WoTKit

Release 1.6.0.SNAPSHOT

Sensetecnic

August 31, 2015

Contents

The WoTKit is a web-centric toolkit that helps organizations manage sensors and actuators to collect, aggregate, store and process sensor data and react to changes in the physical and virtual world.

To get started quickly, see the Quick Start guide. For more information see consult the V1 API Reference.

Please send any questions and feedback to info@sensetecnic.com.

CHAPTER 1

Guide

1.1 WoTKit API Guides

In this section we have listed tutorials which guide users through the API. For reference documentation, refer to Sensor Data.

1.1.1 Querying Sensor Data

WoTKit provides flexibility in how you want to query your data. In this section, we walk through the different ways of building a query to get sensor data out of wotkit. The queries are constructed using query parameters which you append to a URL endpoint.

Typically applications will need to query for raw time-series data of a sensor or group of sensors. There are two different types of queries: *Recent Queries* and *Time Range Queries*.

The following document will walk through some examples of how to take advantage of *Recent Queries* and *Time Range Queries*

Recent Queries

To query for recent data, the API provides parameters for you to either:

1. get the *n* most recent sensor data
2. get sensor data from *t* milliseconds in the past until now

In this section we'll dive in quickly and briefly show an example of *Recent Num Queries* and *Recent Time Queries*.

Recent Num Queries

By default, the data endpoint will return the 1000 most recent sensor data items. Try it using a URL like this:

example

```
http://wotkit.sensetecnic.com/api/v2/sensors/sensetecnic.mule1/data
```

The response should look similar to the following:

```
{
  "numFound": 0,
  "data": {
    "data": [
      {
        "id": 47902511,
        "timestamp": "1398698531445",
        "timestamp_iso": "2013-11-29T00:46:36.056Z",
        "sensor_id": 1,
        "sensor_name": "sensetecnic.mule1",
        "value": 69,
        "lng": -123.17608,
        "lat": 49.14103
      },
      {
        "id": 47902514,
        "timestamp": "1398698531445",
        "timestamp_iso": "2013-11-29T00:46:39.556Z",
        "sensor_id": 1,
        "sensor_name": "sensetecnic.mule1",
        "value": 52,
        "lng": -123.17599,
        "lat": 49.13919
      },
      /*more data*/
    ],
    "fields": [ /*Fields information*/ ]
  },
  "query": {
    "limit": 1000,
    "recent_n": 1000
  }
}
```

Field	Description
num-Found	The total number of elements matching this query (Note: that *numFound* is deprecated showing a value of 0)
data	The enclosed sensor_data. Always sorted from oldest to newest timestamp
query	Contains the interpreted query from the request. For debugging.
metadata	Extra information. Depends on use case.

The query field is particularly interesting because it tells you how the query was interpreted. In this case, the query has a **limit** of *1000* and a **recent_n** of *1000*. A recent_n query fetches the **n** most recent items. This is useful when API users want to peek at the recent data without having to construct complex queries.

In essence, the query we ran is a convenient default for the explicit version:

example

```
http://wotkit.sensetecnic.com/api/v2/sensors/sensetecnic.mule1/data?limit=1000&recent_n=10
```

Next we can try a recent_t query, which looks up the timestamp.

Recent Time Queries

Recent Time Queries are very similar to Recent Num Queries. While Recent Num Queries look at data count i.e. the last 10 elements, or the last 50 elements, Recent Time queries look at the timestamp instead. So, it's useful for where we're interested in the elements from the last hour, or the last 12 hours.

Request

example

```
http://wotkit.sensetecnic.com/api/v2/sensors/sensetecnic.mule1/data?recent_t=10000
```

Response

```
{
  "numFound": 0,
  "data":{
    "data": [
        {
            "id": 47967438,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T18:34:09.557Z",
            "sensor_id": 1,
            "sensor_name": "sensetecnic.mule1",
            "value": 62,
            "lng": -123.14509,
            "lat": 49.186
        },
        {
             "id": 47967445,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T18:34:13.059Z",
            "sensor_id": 1,
            "sensor_name": "sensetecnic.mule1",
            "value": 53,
            "lng": -123.1454,
            "lat": 49.18565
        },
        {
            "id": 47967446,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T18:34:16.557Z",
            "sensor_id": 1,
            "sensor_name": "sensetecnic.mule1",
            "value": 67,
            "lng": -123.14844,
            "lat": 49.18323
        }
    ],
    "fields": [ /*Fields information*/ ]
  }
  "query": {
      "limit": 1000,
      "recent_t": 10000
  }
}
```

Looking at the *query* field this time, we can see it was interpreted as a recent_t query. The query looked for items up to 10 seconds ago (10000 milliseconds). You can verify this by inspecting the timestamp of the data.

Note: When accessing WoTKit anonymously for public data, the date string is set to UTC. When accessing it using an api-key the timezone will be set based on the account's timezone setting.

We've just shown you how to run both **Recent Queries**. One parameter to make note of is the limit parameter. At the moment, limit is capped at 1000 – which restricts how much data you get in **recent_n** and **recent_t** queries. To overcome this we will look into paging through historical data next.

Time Range Queries

At the end of the last section, we noted that there is a weakness in the recent queries which limit your ability to sift through historical data. You can page through historical data using the following query parameters. For the remainder of this tutorial we will be working with the sensor `rymndhng.sdq-test`.

Querying with Start and End

We'll start with a simple practical example. We have a defined starting time and ending time where we want to get all the data in between. I want to know what data was there between the iso timestamp `2013-11-21T11:00:51.000Z` and the iso timestamp `2013-11-29T22:59:54.862Z`, or from `start: 1385031651000` to `end: 1385765994862`

Note: It is important to note that `start` is *exclusive* and `end` is *inclusive*. When using `start=100` and `end=200` the query will return:

```
start < sensor_data.timestamp <= end
```

Query Parameters

Query Parameter	Value
start	1385031651000 (2013-11-21T11:00:51.000Z)
end	1385765994862 (2013-11-29T22:59:54.862Z)

The API requires timestamp values to be in milliseconds, thus we can execute the following request:

Request

example

`http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=1385031651000&en`

Response

```
{
  "numFound": 0,
  "data": {
    data: [
      {
          "id": 48232725,
          "timestamp": "1398698531445",
          "timestamp_iso": "2013-11-29T22:59:09.472Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 81
      },
      {
          "id": 48232726,
          "timestamp": "1398698531445",
          "timestamp_iso": "2013-11-29T22:59:09.472Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 53
      },
      {
          "id": 48232727,
          "timestamp": "1398698531445",
          "timestamp_iso": "2013-11-29T22:59:19.633Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 0
      },
      {
          "id": 48232728,
          "timestamp": "1398698531445",
          "timestamp_iso": "2013-11-29T22:59:24.715Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 56
      },
      {
          "id": 48232729,
          "timestamp": "1398698531445",
          "timestamp_iso": "2013-11-29T22:59:54.862Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 97
      }
    ],
    fields: [/* Fields information */]
  },
  "query": {
      "end": "2013-11-29T22:59:54.862Z",
      "start": "2013-11-21T11:00:51.000Z",
      "limit": 1000
  }
}
```

We can see that start/end was interpreted in the query between the start and end points, specifically `start < data[0].timestamp < ... < data[4].timestamp < end`.

Paging Through Data

The previous section illustrated a simple example returning a small range of elements. In real world applications the response of a query will often return thousands of entries. In such case you might want to sift through a small ammount of these entries at a time. Let's try querying a large range by using *start=0* and *end=2000000000000*. We will specify a *limit* of 3 to make the response more comprehendable.

Query Parameters

Query Parameter	Value
start	0 (1970-01-01T00:00:00.000Z
end	2000000000000 (2033-05-18T03:33:20.000Z)
limit	3

Request

---

example

`http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=0&end=2000000000`

Response

```
{
    "numFound": 0,
    "data": {
        data: [
        {
            "id": 48232722,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-21T10:58:51.000Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 6.7
        },
        {
            "id": 48232723,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-21T10:59:51.000Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 6.8
        },
        {
            "id": 48232724,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-21T11:00:51.000Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 6.9
        }
      ],
      "fields": [ /*Fields information*/ ]
    },
    "query": {
        "end": "2033-05-18T03:33:20.000Z",
        "start": "1970-01-01T00:00:00.000Z",
        "limit": 3
    }
}
```

In this query we have only asked for 3 elements. We can page data by setting the parameter `offset` in our request. In our example, we can retrieve the next page by setting `offset=data.size`, in our case 3: `offset=3`. By specifying `offset = prev_offset + data.size` we can page through data in each subsequent request. Now, let's retry the last query with an offset.

Query Parameters

Parameter	Value
start	0 (same as before
end	2000000000000 (same as before)
limit	3
offset	3

Request

example

```
http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=0&end=200000000000
```

Response

```
{
    "numFound": 0,
    "data": {
        data: [
        {
            "id": 48232725,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T22:59:09.472Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "valua": 81
        },
        {
            "id": 48232726,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T22:59:09.472Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "valua": 53
        },
        {
            "id": 48232727,
            "timestamp": "1398698531445",
            "timestamp_iso": "2013-11-29T22:59:19.633Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "valua": 0
        }
      ],
      "fields": [ /*an array of expected values*/ ]
    },
    "query": {
        "offset": 3,
        "end": 2000000000000,
        "start": 0,
        "limit": 3
    }
```

}

Once again, looking at the query, we can now see that offset is specfied as 3. We can also verify that an offset was used by looking at `id` and `timestamp` of the two responses. The **last** element of the first response has `id: 48232724` and `timestamp_iso: "2013-11-21T11:00:51.000Z"`. The **first** element in the second response has `id: 48232725` and `timestamp_iso: "2013-11-29T22:59:09.472Z"`. You can easily verify that they are in sequence.

Advanced Time Range Queries

In general, using *start, end, offset* provides enough flexibility for most queries. However, sensors are allowed to have multiple data on the same timestamp. This can easily happen when historical data is `PUT` into the system. As a result several datapoints can have identical timestamps. What this means is that you cannot expect the timestamp value to be unique for a sensor data.

To solve this we can use the parameters `start_id` and `end_id` for a more precise selection of start and end elements.

We'll start off with our first query

example

```
http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=0&end=20000000000
```

Response

```
{
  "numFound": 0,
  "data": {
    data: [
      {
          "id": 48232722,
          "timestamp": "1385031531000",
          "timestamp_iso": "2013-11-21T10:58:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.7
      },
      {
          "id": 48232723,
          "timestamp": "1385031531000",
          "timestamp_iso": "2013-11-21T10:59:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.8
      },
      {
          "id": 48232724,
          "timestamp": "1385031651000",
          "timestamp_iso": "2013-11-21T11:00:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.9
      },
      {
          "id": 48232725,
          "timestamp": "1385765949472",
          "timestamp_iso": "2013-11-29T22:59:09.472Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 81
      }
    ],
    "fields": [/*Fields*/],
  },
  "query": {
      "start": 0,
      "limit": 4
  }
}
```

If we want to re-run this query in the future using the information we obtained in this query we will use the last item's timestamp "1385765949472" (2013-11-29T22:59:09.472Z) as the start value:

Request

example

http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=1385765949472&enc

Response

```
{
  "numFound": 0,
  "data": {
    "data": [
        {
           "id": 48232727,
           "timestamp": "1385765959633",
           "timestamp_iso": "2013-11-29T22:59:19.633Z",
           "sensor_id": 531,
           "sensor_name": "rymndhng.sdq-test",
           "valua": 0
        },
        {
           "id": 48232728,
           "timestamp": "1385765964715",
           "timestamp_iso": "2013-11-29T22:59:24.715Z",
           "sensor_id": 531,
           "sensor_name": "rymndhng.sdq-test",
           "valua": 56
        },
        {
           "id": 48232729,
           "timestamp": "1385765994862",
           "timestamp_iso": "2013-11-29T22:59:54.862Z",
           "sensor_id": 531,
           "sensor_name": "rymndhng.sdq-test",
           "value": 97
        },
         {
           "id": 48232730,
           "timestamp": "1385766024862,","
           "timestamp_iso": "2013-11-29T23:00:24.862Z",
           "sensor_id": 531,
           "sensor_name": "rymndhng.sdq-test",
           "value": 6.7
        }
    ],
    "fields": [/*Fields information*/]
  },
  "query": {
      "start": 1385765949472,
      "limit": 4
  }
}
```

Everything looks fine doesn't it? Although the timestamps seem incremental there is a problem we are unaware of. We have actually skyppped an element because of the existence of duplicate timestamps. If we run the following request querying the entire range this will become more aparent:

Request

example

`http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data`

Response

```
{
  "numFound": 0,
  "data": {
     data: [
       {
          "id": 48232722,
          "timestamp": "1385031531000",
          "timestamp_iso": "2013-11-21T10:58:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.7
       },
       {
          "id": 48232723,
          "timestamp": "1385031591000",
          "timestamp_iso": "2013-11-21T10:59:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.8
       },
       {
          "id": 48232724,
          "timestamp": "1385031651000",
          "timestamp_iso": "2013-11-21T11:00:51.000Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.9
       },
       {
          "id": 48232725,
          "timestamp": "1385765949472",
          "timestamp_iso": "2013-11-29T22:59:09.472Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 81
       },
       {  "_comment": "HIDDEN DUE TO DUPLICATE TIMESTAMP"
          "id": 48232726,
          "timestamp": "1385765949472",
          "timestamp_iso": "2013-11-29T22:59:09.472Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 53
       },
       {
          "id": 48232727,
          "timestamp": "1385765959633",
          "timestamp_iso": "2013-11-29T22:59:19.633Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 0
       },
       {
          "id": 48232728,
          "timestamp": "1385765964715",
          "timestamp_iso": "2013-11-29T22:59:24.715Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "valua": 56
       },
       {
          "id": 48232729,
          "timestamp": "1385765994862",
          "timestamp_iso": "2013-11-29T22:59:54.862Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 97
       },
       {
          "id": 48232730,
          "timestamp": "1385766024862",
          "timestamp_iso": "2013-11-29T23:00:24.862Z",
          "sensor_id": 531,
          "sensor_name": "rymndhng.sdq-test",
          "value": 6.7
       }
     ],
     "fields": [ /*Fields information*/ ]
  },
  "query": {
      "limit": 100,
      "recent_n": 10
  }
}
```

You can see that the highlighted lines for `id: 48232726` did not exist in either of our previous queries. For example, in *Querying with Start and End*, we performed a query for data after timestamp 1385765949472, but the element highlighted above was not returned.

To solve this issue, use the parameter `start_id`. This parameter can be used in conjuction with `start` to specify specify which data element's id to start with. This works because sensor data are uniquely identified using a tuple `(timestamp, id)`.

Let's rerun the second query with `start_id: 48232725` from the first query.

Request

example
`http://wotkit.sensetecnic.com/api/v2/sensors/rymndhng.sdq-test/data?start=1385031651000&en`

Response

```
{
  "numFound": 0,
  "data": {
    "data": [
        {
            "id": 48232726,
            "timestamp": "1385765949472",
            "timestamp": "2013-11-29T22:59:09.472Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 53
        },
        {
            "id": 48232727,
            "timestamp": "1385765959633",
            "timestamp": "2013-11-29T22:59:19.633Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 0
        },
        {
            "id": 48232728,
            "timestamp": "1385765964715",
            "timestamp": "2013-11-29T22:59:24.715Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 56
        },
        {
            "id": 48232729,
            "timestamp": "1385765994862",
            "timestamp": "2013-11-29T22:59:54.862Z",
            "sensor_id": 531,
            "sensor_name": "rymndhng.sdq-test",
            "value": 97
        }
      ],
    "fields": [ /*Fields information*/ ]
  }
  "query": {
      "start": 1385765949472,
      "limit": 4,
      "start_id": 48232725
  }
}
```

When we used the parameter `start_id` we got a response with the element whose *id: 48232726*‘. The `start_id` allowed us to filter ids greater than 48232726. `end_id` works the same way as `start_id` if you really need fine-grained control over the range of a data query.

Summary of Time Range Data Query

We have learned all the parameters that can be used in a sensor query. But which approach should you use?

1. Without start_id or end_id, the query range is performed like this:

```
start < data_ts <= end
```

where `data_ts` is the sensor data's timestamp, and `data_id` is the data's id element.

2) With start_id and/or end_id, the query range adds extra checks near the bounds like this:

```
        (start < data_ts <= end)
        OR (data_ts = start AND data_id > start_id)
        OR (data_ts = end   AND data_id <= end_id)
```

Below is a quicky summary of what each query parameter means:

Parameter	Type	Description
`start`	timestamp	The absolute starting point (in milliseconds since Jan 1, 1970).
`start_id`	id	The starting id of sensor_data at timestamp `start`. Used for paging.
`end`	timestamp	The absolute ending timestamp (in milliseconds since Jan 1, 1970)
`end_id`	timestamp	The end id of sensor_data with timestamp `end`. Used for paging.

Additional Sensor Data Query Recipes

You can combine the information above in novel ways to query sensor data.

1. Use start_id instead of start for start of query

   In the documentation, we used `start_id` alongisde `start`, but actually, this is optional. If you use `start_id` without `start`, WoTKit will lookup the `timestamp` of the element with id `start_id`, and then use that as the starting timestamp.

2. Making Start Inclusive

   From *Summary of Time Range Data Query*, it shows the start range is exclusive. But, there is a way to make this inclusive. If you set `start_id:   0`, it will make the data range inclusive.

1.2 V1 API Reference

This section contains API References for V1 of WoTKit's API. In addition to the documentation posted here, our API can be explored using Swagger with the following URL http://wotkit.sensetecnic.com/api/v1/api-docs?path=v1.

1.2.1 Authentication

The WoTKit API supports three forms of authentication to control access to a user's sensors and other information on the WoTKit.

1. Basic authentication using the user's name and password
2. Basic authentication with *Keys* (key id and key password)
3. OAuth2 authorization of server-based *Applications*

Using the WoTKit portal, developers can create *keys* for use by one or more sensor gateways or scripts. Users can also register new server side applications and then authorize these applications to allow them to access a user's sensors on their behalf.

Note: Most examples in this document use basic authentication with keys or WoTKit username and passwords. However, OAuth2 authorization is also possible by removing the id and password and by appending an access_token parameter. See *Registered Applications and OAuth2* for details.

Methods privacy

Some API methods are private and will return an HTTP status code of 403 Forbidden if accessed without authenticating the request, while others are completely private or are restricted to certain users. (Currently only system administrators have access to ALL methods),

Every method has a description of its private level in one of the following forms:

- **Public** accessible to all
- **Private** accessible only to authenticated users
- **Public or Private** accessible to all, but might return different results to authenticated users.
  - Example of different results is the "get sensors" method, which might return a user's private sensors when the method is called as an authenticated user.
- **Admin** accessible only to authenticated admin users

Keys and Basic Authentication

Keys are created on the WoTKit UI (http://wotkit.sensetecnic.com/wotkit/keys) and are unique to each user.

To grant a client access to your sensors, you can create a *key*. The client can then be supplied the auto-generated 'key id' and 'key password'. These will act as username and password credentials, using basic authentication to access sensors on the user's behalf.

For instance, the following curl command uses a 'key id' and 'key password' to get information about the sensor **sensetecnic.mule1**.

Note: Replace the `{key_id}` and `{key_password}` in the code below with your own generated keys. To generate them visit the WoTKit UI at http://wotkit.sensetecnic.com/wotkit/keys, click on *New API Key*, after filling form with *Key Name* and *Key Description* to track your keys you will be presented with the values you can use.

example

```
curl --user {key_id}:{key_password}
``http://wotkit.sensetecnic.com/api/v1/sensors/sensetecnic.mule1``
```

This returns:

```
{
        "name":"mule1",
        "fields":[
        {"name":"lat","value":49.20532,"type":"NUMBER","index":0,
         "required":true,"longName":"latitude","lastUpdate":"2012-12-07T01:47:18.639Z"},
        {"name":"lng","value":-123.1404,"type":"NUMBER","index":1,
         "required":true,"longName":"longitude","lastUpdate":"2012-12-07T01:47:18.639Z"},
        {"name":"value","value":58.0,"type":"NUMBER","index":2,
         "required":true,"longName":"Data","lastUpdate":"2012-12-07T01:47:18.639Z"},
        {"name":"message","type":"STRING","index":3,
         "required":false,"longName":"Message"}
                ],
        "id":1,
        "visibility":PUBLIC,
        "owner":"sensetecnic",
        "description":"A big yellow taxi that travels from
                       Vincent's house to UBC and then back.",
        "longName":"Big Yellow Taxi",
        "latitude":51.060386316691,
        "longitude":-114.087524414062,
        "lastUpdate":"2012-12-07T01:47:18.639Z"}
}
```

Registered Applications and OAuth2

Applications registered with the WoTKit UI (http://wotkit.sensetecnic.com/wotkit/apps) provide an easy way to allow several clients access to a WoTKit's user data. A common scenario is when a developer creates an application that publishes data *on behalf* of other WoTKit users.

For example, to grant a third-party client access to your sensors, you first register an *application*. The client can then be supplied the 'application client id' and auto-generated 'application secret'. These will act as credentials, allowing clients to access the WoTKit on your behalf, using OAuth2 authorization. You can always delete the application and revoke access to any clients using the generated oauth credentials.

In more detail, an OAuth2 authorization will ask the user's permission for a client to utilize the application credentials on the user's behalf. If the user allows this, an access token is generated. This access token can then be appended to the end of each WoTKit URL. In this case no further id/passwords are needed.

For instance, the following curl command uses an access token to get information about the sensor **sensetecnic.mule1**.

Note: Replace the `{access_token}` the request below with your own generated access token as explained below

example

```
curl ``http://wotkit.sensetecnic.com/api/v1/sensors/sensetecnic.mule1?access_token={access_
```

In order to obtain an access token a client must follow the following steps, which follow the oauth2 specification (http://oauth.net/2/).

1. An attempt to access the WoTKit is made by providing an 'application client id' and requesting a code. This can be obtained

   ```
   http://wotkit.sensetecnic.com/api/oauth/authorize?client_id={application
   client id}&response_type=code&redirect_uri={redirect_uri}
   ```

2. If no user is currently logged in to the WoTKit, a login page will be presented. A WoTKit user must log in to continue.

3. A prompt asks the user to authorize the 'application client id' to act on their behalf. Once authorized, a code is provided.

4. The user is redirected to a *redirect_uri* that obtains an access token that can be appended to the end of each URL to perform queries on behalf of the user.

Note: An application's *Client ID* and *Application Secret* can be found at after you have created an application in the WoTKit UI: http://wotkit.sensetecnic.com/wotkit/apps/'{application-id}

The following example in PHP exemplifies the flow explained above. The example below is deployed at a *{redirect_uri}* that is pointed to by the WoTKit after the request in (1) above is made.

```
<?php
        $code = $_GET['code'];
        $access_token = "none";
        $ch = curl_init();
```

```
        if(isset($code)) {
                // try to get an access token
                $params = array("code" => $code,
                        "client_id"=> {application client id},
                        "client_secret" => {application secret},
                        "redirect_uri" => {redirect uri},
                        "grant_type" => "authorization_code");
                $data = ArraytoNameValuePairs ($params);

                curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
                curl_setopt($ch, CURLOPT_URL, "http://wotkit.sensetecnic.com/api/oauth/token");
                curl_setopt($ch, CURLOPT_POST, TRUE);
                curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

                $access_token = json_decode($response)->access_token;
        }
?>
```

Access Token Facts

When obtaining an access token, the 'response' field holds the access token required by an application to make future requests on behalf of a user:

- `response->access_token`
- `response->expires_in`

Note: The default value of response->expires_in is approx. 43200 seconds (or 12 hrs)

Smart Streets Authentication

The WoTKit API for Smart Streets supports basic authentication using user name and password, WoTKit keys, as well as a developer key. Note that Smart Streets does not support OAuth2.

1.2.2 Error Reporting

Errors are reported with an HTTP status code accompanied by an error JSON object. The object contains the status, an internal error code, user-displayable message, and an internal developer message.

For example, when a sensor cannot be found, the following error is returned:

```
HTTP/1.1 404 Not Found

{
    "error": {
        "status": 404,
        "code": 0,
        "message": "No thing with that id or name",
        "developerMessage": ["my_sensor"]
    }
}
```

1.2.3 Sensors

A sensor represents a physical or virtual sensor or actuator. It contains a data stream made up of *fields*.

A sensor has the following attributes:

Name	Value Description
id	the numeric id of the sensor. This may be used in the API in place of the sensor name.
name **	**the name of the sensor.** Note that the global name is `{username}.{sensorname}`. When logged in as a the owner, you can refer to the sensor using only `{sensorname}`. To access a public sensor created by another user, you can refer to it by its numeric id or the global name, `{username}.{sensorname}`.
description **	a description of the sensor for text searches.
longName **	longer display name of the sensor.
url	**DEPRECATED**
latitude	**the latitude location of the sensor in degrees.** This is a static location used for locating sensors on a map and for location-based queries. (Dynamic location (e.g. for mobile sensors) is in the *lat* and *lng* fields of sensor data.)
longitude	**the longitude location of the sensor in degrees.** This is a static location used for locating sensors on a map and for location-based queries. (Dynamic location (e.g. for mobile sensors) is in the *lat* and *lng* fields of sensor data.)
lastUpdate	**last update time in milliseconds.** This is the last time sensor data was recorded, or an actuator script polled for control messages.
visibility	**PUBLIC**: The sensor is publicly visible **PRIVATE**: The sensor is only visible to the owner. In any case posting *data* to the sensor is restricted to the sensor's owner.
owner	the owner of the sensor
fields	the expected data fields, their type (number or string), units and if available, last update time and value. (For more info: *Sensor Fields*)
tags	the list of tags for the sensor (For more info: *Tags*)
data	sensor data (not shown yet)

** Required when creating a new sensor.

Querying Sensors

A list of matching sensors may also be queried by the system.

The current query parameters are as follows:

Name	Value Description
scope	**all** - all sensors the current user has access to **subscribed** - the sensors the user has subscribed to | **contributed** - the sensors the user has contributed to the system.
metadata	**a key:value metadata pair** Example: **metadata=appliance:toaster**
tags	list of comma separated tags
orgs	list of comma separated organization names
private	**DEPRECATED**, use **visibility** instead. (*true* - private sensors only; *false* - public only‘).
visibility	filter by the visibility of the sensors, either of **public**, or **private**
text	text to search for in the name, long name and description
active	when true it returns sensors that have been updated in the last 15 minutes; when false it returns sensors that have *not* been updated in the last 15 minutes.
offset	offset into list of sensors for paging
limit	limit to show for paging. The maximum number of sensors to display is 1000.
location	**geo coordinates for a bounding box to search within.** Format is **yy.yyy,xx.xxx:yy.yyy,xx.xxx**, and the order of the coordinates are North,West:South,East. Example: **location=56.89,-114.55:17.43,-106.219**

Note: If active is ommited the query will not evaluate if a sensor has, or has not, been updated in the last 15 minutes.

To query for sensors, add query parameters after the sensors URL as follows:

URL	http://wotkit.sensetecnic.com/api/v1/sensors?{query}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing a list of sensor descriptions matching the query.

example

```
curl --user {id}:{password}
``http://wotkit.sensetecnic.com/api/v1/sensors?tags=canada``
```

Output:

```
[
 {
  "id": 71,
  "name": "api-data-test",
  "longName": "api-data-test",
  "description": "api-data-test",
  "tags": [
    "canada",
    "data",
    "winnipeg"
  ],
  "latitude": 0,
  "longitude": 0,
  "visibility": "PUBLIC",
  "owner": "sensetecnic",
  "lastUpdate": "2013-03-09T03:12:35.438Z",
  "created": "2013-07-01T23:17:37.000Z",
  "subscriberNames": [],
  "fields": [
    {
      "name": "lat",
      "longName": "latitude",
      "type": "NUMBER",
      "index": 0,
      "required": false,
      "value": 0
    },
    {
      "name": "lng",
      "longName": "longitude",
      "type": "NUMBER",
      "index": 1,
      "required": false,
      "value": 0
    },
    {
      "name": "value",
      "longName": "Data",
      "type": "NUMBER",
      "index": 2,
      "required": true,
      "value": 5,
      "lastUpdate": "2013-03-09T03:12:35.438Z"
    },
    {
      "name": "message",
      "longName": "Message",
      "type": "STRING",
      "index": 3,
      "required": false,
      "value": "hello",
      "lastUpdate": "2013-03-09T03:12:35.438Z"
    }
  ],
  "publisher": "sensetecnic",
  "thingType": "SENSOR"
 }
]
```

Viewing a Single Sensor

To view a single sensor, query the sensor by sensor name or id as follows:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body describing a sensor.

example

```
curl --user {id}:{password}
``http://wotkit.sensetecnic.com/api/v1/sensors/sensetecnic.mule1``
```

Output:

```
{
  "id": 1,
  "name": "mule1",
  "longName": "Yellow Taxi 2",
  "description": "A big yellow taxi that travels from Vincent's house to UBC and then back.",
  "tags": [
    "gps",
    "taxi"
  ],
  "imageUrl": "",
  "latitude": 51.06038631669101,
  "longitude": -114.087524414062,
  "visibility": "PUBLIC",
  "owner": "sensetecnic",
  "lastUpdate": "2014-06-19T22:45:36.556Z",
  "created": "2013-07-01T23:17:37.000Z",
  "subscriberNames": [
    "mike",
    "fred",
    "nhong",
    "smith",
    "roseyr",
    "mitsuba",
    "rymndhng",
    "lchyuen",
    "test",
    "lesula"
  ],
  "metadata": {},
  "fields": [
    {
      "name": "lat",
      "longName": "latitude",
      "type": "NUMBER",
      "index": 0,
      "units": "degrees",
      "required": false,
      "value": 49.22288,
      "lastUpdate": "2014-04-28T16:20:23.891Z"
    },
    {
      "name": "lng",
      "longName": "longitude",
      "type": "NUMBER",
      "index": 1,
      "units": "degrees",
      "required": false,
      "value": -123.16246,
      "lastUpdate": "2014-04-28T16:20:23.891Z"
    },
    {
      "name": "value",
      "longName": "Speed",
      "type": "NUMBER",
      "index": 2,
      "units": "km/h",
      "required": true,
      "value": 10,
      "lastUpdate": "2014-06-19T22:45:36.281Z"
    },
    {
      "name": "message",
      "longName": "Message",
      "type": "STRING",
      "index": 3,
      "required": false
    }
  ],
  "publisher": "sensetecnic",
  "thingType": "SENSOR"
}
```

Creating/Registering a Sensor

The sensor resource is a JSON object. To register a sensor, you POST a sensor resource to the url `/sensors`.

To create a sensor the API end-point is:

URL	http://wotkit.sensetecnic.com/api/v1/sensors
Privacy	Private
Format	json
Method	POST
Returns	**201 Created** if successful; **400 Bad Request** if sensor is invalid; **409 Conflict** if sensor with the same name already exists.

The JSON object has the following fields:

	Field Name	Information
(*REQUIRED*)	name	The unique name for the sensor field. It is required when creating/updating/deleting a field and cannot be changed. The sensor name must be at least 4 characters long, contain only lowercase letters, numbers, dashes and underscores, and can start with a lowercase letter or an underscore only.
(*REQUIRED*)	longName	The display name for the field. It is required when creating/updating/deleting a field and can be changed.
(*OPTIONAL*)	latitude	The GPS latitude position of the sensor, it will default to 0 if not provided.
(*OPTIONAL*)	longitude	The GPS longitude position of the sensor, it will default to 0 if not provided.
(*OPTIONAL*)	visibility	It will default to “PRIVATE” if not provided (even when updating a sensor).
(*OPTIONAL*)	tags	A list of tags for the sensor (For more info: *Tags*)
(*OPTIONAL*)	fields	A fields object in the format `{"name":"test-field","type":"STRING"}` (For more info: *Sensor Fields*)

example

```
curl --user {id}:{password} --request POST --header ``Content-Type: application/json''
--data-binary @test-sensor.txt `http://wotkit.sensetecnic.com/api/v1/sensors`
```

For this example, the file *test-sensor.txt* contains the following.

```
{
        "visibility":"PUBLIC",
        "name":"taxi-cab",
        "longName":"taxi-cab"
        "description":"A big yellow taxi.",
        "longName":"Big Yellow Taxi",
        "latitude":51.060386316691,
        "longitude":-114.087524414062
}
```

Creating/Registering multiple Sensors

To register multiple sensors, you PUT a list of sensor resources to the url `/sensors`.

- The sensor resources is a JSON list of objects as described in *Creating/Registering a Sensor*.
- Limited to 100 new sensors per call. (subject to change)

URL	http://wotkit.sensetecnic.com/api/v1/sensors
Privacy	Private
Format	json
Method	PUT
Returns	**201 Created** if successful; **400 Bad Request** if sensor is invalid; **409 Conflict** if sensor with the same name already exists ; **201 Created** and a JSON object in the response body describing a dictionary where the keys are the sensor names and the values are true/false depending on whether creating the sensor succeeded.

Updating a Sensor

Updating a sensor is the same as registering a new sensor other than PUT is used and the sensor name or id is included in the URL.

Note that all top level fields supplied will be updated.

- You may update any fields except “id”, “name” and “owner”.
- Only fields that are present in the JSON object will be updated.
- If “tags” list or “fields” list are included, they will replace the existing lists.
- If “visibility” is hardened (that is, the access to the sensor becomes more restrictive) then all currently subscribed users are automatically unsubscribed, regardless of whether they can access the sensor after the change.

To update a sensor owned by the current user:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}
Privacy	Private
Format	json
Method	PUT
Returns	**204 No Content** if successful.

For instance, to update a sensor description and add tags:

example

```
curl --user {id}:{password} --request PUT
--header ``Content-Type: application/json''
--data-binary @update-sensor.txt
`http://wotkit.sensetecnic.com/api/v1/sensors/taxi-cab`
```

The file *update-sensor.txt* would contain the following:

```
{
   "visibility":"PUBLIC",
   "name":"taxi-cab",
   "description":"A big yellow taxi. Updated description",
   "longName":"Big Yellow Taxi",
   "latitude":51.060386316691,
   "longitude":-114.087524414062,
   "tags": ["big", "yellow", "taxi"]
}
```

Deleting a Sensor

Deleting a sensor is done by deleting the sensor resource through a DELETE request.

To delete a sensor owned by the current user:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}
Privacy	Private
Format	not applicable
Method	DELETE
Returns	**204 No Content** if successful.

example

```
curl --user {id}:{password} --request DELETE
`http://wotkit.sensetecnic.com/api/v1/sensors/test-sensor`
```

1.2.4 Sensor Subscriptions

Sensor subscriptions are handled using the `/subscribe` URL.

Get Sensor Subscriptions

To view sensors that the current user is subscribed to:

URL	http://wotkit.sensetecnic.com/api/v1/subscribe
Privacy	Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing sensors subscribed by the user.

Subscribe

To subscribe to a non-private sensor or private sensor owned by the current user:

URL	http://wotkit.sensetecnic.com/api/v1/subscribe/{sensorname}
Privacy	Private
Format	json
Method	PUT
Returns	**204 No Content** if successful.

Unsubscribe

To unsubscribe from a sensor:

URL	http://wotkit.sensetecnic.com/api/v1/subscribe/{sensorname}
Privacy	Private
Format	json
Method	DELETE
Returns	**204 No Content** if successful.

1.2.5 Sensor Fields

Sensor fields are the fields of data saved in a sensor stream. Together they make up the sensor schema. Sensor data objects must follow declared fields.

Each sensor has the following default fields:

	Field Name	Information
(*OPTIONAL*)	value	The numerical data for the sensor.
(*OPTIONAL*)	lat	The latitude of the sensor.
(*OPTIONAL*)	lng	The longitude of the sensor.
(*OPTIONAL*)	message	The string message for the sensor.

Additional fields can be added. Each new field consists of the following:

Field Name	Information
name (*REQUIRED*)	The unique name for the sensor field. Required when creating/updating/deleting a field and cannot be changed.
longName (*REQUIRED*)	The display name for the field. Required when creating/updating/deleting a field and can be changed.
type (*REQUIRED*)	Can be "NUMBER" or "STRING". Required when creating/updating a field.
required (*OPTIONAL*)	Is a boolean field. If true, data sent to a sensor must include this field or an error will result.
units (*OPTIONAL*)	A string to identify the units to represent data.
index (*READ-ONLY*)	The numerical index of the field used to maintain ordering. This field is automatically generated by the system and is read only.
value (*READ-ONLY*)	The last value of this sensor field received by the sensor when sending data. This is a read only field set when the sensor receives data for this field.
lastUpdate (*READ-ONLY*)	The time stamp of the last value sent to the field. This is a read only field set when the sensor receives data for this field.

Querying Sensor Fields

To retrieve the sensor fields for a specific sensor:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/fields
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing the fields of the sensor is returned in the body of the response.

To query a single sensor field for a specific sensor:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/fields/{fieldName}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body describing the field is returned in the body of the response.

Updating a Sensor Field

You can update or add a sensor field by performing a PUT operation to the specified field. The field information is supplied in a JSON format.

If the sensor already has a field with the given name, it will be updated with new information. Otherwise, a new field with that name will be created.

Notes:

- When inputting field data, the sub-fields "name" and "type" are required-both for adding a new field or updating an existing one.
- Read only sub-fields such as index, value and lastUpdate should not be supplied.
- The "name" sub-field of an existing field cannot be updated.
- For user defined fields, the "longName", "type", "required", and "units" sub-fields may be updated.
- You cannot change the index of a field. If a field is deleted, the index of the following fields will be adjusted to maintain the field order.

To update/add a sensor field:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/fields/{fieldname}
Privacy	Private
Format	json
Method	PUT
Returns	**204 No Content** if successful.

For instance, to create a new field called "test-field":

example

```
curl --user {id}:{password} --request PUT
--header ``Content-Type: application/json'' --data-binary @field-data.txt
`http://wotkit.sensetecnic.com/api/v1/sensors/test-sensor/fields/test-field`
```

The file *field-data.txt* could contain the following. (Note that this is the minimal information needed to create a new field.)

```
{
        "name":"test-field",
        "type":"STRING"
}
```

To then update "test-field" sub-fields, the curl command would be used to send a PUT request.

example

```
curl --user {id}:{password} --request PUT
--header ``Content-Type: application/json'' --data-binary @field-data.txt
`http://wotkit.sensetecnic.com/api/v1/sensors/test-sensor/fields/test-field`
```

And ''field-data.txt" could now contain the following.

```
{
        "name":"test-field",
        "type":"NUMBER",
        "longName":"Test Field",
        "required":true,
        "units":"mm"
}
```

Deleting a Sensor Field

You can delete an existing sensor field by performing a DELETE and including the field name in the URL.

To delete a sensor field:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/fields/{fieldname}
Privacy	Private
Format	n/a
Method	DELETE
Returns	**204 No Content** if successful.

1.2.6 Sensor Data

In the WoTKit, *sensor data* consists of a timestamp followed by one or more named fields. There are a number of reserved fields supported by the WoTKit:

	Reserved field name	Description
(*OPTIONAL*)	timestamp	the time that the sensor data was collected. This is an ISO 8601 timestamp (for example Jan 1, 1970 UTC in ISO 8601: 1970-01-01T00:00:00Z) Optional; if not supplied, a server-supplied timestamp will be used.
(*READ-ONLY*)	id	a unique identifier for the data reading. This is to distinguish one reading from another when they share the same timestamp. This field is read only and should not be sent by the client when sending new data.
(*READ-ONLY*)	sensor_id	the globally unique sensor id that produced the data. This is a read only field generated by the wotkit that should not be sent by a client when sending new data.
(*READ-ONLY*)	sensor_name	the globally unique sensor name, in the form `{username}.{sensorname}`. This is a read only field and should not be sent by the client when sending new data.

When a new sensor is created, a number of default fields are created by the wotkit for a sensor as follows. Note that these can be changed by editing the sensor fields.

Default field name	Description
lat	the current latitude location of the sensor in degrees (number). Needed for map visualizations.
lng	the current longitude location of the sensor in degrees (number). Needed for map visualizations.
value	the primary value of the sensor data collected (number). Needed for most visualizations.
message	a text message, for example a twitter message (text). Needed for text/newsfeed visualizations.

In addition to these reserved fields, additional required or optional fields can be added by updating the *sensor fields* in the WoTKit UI or *Sensor Fields* in the API.

Note: Remember that * Python's `time.time()` function generates the system time in *seconds*, not milliseconds. To convert this to an integer in milliseconds use `int(time.time()*1000)`. Using Java you can obtain the timestam in milliseconds via `System.currentTime()`.

Sending New Data

To send new data to a sensor, POST name value pairs corresponding to the data fields to the `/sensors/{sensorname}/data` URL.

Any fields marked as *required* must be provided, or an error will be returned. There is no need to provide a timestamp since it will be assigned by the server. Data posted to the system will be processed in real time.

Note: When sending name value pairs that are not specified by the sensor's fields the server will save the data without a type. When adding a new field after sending this data WoTKit will make an attempt to cast the recorded data to the type specified by the new field.

To send new data:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/data
Privacy	Private
Format	json or x-www-form-urlencoded
Method	POST
Returns	**201 Created** if successful.

You can POST data as either *application/json* or *appliction/x-www-form-urlencoded*.

An example of POSTing using www-form-urlencoded data would be:

example

```
curl --user {username}:{password} --request POST
-d value=5 -d lng=6 -d lat=7 `http://wotkit.sensetecnic.com/api/v1/sensors/{username}.{sens
```

The same example using JSON would be:

example

```
curl --user {username}:{password} --request POST -H `Content-Type: application/json'
-d `{``value'':5, ``lng'':6, ``lat'':7}' `http://wotkit.sensetecnic.com/api/v1/sensors/{use
```

Sending Bulk Data

To send a range of data, you PUT data (rather than POST) data into the system.

- The data sent must contain a list of JSON objects, any fields marked as *required* in the sensor fields must be contained in each JSON object.
- If providing a single piece of data, existing data with the provided timestamp will be deleted and replaced. Otherwise, the new data will be added.
- If providing a range of data, any existing data within this timestamp range will be deleted and replaced by the new data.

Note: The data sent does not require a timestamp. If the timestamp is omitted WoTKit will use the current server time. Again, any fields marked as *required* must be provided.

To update data:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{username}.{sensorname}/data
Privacy	Private
Format	json
Method	PUT
Returns	****HTTP status code; No Response 204 if successful

Example of valid data:

```
[{"timestamp":"2012-12-12T03:34:28.626Z","value":67.0,"lng":-123.1404,"lat":49.20532},
{"timestamp":"2012-12-12T03:34:28.665Z","value":63.0,"lng":-123.14054,"lat":49.20554},
{"timestamp":"2012-12-12T03:34:31.621Z","value":52.0,"lng":-123.14063,"lat":49.20559},
{"timestamp":"2012-12-12T03:34:35.121Z","value":68.0,"lng":-123.14057,"lat":49.20716},
{"timestamp":"2012-12-12T03:34:38.625Z","value":51.0,"lng":-123.14049,"lat":49.20757},
{"timestamp":"2012-12-12T03:34:42.126Z","value":55.0,"lng":-123.14044,"lat":49.20854},
{"timestamp":"2012-12-12T03:34:45.621Z","value":56.0,"lng":-123.14215,"lat":49.20855},
{"timestamp":"2012-12-12T03:34:49.122Z","value":55.0,"lng":-123.14727,"lat":49.20862},
{"timestamp":"2012-12-12T03:34:52.619Z","value":59.0,"lng":-123.14765,"lat":49.20868}]
```

example

```
curl --user {username}:{password} --request PUT -H ``Content-Type: application/json'' --dat
```

where *data.txt* contains JSON data similar to the above JSON array.

Deleting Data

Currently you can only delete data by timestamp, where timestamp is in numeric or ISO form. Note that if more than one sensor data point has the same timestamp, they all will be deleted.

To delete data:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/data/{timestamp}
Privacy	Private
Format	n/a
Method	DELETE
Returns	**204 No Content** if successful.

Raw Data Retrieval

To retrieve raw data use the following:

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensor-name}/data?{query-params}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of timestamped data records.

The query parameters supported are the following:

Name	Value Description
start	the absolute start time of the range of data selected in milliseconds. (Defaults to current time.) May only be used in combination with another parameter.
end	the absolute end time of the range of data in milliseconds
after	the relative time after the start time, e.g. after=300000 would be 5 minutes after the start time (Start time MUST also be provided.)
af-terE	the number of elements after the start element or time. (Start time MUST also be provided.)
be-fore	the relative time before the start time. E.g. data from the last hour would be before=3600000 (If not provided, start time default to current time.)
be-foreE	the number of elements before the start time. E.g. to get the last 1000, use beforeE=1000 (If not provided, start time default to current time.)
re-verse	**true**: order the data from newest to oldest; **false** (default):order from oldest to newest

Note: These queries looks for timestamps > "start" and timestamps <= "end"

Formatted Data Retrieval

To retrieve data in a format suitable for Google Visualizations, we support an additional resource for retrieving data called *dataTable*.

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensor-name}/dataTable?{query-params}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** on success. A formatted JSON object in the response body containing a list of timestamped data records.

This resource is similar to *Raw Data Retrieval*, but adds two parameters: `tqx` and `tq`. You can read more about these parameters at the specification document: Chart Tools Datasource Protocol.

The complete list of available parameters is:

Name	Value Description
start	the absolute start time of the range of data selected in milliseconds. (Defaults to current time.) May only be used in combination with another parameter.
end	the absolute end time of the range of data in milliseconds
after	the relative time after the start time, e.g. after=300000 would be 5 minutes after the start time (Start time MUST also be provided.)
afterE	the number of elements after the start element or time. (Start time MUST also be provided.)
before	the relative time before the start time. E.g. data from the last hour would be before=3600000 (If not provided, start time default to current time.)
beforeE	the number of elements before the start time. E.g. to get the last 1000, use beforeE=1000 (If not provided, start time default to current time.)
reverse	**true**: order the data from newest to oldest; **false** (default):order from oldest to newest
tqx	A set of colon-delimited key/value pairs for standard parameters, defined here.
tq	A SQL clause to select and process data fields to return, explained here.

Note: When using tq sql queries, they must be url encoded. When using tqx name/value pairs, the reqId parameter is necessary.

For instance, the following would take the "sensetecnic.mule1", select all data where value was greater than 20, and display the output as an html table.

example

```
curl --user {username}:{password} http://wotkit.sensetecnic.com/api/v1/sensors/sensetecnic
dataTable?tq=select%20*%20where%20value%3E20
```

The following combines SQL filtering and formatting with a range to output the last 100 elements of the sensor where the value is greater than 55, formated using HTML:

example

```
curl --user {username}:{password} http://wotkit.sensetecnic.com/api/v1/sensors/sensetecnic
```

An example response, limited to the last 5 elements will return 3 elements, in the form:

```
google.visualization.Query.setResponse (
  {
    "version": "0.6",
    "status": "ok",
    "sig": "582888298",
    "table": {
    "cols": [
      {
        "id": "sensor_id",
        "label": "Sensor Id",
        "type": "number",
        "pattern": ""
      },
      {
        "id": "sensor_name",
        "label": "Sensor Name",
        "type": "string",
        "pattern": ""
      },
      {
        "id": "timestamp",
        "label": "Timestamp",
        "type": "datetime",
        "pattern": ""
      },
      {
        "id": "lat",
        "label": "latitude",
        "type": "number",
        "pattern": ""
      },
      {
        "id": "lng",
        "label": "longitude",
        "type": "number",
        "pattern": ""
      },
      {
        "id": "value",
        "label": "Speed",
        "type": "number",
        "pattern": ""
      },
      {
        "id": "message",
        "label": "Message",
        "type": "string",
        "pattern": ""
      }
    ],
    "rows": [


      {
        "c":[
          {"v":1.0},
          {"v":"sensetecnic.mule1"},
          {"v":new Date(2014,3,28,16,20,13)},
          {"v":49.22522},{"v":-123.166},
          {"v":66.0},{"v":null}
        ]
      },
      {
        "c":[
          {"v":1.0},
          {"v":"sensetecnic.mule1"},
          {"v":new Date(2014,3,28,16,20,16)},
          {"v":49.22422},
          {"v":-123.16398},
          {"v":58.0},
          {"v":null}
        ]
       },
      {
        "c":[
          {"v":1.0},
          {"v":"sensetecnic.mule1"},
          {"v":new Date(2014,3,28,16,20,20)},
          {"v":49.22307},
          {"v":-123.16276},
          {"v":58.0},
          {"v":null}
        ]
      }
    ],
    "p": {
      "lastId": "2014-06-19T22:45:36.281Z"
    }
  }
}
)
```

Aggregated Data Retrieval

Aggregated data retrieval allows one to receive data from multiple sensors, queried using the same parameters as when searching for sensors or sensor data. The query must be specified using one of the following 5 patterns.

Pattern 1 - With Start/End

start	The most recent starting time of the query. This value is optional and defaults to the current time.
end	A timestamp *before* the start time.
limit	Specifies the limit to return. This value is optional, with a default value of `1000`.
offset	Specifies the offset to return. This value is optional, with a default value of `0`.

Pattern 2 - With Start/After

start	A starting timestamp.
after	A **relative** timestamp *after start*.
limit	Specifies the limit to return. This value is optional, with a default value of `1000`
offset	Specifies the offset to return. This value is optional, with a default value of `0`

Pattern 3 - With Start/Before

start	A starting timestamp.
before	A **relative** timestamp *before start*.
limit	Specifies the limit to return. This value is optional, with a default value of `1000`
offset	Specifies the offset to return. This value is optional, with a default value of `0`

Pattern 4 - With Start/BeforeE

start	A starting timestamp.
beforeE	The number of elements to return before `start`
offset	Specifies the offset to return. This value is optional, with a default value of `0`

Pattern 5 - With Start/AfterE

start	A starting timestamp.
afterE	The number of elements to return after `start`
offset	Specifies the offset to return. This value is optional, with a default value of `0`

The following parameters may be added to any of the above patterns:

- scope
- tags
- private (deprecated, use visibility instead)
- visibility
- text
- active
- location (in the form: "location=-31.257,-12.55:-21.54,9.65")
- metadata
- groups

To receive data from more that one sensor, use the following:

URL	http://wotkit.sensetecnic.com/api/v1/data?{query-param}={query-value}&{param}={value}...
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of timestamped data records.

example

```
curl --user {username}:{password} http://wotkit.sensetecnic.com/api/v1/data?tags=vancouver
```

1.2.7 Sensor Control Channel: Actuators

An actuator is a sensor that uses a control channel to actuate things. Rather than POSTing data to the WoTKit, an actuator script or gateway polls the control URL for messages to affect the actuator, to do things like move a servo motor, turn a light on or off, or display a message on a screen. Any name/value pair can be sent to an actuator in a message.

For example, provided with the WoTKit at , a *control* widget that can be added to a dashboard (http://wotkit.sensetecnic.com/wotkit//dashboards) sends three types of events to the sensor control channel:

button	'on' or 'off' to control a light, or switch.
message	text message for use by the actuator, for example to be shown on a message board or display.
slider	a numeric value to affect the position of something, such as a server motor.

Sending Actuator Messages

To send a control message to a sensor (actuator), you must POST name value pairs corresponding to the data fields to the `/sensors/{sensorname}/message` URL.

URL	http://wotkit.sensetecnic.com/api/v1/sensors/{sensorname}/message
Privacy	Public or Private
Format	x-www-form-urlencoded
Method	POST
Returns	**OK 200** (no content) on success.

Receiving Actuator Messages

To receive actuator messages you must first subscribe to an Actuator Controller, then you can query for messages.

Note: In order to receive messages from an actuator, you must own that actuator.

Subscribing to an Actuator Controller

First, subscribe to the controller by POSTing to `/api/control/sub/{sensor-name}`. In return, we receive a json object containing a subscription id.

URL	http://wotkit.sensetecnic.com/api/v1/control/sub/{sensor-name}
Privacy	Private
Format	json
Method	POST
Returns	**200 OK** on success. A JSON object in the response body containing subscription id.

Example subscription id returned:

```
{
        "subscription":1234
}
```

Query an Actuator

Using the subscription id, then poll the following resource: `/api/control/sub/{subscription-id}?wait=10`. The `wait` parameter specifies the time to wait in seconds for a control message. If unspecified, a default wait time of 10 seconds is used. The maximum wait time is 20 seconds. The server will respond on timeout, or when a control messages is received.

URL	http://wotkit.sensetecnic.com/api/v1/control/sub/{subscription-id}?wait={wait-time}
Privacy	Private
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing control messages.

Note: Each subscription will be open for as long as the client that created it keeps sending long pull requests. A subscription that does not receive any requests after 5 minutes (3000 seconds) will be garbage-collected and will not be accessible after that. A client must catch this error and create a new subscription if this occurs.

To illustrate, the following code snippet uses HTTP client libraries to subscribe and get actuator messages from the server, and then print the data. Normally, the script would change the state of an actuator like a servo or a switch based on the message received.

```
# sample actuator code
import urllib
import urllib2
import base64
import httplib

try:
        import json
except ImportError:
        import simplejson as json

#note trailing slash to ensure .testactuator is not dropped as a file extension
actuator="mike.testactuator/"

# authentication setup
```

```
conn = httplib.HTTPConnection("wotkit.sensetecnic.com")
base64string = base64.encodestring('%s:%s' % ('{id}', '{password}'))[:-1]
authheader =  "Basic %s" % base64string
# In some clients (<Python 2.6) params must be used to force sending Content-Length header
# so, we'll use dummy params.
params = urllib.urlencode({'@type': 'subscription'})
headers = {'Authorization': authheader}

#subscribe to the controller and get the subscriber ID
conn.request("POST", "/api/v1/control/sub/" + actuator, params, headers=headers)
response = conn.getresponse()
data = response.read()

json_object = json.loads(data)
subId = json_object['subscription']

#loop to long poll for actuator messages
while 1:
        print "request started for subId: " + str(subId)
        conn.request("GET", "/api/v1/control/sub/" + str(subId) + "?wait=10", headers=headers)
        response = conn.getresponse()
        data = response.read()

        json_object = json.loads(data)

                # change state of actuator based on json message received
        print json_object
```

1.2.8 Tags

Sensors can specify several tags that can be used to organize them. You can get a list of tags, either the most used by public sensors or by a particular sensor query.

Querying Sensor Tags

A list of matching tags. The query parameters are as follows:

Name	Value Description
scope	**all**-all tags used by sensors that the current user has access to; | **subscribed**-tags for sensors the user has subscribed to; | **contributed**-tags for sensors the user has contributed to the system.
private	**DEPRECATED**, use visibility instead. (true - private sensors only; false - public only)
visibility	filter by the visibility of the sensors, either of **public**, **organization** or **private**
text	text to search in the sensors's name, long name and description
active	when true, only returns tags for sensors that have been updated in the last 15 minutes.
offset	offset into list of tags for paging
limit	limit to show for paging. The maximum number of tags to display is 1000.
location	**geo coordinates for a bounding box to search within.** Format is **yy.yyy,xx.xxx:yy.yyy,xx.xxx**, the order of the coordinates are North,West:South,East. Example: **location=56.89,-114.55:17.43,-106.219**

To query for tags, add query parameters after the tags URL as follows:

URL	http://wotkit.sensetecnic.com/api/v1/tags?{query}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of tag count objects matching the query.

To query for all tags that contain the text *bicycles* use the URL:

example

```
curl --user {id}:{password}
``http://wotkit.sensetecnic.com/api/v1/tags?text=bicycles``
```

Output:

```
[
        {
                'name': 'bicycle',
                'count': 3,
                'lastUsed': 1370887340845
        },{
                'name': 'bike',
                'count': 3,
                'lastUsed': 1350687440754
        },{
                'name': 'montreal',
                'count': 1,
                'lastUsed': 1365857340341
        }
]
```

The *lastUsed* field represents the creation date of the newest sensor that uses this tag.

1.2.9 Organizations

Organizations allow multiple users to create and manage shared sensors. All users can see organizations, but only admins can manipulate them.

List/Query Organizations

A list of matching organizations may be queried by the system. The optional query parameters are as follows:

Name	Value Description
text	text to search for in the name, long name and/or description
offset	offset into list of organizations for paging
limit	limit to show for paging. The maximum number of organizations to display is 1000.

To query for organizations, add query parameters after the sensors URL as follows:

URL	http://wotkit.sensetecnic.com/api/v1/orgs?{query}
Privacy	Public
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of organizations matching the query from newest to oldest.

Viewing a Single Organization

To view a single organization, query by name:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}
Privacy	Public
Format	json
Method	GET
Returns	**200 OK** if successful and a JSON on body describing the organization.

example

```
curl ``http://wotkit.sensetecnic.com/api/v1/orgs/electric-inc``
```

Output:

```
{
        "id": 4764,
        "name": "electric-inc",
        "longName": "Electric, Inc.",
        "description": "Electric, Inc. was established in 1970.",
        "imageUrl": "http://www.example.com/electric-inc-logo.png"
}
```

Creating/Registering an Organization

To register a new organization, you POST an organization resource to the url */org*. The organization resources is a JSON object with the following fields:

- The organization resources is a JSON object.
- The "name" and "longName" fields are **required** and must both be at least 4 characters long.
- The "imageUrl" and "description" fields are **optional**.

	Field Name	Information
(*REQUIRED*)	name	The name of the organization. Must be at least 4 characters long.
(*REQUIRED*)	longName	A descriptive name of the organization. Must be at least 4 characters long. Can be updated.
(*OPTIONAL*)	description	Description of the organization. Can be updated.
(*OPTIONAL*)	imageUrl	An image that often used in thumbnails to identify the organization. Can be updated.

To create an organization:

URL	http://wotkit.sensetecnic.com/api/v1/orgs
Privacy	Admin
Format	json
Method	POST
Returns	**201 Created** if successful; **Bad Request 400** if organization is invalid; **Conflict 409** if an organization with the same name already exists.

Updating an Organization

- You may update any fields except "id" and "name".
- Only fields that are present in the JSON object will be updated.

To update an organization:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}
Privacy	Admin
Format	json
Method	PUT
Returns	**200 OK** if successful. No content on body.

Deleting an Organization

Deleting an organization is done by deleting the organization resource.

To delete a user:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}
Privacy	Admin
Format	n/a
Method	DELETE
Returns	**200 OK** if successful. No content on body.

Organization Membership

List all members of an Organization

To query for organization members:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}/members
Privacy	Admin
Format	n/a
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of organization members.

Add new members to an Organization

To add new members to an organization, post a JSON array of usernames:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}/members
Privacy	Admin
Format	json
Method	POST
Returns	**204 No Content** on success.

Usernames that are already members, or usernames that do not exist, will be ignored.

For instance, to add the users “abe”, “beth”, “cecilia” and “dylan” to the organization “electric-inc”:

example

```
curl --user {id}:{password} --request POST
--header ``Content-Type: application/json'' --data-binary @users-list.txt
`http://wotkit.sensetecnic.com/api/v1/orgs/electric-inc/members`
```

The file *users-list.txt* would contain the following.

```
["abe", "beth", "cecilia", "dylan"]
```

Remove members from an Organization

To remove members from an organization, DELETE a JSON array of usernames:

URL	http://wotkit.sensetecnic.com/api/v1/orgs/{org-name}/members
Privacy	Admin
Format	json
Method	DELETE
Returns	**204 No Content** on success. A JSON object in the response body containing a list of usernames.

Usernames that are not members, or usernames that do not exist, will be ignored.

1.2.10 Sensor Groups

Sensor Groups are used to logically organize related sensors. Any sensor can be a member of many groups.

Currently, all Sensor Groups have **private** visibility, and **only** the **owner** (creator) can add/remove sensors from the group, or make a group **public**.

Sensor Groups can be manipulated using a REST API in the following section

Sensor Group Format

All request body and response bodies use JSON. The following fields are present:

	Field Name	Type	Notes
(*REQUIRED*)	**id**	Integer	The id contains a unique number which is used to identify the group
(*REQUIRED*)	**name**	String[4,50]	The name is a system-unique string identifier for the group. Names must be lowercase containing alphanumeric, underscores or hyphens `[a-z0-9_-]`. The first character **must** be an alphabetic character
(*REQUIRED*)	**longName**	String[4,100]	A readable name used for visual interfaces. It must be at least 4 characters long.
(*READ-ONLY*)	**owner**	String[4,50]	The name of the group's owner. This field is set by the system and cannot be modified.
(*REQUIRED*)	**description**	String[,255]	A simple description of the group
(*OPTIONAL*)	**imageUrl**	String[,255]	A string url to an image which can be used to represent this group
(*OPTIONAL*)	**sensors**	Array[Sensor]	Contains a JSON list of sensors. This field is only useful for viewing sensors. To append/remove sensors from Sensor Groups, refer to *Adding a Sensor to Sensor Group*.

An example of a Sensor Group JSON would be as follows:

```
{
  "id": 602,
  "name": "test",
  "longName": "test",
  "description": "test",
  "tags": [
    "group",
    "test"
  ],
  "imageUrl": "",
  "latitude": 49.25,
  "longitude": -123.1,
  "visibility": "PUBLIC",
  "owner": "sensetecnic",
  "lastUpdate": "1970-01-01T00:00:00.000Z",
  "created": "2014-03-27T23:29:51.479Z",
  "metadata": {
    "meta": "data"
  },
  "childCount": 0,
  "things": [],
  "thingType": "GROUP"
}
```

List Groups

Provides a list of all PUBLIC groups on the system as an array using the JSON format specified in *Sensor Group Format*

URL	http://wotkit.sensetecnic.com/api/v1/groups/
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successul. A JSON object in the response body containing a list of groups.

example

```
curl --user {id}:{password} --request GET `http://wotkit.sensetecnic.com/api/v1/groups`
```

Viewing a Single Sensor Group

Similar to *List Groups*, but will retrieve only a single sensor group. Replace `{group-name}` with the group's `{id}` integer or `{owner}.{name}` string. The API accepts both formats

URL	http://wotkit.sensetecnic.com/api/v1/groups/{group-name}
Privacy	Public or Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body describing the sensor group.

example

```
curl --user {id}:{password} --request GET `http://wotkit.sensetecnic.com/api/v1/groups/sens
```

Creating a Sensor Group

To create a sensor group, append the Sensor Group contents following *Sensor Group Format*.

On creation, the **id** is **ignored** because it is system generated. You should not provide an **owner** as it will be generated by the system to match the credentials used to call the API. Only if you are an administrator user you will be able to provide an **owner**.

URL	http://wotkit.sensetecnic.com/api/v1/groups
Privacy	Private
Format	json
Method	POST
Returns	**201 Created** if successful; **409 Conflict** if a sensor with the same name exists.

Modifying Sensor Group Fields

Modifying is similar to creation, the content is placed in the response body

Again, the **id** and **owner** fields in the JSON object are **ignored** if they are modified. The Sensor Group is specified by substituting `{group-name}` group's `{id}` integer or `{owner}.{name}` string. The API accepts both formats.

URL	http://wotkit.sensetecnic.com/api/v1/groups/{group-name}
Privacy	Private
Format	json
Method	PUT
Returns	**204 No Content** if successful; **401 Unauthorized** if user has no permissions to edit group.

Deleting a Sensor Group

Deleting a Sensor Group is fairly trivial, assuming you are the owner of the group. A response body is unnecessary.

URL	http://wotkit.sensetecnic.com/api/v1/groups/{group-name}
Privacy	Private
Format	json
Method	DELETE
Returns	**204 No Content** if successful; **401 Unauthorized** if user has no permissions to edit group.

Adding a Sensor to Sensor Group

This is done by invoking the URL by replacing the specified parameters where `{group-name}` can be the group's `{id}` integer or `{owner}.{name}` string. `{sensor-id}` should be the sensor's `id` integer.

URL	http://wotkit.sensetecnic.com/api/v1/groups/{group-name}/sensors/{sensor-id}
Privacy	Private
Format	json
Method	POST
Returns	**204 No Content** if successful; **400** if sensor is already a member of sensor group; **401 Unauthorized** if user is unauthorized to edit group.

Removing a Sensor from Sensor Group

The format is the same as *Adding a Sensor to Sensor Group* except replacing `method` with `DELETE`. Replace `{sensor-id}` with the sensor's `{id}` integer.

URL	http://wotkit.sensetecnic.com/api/v1/groups/{group-name}/sensors/{sensor-id}
Privacy	Private
Format	n/a
Method	DELETE
Returns	**204 No Content** if successful; **401 Unauthorized** if user is unauthorized to edit group.

1.2.11 News

The "news" resource provides a list of interesting and recent activities in the WoTKit.

URL	http://wotkit.sensetecnic.com/api/v1/news
Privacy	Public
Format	n/a
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing a list of news items.

example

curl ``http://wotkit.sensetecnic.com/api/v1/news``

Output:

```
[{
        'timestamp': 1370910428123,
        'title': u'The sensor "Light Sensor" has updated data.',
        'url': u'/sensors/5/monitor'
},{
        'timestamp': 1370910428855,
        'title': u'The sensor "api-data-test-1" has updated data.',
        'url': u'/sensors/40/monitor'
}]
```

1.2.12 Statistics

The "stats" resource provides statistics, like number of public sensors, active sensors, or new sensors. It can be accessed via:

URL	http://wotkit.sensetecnic.com/api/v1/stats
Privacy	Public
Format	not applicable
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing describing statistics.

example

curl ``http://wotkit.sensetecnic.com/api/v1/stats``

Output:

```
{
        'total': 65437,
        'active': 43474,
        'new': {
                'day': 53,
                'week': 457,
                'month': 9123,
                'year': 40532
        }
}
```

1.3 V2 API Reference

Please be advised that V2 is in beta and not ready for production.

1.3.1 Sensor Data

In the WoTKit, *sensor data* consists of a timestamp followed by one or more named fields. There are a number of reserved fields supported by the WoTKit:

Reserved field name	Description
timestamp	the time that the sensor data was collected. This is a long integer representing the number of milliseconds from Jan 1, 1970 UTC. Optional; if not supplied, a server-supplied timestamp will be used.
id	a unique identifier for the data reading. This is to distinguish one reading from another when they share the same timestamp. **Read only**; This field is read only and should not be sent by the client when sending new data.
sensor_id	the globally unique sensor id that produced the data. **Read only**; This is a read only field generated by the wotkit that should not be sent by a client when sending new data.
sensor_name	the globally unique sensor name, in the form `{username}.{sensorname}`. **Read only**; This is a read only field and should not be sent by the client when sending new data.

When a new sensor is created, a number of default fields are created by the wotkit for a sensor as follows. Note that these can be changed by editing the sensor fields.

Default field name	Description
lat	the current latitude location of the sensor in degrees (number). Needed for map visualizations.
lng	the current longitude location of the sensor in degrees (number). Needed for map visualizations.
value	the primary value of the sensor data collected (number). Needed for most visualizations.
message	a text message, for example a twitter message (text). Needed for text/newsfeed visualizations.

In addition to these default fields, additional fields can be added by updating the *sensor fields* in the WoTKit UI or *Sensor Fields* in the API.

Note: Python's `time.time()` function generates the system time in *seconds*, not milliseconds. To convert this to an integer in milliseconds use `int(time.time()*1000)`.

In Javascript: `var d = new Date(); d.getTime();`

In Java: `System.currentTime()`.

Sending New Data

To send new data to a sensor, POST name value pairs corresponding to the data fields to `/sensors/{sensorname}/data`. There is no need to supply the sensor id, or sensor name fields since the sensor is specified in the URL.

If a timestamp is not provided in the request body, it will be set to the current time by the the server.

To send new data:

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensorname}/data
Privacy	Private
Format	not applicable
Method	POST
Returns	**201 Created** if successful.

Example

```
curl --user {id}:{password} --request POST -d value=5 -d lng=6 -d lat=7
`http://wotkit.sensetecnic.com/api/v2/sensors/test-sensor/data`
```

Updating a Range of Historical Data

To insert or update a range of historical data, you PUT data (rather than POST) data into the system. Note that data PUT into the WoTKit will not be processed in real time, since it occurred in the past. Thus, a timestamp field is required.

- The request body must be a list of JSON objects, as specified in *Sensor Data*. In the case of updating existent data is that each objet MUST contain a timestamp value which will be updated.

Note: Any existing data matching the provided timestamp be deleted and replaced by the data supplied.

To update data:

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensorname}/data
Privacy	Private
Format	JSON
Method	PUT
Returns	**204 No Content** if successful. **400 Bad Request** if unsuccessful.

Example of valid data:

```
[{"timestamp":"2012-12-12T03:34:28.626Z","value":67.0,"lng":-123.1404,"lat":49.20532},
{"timestamp":"2012-12-12T03:34:28.665Z","value":63.0,"lng":-123.14054,"lat":49.20554},
{"timestamp":"2012-12-12T03:34:31.621Z","value":52.0,"lng":-123.14063,"lat":49.20559},
{"timestamp":"2012-12-12T03:34:35.121Z","value":68.0,"lng":-123.14057,"lat":49.20716},
{"timestamp":"2012-12-12T03:34:38.625Z","value":51.0,"lng":-123.14049,"lat":49.20757},
{"timestamp":"2012-12-12T03:34:42.126Z","value":55.0,"lng":-123.14044,"lat":49.20854},
{"timestamp":"2012-12-12T03:34:45.621Z","value":56.0,"lng":-123.14215,"lat":49.20855},
{"timestamp":"2012-12-12T03:34:49.122Z","value":55.0,"lng":-123.14727,"lat":49.20862},
{"timestamp":"2012-12-12T03:34:52.619Z","value":59.0,"lng":-123.14765,"lat":49.20868}]
```

example

```
curl --user {id}:{password} --request PUT --data-binary @data.txt
`http://wotkit.sensetecnic.com/api/v2/sensors/test-sensor/data`
```

where *data.txt* contains JSON data similar to the above JSON array.

Retrieving a Single Data Item

If you know the data element's id, you can query for a single data element using the following query.

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensor-name}/data/{data_id}
Privacy	Public or Private, depending on sensor privacy
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of timestamped data records.

Retrieving Data Using Query

To retrive sensor data over a time range you can use the following endpoint. An interactive guide on how to use this endpoint is available at: Querying Sensor Data.

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensor-name}/data
Privacy	Public or Private, depending on sensor privacy
Format	json
Method	GET
Returns	**200 OK** on success. A JSON object in the response body containing a list of timestamped data records.

The query parameters supported are the following. They can only be used together if they appear in the same *Group* below.

Parameter	Group	Type	Description
`recent_t`	1	integer	Gets the elements up to recent_t milliseconds ago
`recent_n`	2	integer	Gets the n recent elements
`start`	3	timestamp	The absolute starting point (in milliseconds since Jan 1, 1970).
`start_id`	3	id	The starting id of sensor_data at timestamp `start`. Used for paging and to distinguish data elements that share the same timestamp.
`end`	3	timestamp	The absolute ending timestamp (in milliseconds since Jan 1, 1970)
`end_id`	3	timestamp	The end id of sensor_data with timestamp `end`. Used for paging.
`limit`	[2,3]	integer	specifies how many datapoints to see on each response

Delete Data by Id

Same as api-v2-get-single-data instead using HTTP Delete.

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensorname}/data/{data_id}
Privacy	Private
Format	not applicable
Method	DELETE
Returns	**204 No Content** if successful.

Delete Data using Data Query

Can delete using query parameters in *Retrieving Data Using Query* with the restriction on only using **group 3** parameters.

URL	http://wotkit.sensetecnic.com/api/v2/sensors/{sensorname}/data
Privacy	Private
Format	not applicable
Method	DELETE
Returns	**204 No Content** if successful.

Sending Aggregated Data

One can send aggregated data from multiple sensors from an organization/owner. Each data point MUST have at least the following fields: "sensor_name" (without owner), "timestamp", and other fields marked as "required" by each individual sensor.

To publish data from more that one sensor, use the following:

URL	http://wotkit.sensetecnic.com/api/v2/data/{owner}
Privacy	Private
Format	json
Method	POST
Returns	**204 No Content** if successful

Example of valid data:

```
[{"sensor_name": "{sensorname}", "value":12.0,"lng":-123.1404,"lat":49.20532,"timestamp":1441068278},
{"sensor_name": "{sensorname}", "value":67.0,"lng":-123.1404,"lat":49.20532,"timestamp":1441068278},
{"sensor_name": "{sensorname}", "value":1.0,"lng":-123.1404,"lat":49.20532,"timestamp":1441068278}]
```

example

```
curl --user {username}:{password} --request POST -H ``Content-Type: application/json'' --da
```

where *data.txt* contains JSON data similar to the above JSON array.

1.3.2 Alerts

An alert is set up by an user for notification purpose. Multiple conditions can be attached to an alert. Each condition is associated with a sensor field. An alert fires and sends a message to the owner's inbox and email (if email functionality is enabled) when all of its attached conditions are satisfied. Currently, each user is limited to have a maximum of 20 alerts.

An alert has the following attributes:

Name	Value Description
id	the numeric id of the alert. It is automatically assigned when alert is created.
name **	the name of the alert.
longName	longer display name of the alert.
description	a description of the alert.
owner	the user that creates the alert. The value of this field is automatically assigned as a user creates an alert.
disabled	**the on/off state of the alert.** - If 'disabled' is 'true', the alert is switched off; it switches on if otherwise.
inProgress	**whether conditions are still true after an alert has fired.** - inProgress is 'true' if all alert conditions remain true after an alert has fired. It becomes 'false' when any condition turns false. An alert gets fired when its inProgress state changes from false to true.
template	The message template that is sent to the inbox when alert is fired. You can use ${alertName}, ${id}, ${description} and ${user} to compose a message.
email	The email the alert is sent to.
sendEmail	A boolean to enable/disable send email functionaity.
conditions	the list of alert conditions

** Required when creating a new alert.

An alert condition is composed of a sensor field, an operator for evaluation, and a value. It has the following attributes:

Name	Value Description
sensorId	the ID of the sensor associated with the condition
field	the field name to be compared of the chosen sensor
operator	**the conditon operator, its value can be one of following:** 'LT': Less Than 'LE': Less Than Or Equal To 'GT': Greater Than 'GE': Greater Than Or Equal To 'EQ': Equal 'NEQ': Not Equal 'NULL': Is Null 'NOTNULL': Is Not Null
value	value that the operator compares with

Listing Alerts of an User

To view a list of "alerts" created by an user:

URL	http://wotkit.sensetecnic.com/api/v2/alerts
Privacy	Private
Format	JSON
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing a list of alerts.

example

```
curl --user {id}:{password} ``http://wotkit.sensetecnic.com/api/v2/alerts``
```

Sample Output:

```
    [{
    "id": 6,
    "owner": "crysng",
    "name": "temperature-alert",
    "longName": "Temperature Alert",
    "description": "This alert notifies user when Hydrogen Sulfide content and Wind speed is too high
    "disabled": false,
    "inProgress": false,
    "template": "Hydrogen Sulfide and wind speed is high!",
    "sendEmail": true,
    "email": "rottencherries@hotmail.com",
    "conditions": [
        {
            "sensorId": 241,
            "field": "h2s",
            "operator": "GT",
            "value": 10
        },
        {
            "sensorId": 241,
            "field": "wspd",
            "operator": "GE",
            "value": 50
        }
    ]
},
{
    "id": 5,
    "owner": "crysng",
    "name": "test",
    "longName": "Moisture Sensor Alert",
    "description": "This alert fires when moisture level is too low. ",
    "disabled": false,
    "inProgress": false,
    "template": "Moisture level is too low, water the plant now!",
    "sendEmail": true,
    "email": "someone@email.com",
    "conditions": [
        {
            "sensorId": 504,
            "field": "value",
            "operator": "LT",
            "value": 3
        }
    ]
}]
```

Viewing an Alert

To view an alert, query the alert by its id as followed:

URL	http://wotkit.sensetecnic.com/api/v2/alerts/{alert id}
Privacy	Private
Format	json
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body describing an alert.

example

`curl --user {id}:{password}`
``` ``http://wotkit.sensetecnic.com/api/v2/alerts/5`` ```

Output:

```
    {
    "id": 5,
    "owner": "crysng",
    "name": "test",
    "longName": "Moisture Sensor Alert",
    "description": "This alert fires when moisture level is too low. ",
    "disabled": false,
    "inProgress": false,
    "template": "Moisture level is too low, water the plant now!",
    "sendEmail": true,
    "email": "someone@email.com",
    "conditions": [
        {
            "sensorId": 504,
            "field": "value",
            "operator": "LT",
            "value": 3
        }
    ]
}
```

Creating Alerts

The alert resource is a JSON object. To create an alert you POST a sensor resource to the url `/v2/alerts`.

To create an alert:
```

URL	http://wotkit.sensetecnic.com/api/v2/alerts
Privacy	Private
Format	JSON
Method	POST
Returns	**200 OK** if successful; **400 Bad Request** if sensor is invalid; **409 Conflict** if alert with the same name already exists.

THE JSON object has the following fields:

	Field Name	Information
(*REQUIRED*)	name	The unique name for the alert. It must be at least 4 characters long, contain only lowercase letters, numbers, dashes and underscores, and can start with a lowercase letter or an underscore only.
(*OPTIONAL*)	longName	longer display name of the alert.
(*OPTIONAL*)	description	a description of the alert.
(*OPTIONAL*)	disabled	**the on/off state of the alert.** - If ‘disabled’ is ‘true’, the alert is switched off; it switches on if otherwise.
(*OPTIONAL*)	template	The message template that is sent to the inbox when alert is fired. You can use ${alertName}, ${id}, ${description} or ${user} to compose a message, e.g. “Alert by ${user} fired”
(*OPTIONAL*)	email	The email the alert is sent to. It defaults to the owner’s email.
(*OPTIONAL*)	sendEmail	A boolean to enable/disable send email functionaity.
(*OPTIONAL*)	conditions	The list of alert conditions

example1

```
curl --user {id}:{password} --request POST --header ``Content-Type: application/json''
--data-binary @test-alert.txt `http://wotkit.sensetecnic.com/api/v2/alerts`
```

For this example, the file *test-alert.txt* contains the following. This is the minimal information needed to create an alert.

```
{
        "name":"test alert",
        "description":"A test alert.",
        "template":"Template for test alert using any of ${alertName}, ${id}, ${description} or ${use
        "sendEmail":false
}
```

example2

Now, let's create an alert with additional information and conditions. The file *test-alert.txt* contains the following.

```
{
        "name": "test alert 2",
        "longName": "Test Alert 2",
        "description": "This is test 2. ",
        "disabled": false,
        "template": "The alert ${alertName} has fired!! ",
        "sendEmail": true,
        "email": "someone@email.com",
        "conditions": [
        {
                "sensorId": 504,
                "field": "value",
                "operator": "LT",
                "value": 3
        },
        {
                "sensorId": 24,
                "field": "data",
                "operator": "NOTNULL"
        }
        ]
}
```

Updating Alerts

Updating an alert is the same as creating a new alert other than PUT is used and the alert id is included in the URL.

Note that all top level fields supplied will be updated.

- You may update any fields except "id", and "owner".
- Only fields that are present in the JSON object will be updated.

To update an alert owned by the current user:

URL	http://wotkit.sensetecnic.com/api/v2/v2/alerts/{alert id}
Privacy	Private
Format	JSON
Method	PUT
Returns	**200 OK** if successful.

For instance, to update an alert:

example

```
curl --user {id}:{password} --request PUT --header ``Content-Type: application/json''
--data-binary @update-alert.txt `http://wotkit.sensetecnic.com/api/v2/alerts/{alert id}`
```

The file *update-alert.txt* would contain the following:

```
{
        "longName": "New Alert Name",
        "description":"Updated Description"
}
```

Deleting Alerts

Deleting an alert is done by deleting the alert resource.

To delete an alert owned by the current user:

URL	http://wotkit.sensetecnic.com/api/v2/alerts/{alert id}
Privacy	Private
Format	not applicable
Method	DELETE
Returns	**204 No Response** if successful.

example

```
curl --user {id}:{password} --request DELETE
`http://wotkit.sensetecnic.com/api/v2/alerts/{alert id}`
```

1.3.3 Inbox

The Inbox is the storage place for inbox messages that are sent by an alert firing event.

An inbox message has the following attributes:

Name	Value Description
id	the numeric id of the message. It is automatically generated.
timestamp	the time that the message is sent to inbox.
title	title of the inbox message
message	the message content
sendEmail	the boolean variable of whether email functionality is enabled
read	the flag of whether the message is read
sent	the flag of whether an email is sent

Listing Inbox Messages of an User

To view a list of "inbox messages" of an user:

URL	http://wotkit.sensetecnic.com/api/v2/inbox
Privacy	Private
Format	JSON
Method	GET
Returns	**200 OK** if successful. A JSON object in the response body containing a list of messages.

example

`curl --user {id}:{password} ``http://wotkit.sensetecnic.com/api/v2/inbox```

Sample Inbox Messages Output:

```
[
        {
        "id": 5,
        "timestamp": "2014-04-17T00:51:41.701Z",
        "title": "Moisture Sensor Alert",
        "message": "Moisture level is too low, water the plant now!",
        "sendEmail": true,
        "email": "someone@email.com",
        "read": false,
        "sent": false
        }
]
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search