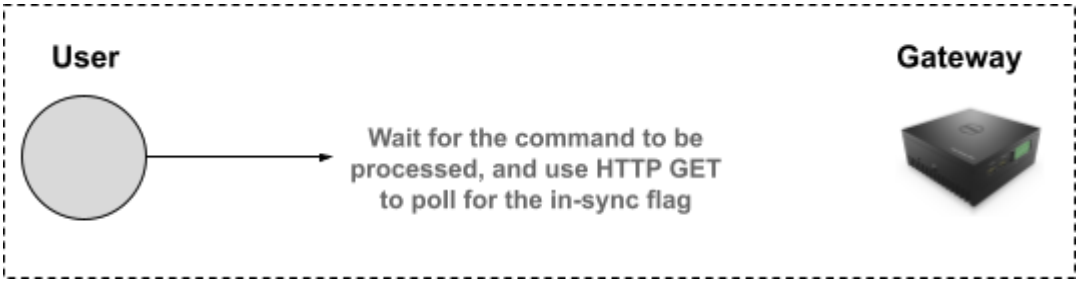
Send the command to the relevant endpoint on the API using an HTTP POST request.



Step 2: Confirm & Poll for sync

Record the response and whether the message has succeeded in being queued for processing (a reason why the message would not be queued could be that the ZAP is not connected, for example).

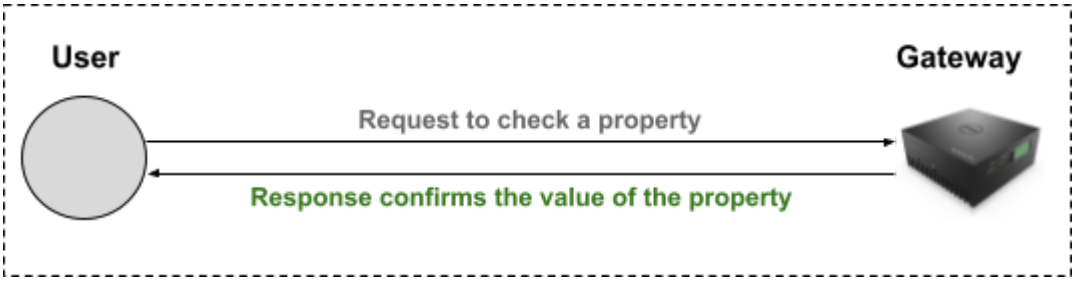
2 seconds later, send an HTTP GET request the endpoint of the sensor and note the "in-sync" status. If "in-sync" is returned false, the command is still pending. Keep querying every 5 seconds until the "in-sync" flag is returned true.



Step 3: Confirm

Compare the property change that was requested to the property that is returned when "in-sync" is returned true. If in the example above "reporting_interval" is returned as "60", the command has been successful and the reporting interval of the node has been updated.

If "reporting_interval" has not been updated, log the fact that the command didn't succeed and move on to the next node (not just the next sensor) - the node may be offline at the time the command is being sent.



One exception to the workflow above is a sensor's Export ID - this value is stored on the Gateway, and so a response message indicating success or failure should be returned by the API without latency.