WoTKit

Release 1.3.0

Sensetecnic

October 17, 2013

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:

- *User Documentation*
- *API Documentation*

Note: This documentation is work in progress. Please send questions and feedback to info@sensetecnic.com.

CHAPTER

ONE

QUICK START

1.1 Overview

The WoTKit lets you quickly publish, find and use interesting data streams in quick visualizations and your own applications; from environmental sensors, GPS and on board data collection from vehicles, real time data feeds from mobile applications, building sensors, and internet-sourced content. With the WoTKit you can easily add visualizations for display on a WoTKit dashboard and create applications using the WoTKit API.

This quick start tutorial will get you started with the WoTKit.

1.2 Finding Sensors

The WoTKit hosts many interesting sensor streams. Some sensors on the system represent physical sensors and actuators such as temperature and light sensors connected to zigbee radios, or servo motors used to control things. Other sensors host data pulled from web sites and external sensor systems such as the power use of buildings or ferry locations.

To find interesting public sensors, you need not create an account. Simply hit the Sensor Search page and use the UI there to find sensors. The sensor search text box allows you to search by sensor name and description. Click on tags to find sensors that use the selected tags.

1.3 Viewing a Sensor

To sensor details and data click on the sensor in the list link in the map view in the sensor search page. This will bring up a monitor page where you get information about the sensors such as its name, contributor, location, a table containing the data stream. See the Yellow Taxi for example.

To do more with the WoTKit, you'll need to create an account!

1.4 Create an Account

To create your own sensors or add visualizations to your own dashboards, you'll need to create an account. To do so, click on the log in button on the top right, then click on the *Create an Account* button. Fill in the form and log in to the WoTKit.

1.5 Create a Widget

Now that we're logged in, lets create a widget that displays sensor data on a dashboard using a line chart.

1. First, choose a sensor that you would like to visualize using the sensor seaarch page.
2. Type in 'light' in the search area. Click on the sensor called 'Light Sensor' published by Sense Tecnic.
3. In the sensor monitor view, click on the *Visualizations and Widgets* tab on the lower half of the screen to view available visualizations for the sensor. Lets select the visualization we want. Feel free to try out available visualizations.
4. Lets go with the *Line Graph* in the pop up. Click on the *Create this Widget* button to create a widget.

The widget will appear in the widget list. To add it to your default dashboard, click on the *Add to Dashboard* button beside the widget.

The Widget will appear on your dashboard. Feel free to move and resize the widget where you like.

1.6 Adding your own Sensor

To add your own sensors to the WoTkit, you will first use the UI to create a sensor, create a key to generate credentials for your sensor script to send data using the WoTKit API, then run your script to send data to the WoTKit.

1. Create a sensor by clicking on the Sensors tab in the navigation bar to take you to the Sensor Search page. Click on the *New Sensor* button in the top right.
2. Fill in the new sensor form. Lets call it 'Test Sensor' with the name 'test-sensor'. Click on the map to set a location for your sensor.
3. Once you've filled in the form, you can view the monitor page for that sensor.

At this point you've created a resource on the wotkit for your sensor. Now it is time to create a key to use in your sensor scripts to send data to the WoTKit using the API.

1. Create an API key by clicking on the Keys button in the navigation bar to take you to the Keys page.
2. Click on the *New Key* button in the top right.
3. Fill in the new key form. Lets call the key a 'Test Key' since we'll only use it for our test sensors.

Now that we've created a sensor resource and a key, lets write a script to send data to our sensor. Lets start with something simple like sending a random value to the sensor using Python.

Here's the code:

```
import random
import time
import datetime
import urllib
import urllib2
import base64

KEY_ID = 'PASTE_YOUR_KEY_ID_HERE'
KEY_PASS = 'PASTE_YOUR_KEY_PASSWORD_HERE'

if __name__ == '__main__':

    random.seed(time.time())
```

```
# encode our key id and password
base64string = base64.encodestring('%s:%s' % (KEY_ID, KEY_PASS))[:-1]

# the URL for our sensor
url = 'http://wotkit.sensetecnic.com/api/sensors/test-sensor/data'

while 1:

    # get value from the sensor, in this case we'll just generate a random number
    value = random.randint(0,100)

    datafields = [('value','%d' % value)]

    params = urllib.urlencode(datafields)

    headers = {
        'User-Agent': 'httplib',
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': "Basic %s" % base64string
    }

    req = urllib2.Request(url,params,headers)
    try:
        result = urllib2.urlopen(req)

    except urllib2.URLError, e:
        print "error", e


    print 'random value sent: %d' % (value)

    time.sleep(2.0)
```

Be sure to paste your generated key id and password into the variables above and make sure the sensor name is the one you chose for your sensor in the URL (we suggested 'test-sensor').

Now if all goes well, the script will send a random value to the wotkit every 2 seconds. View the monitor page to see the new data added to the data table below in near real time. Click on the 'Visualizations and Widgets' tab to visualize the data with line charts and graphs.

1.7 Where to go from here

Consult the *User Documentation* for more information on using the WoTKit portal.

To create your own WoTKit applications, register sensors dynamically and take advantage of the WoTKit platform with your own applications, consult the *WoTKit API documentation*.

CHAPTER

TWO

USER DOCUMENTATION

2.1 Overview

With the WoTKit user interface, you can easily complete a wide range of tasks including registering sensors, subscribing to sensor feeds, and visualizing sensor data.

For information about the API including sending and receiving data from a sensor or an actuator, please see the *API Documentation*.

2.1.1 Public Sensors

The WoTKit allows you to view public sensors and their data without the need to create an account. Without logging in, click on the *Sensors* tab on the top navigation bar to view the list of public WoTKit sensors.

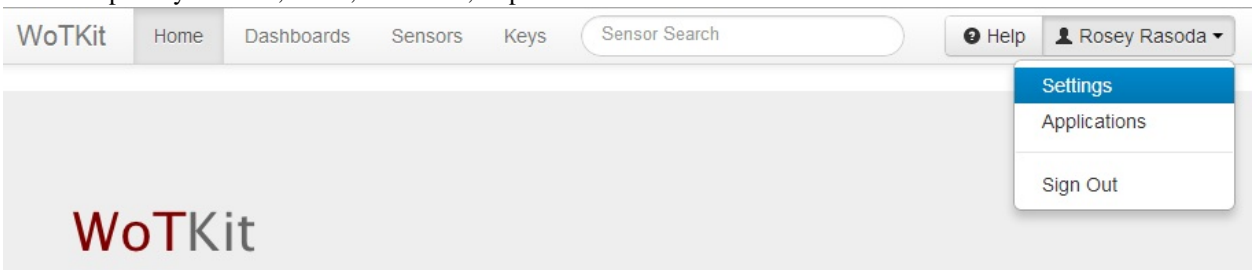
2.1.2 Creating an Account

To use other features of the WoTKit, you must create an account. To do so, click on the 'Log In' button on the top left, then click on the 'Create an Account' button in the log in page.

2.2 User Information

To edit your user profile information, select the user menu (located on the top right) and choose the *Settings* options.

You can update you name, email, time zone, or password there.



2.3 Sensors

To view the sensor gallery, click on the *Sensors* tab.

All available sensors are listed here, with the ability to page through the list and perform queries on the available sensors, filter based on tags, organizations and visibility.

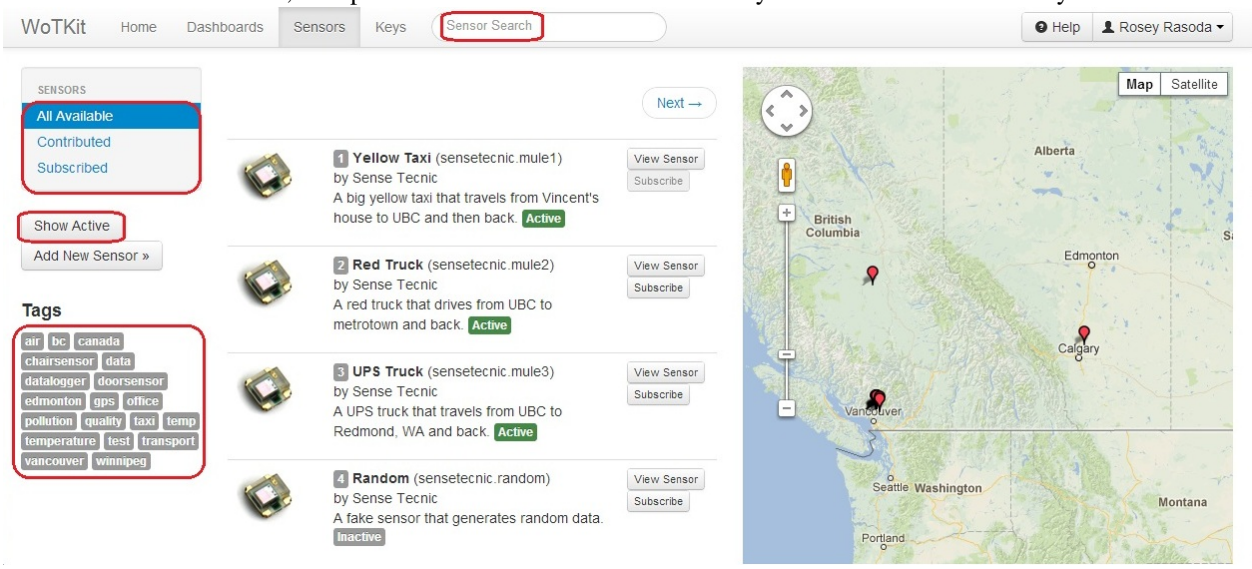
From there, there are three additional list views:

- **All Available**: display all of the sensors you can view on the WoTKit including public sensors, sensors that are in organizations you are a member of, and any private sensors you've contributed.
- **Subscribed**: showing all sensors to which the user has subscribed.
- **Contributed**: showing all sensors the user has added to the WoTKit.

These list views can be further narrowed as follows: * Using the search bar at the top of the screen. This searches by text contained in the sensor name and description.

- Selecting a tag from the tag list displayed on the left of the screen.
- Selecting the *Show Active* button displayed on the left of the screen. Active sensors are those which have received data in the last 15 minutes.

In addition to this list view, a map view is available that is automatically centered on the sensors in your list view.



2.3.1 Registering a New Sensor

To add a new sensor, click *Add New Sensor* button.

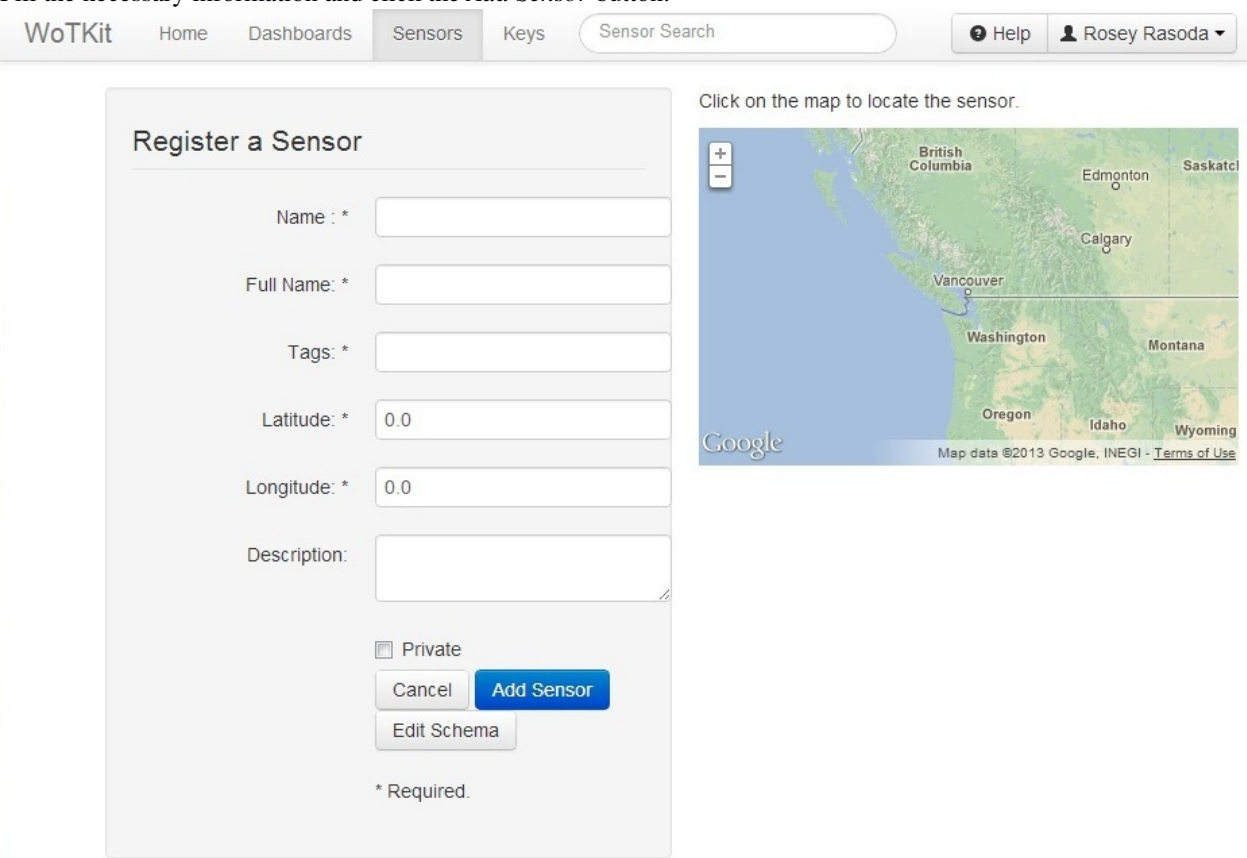
From here, you can provide basic information about your sensor, including:

- **Name**
  - Unique URL-friendly name for the sensor.
- **Full Name**
  - Name for the sensor shown in various lists and views.
- **Tags**
  - Tags for the sensor separated by commas.
- **Description**
  - A description for the sensor.

- **Latitude & Longitude**
  - The static location of the sensor entered manually or by using the provided map.
- **Visibility**
  - A private sensor cannot be viewed by other users.
  - The default is for a sensor to be public.

All fields except Description and Visibility are required.

Fill the necessary information and click the *Add Sensor* button.



2.3.2 Monitoring Sensors

The sensor monitor view privides details about the sensor, recent data sent to a sensor, and a way to visualize sensor data.

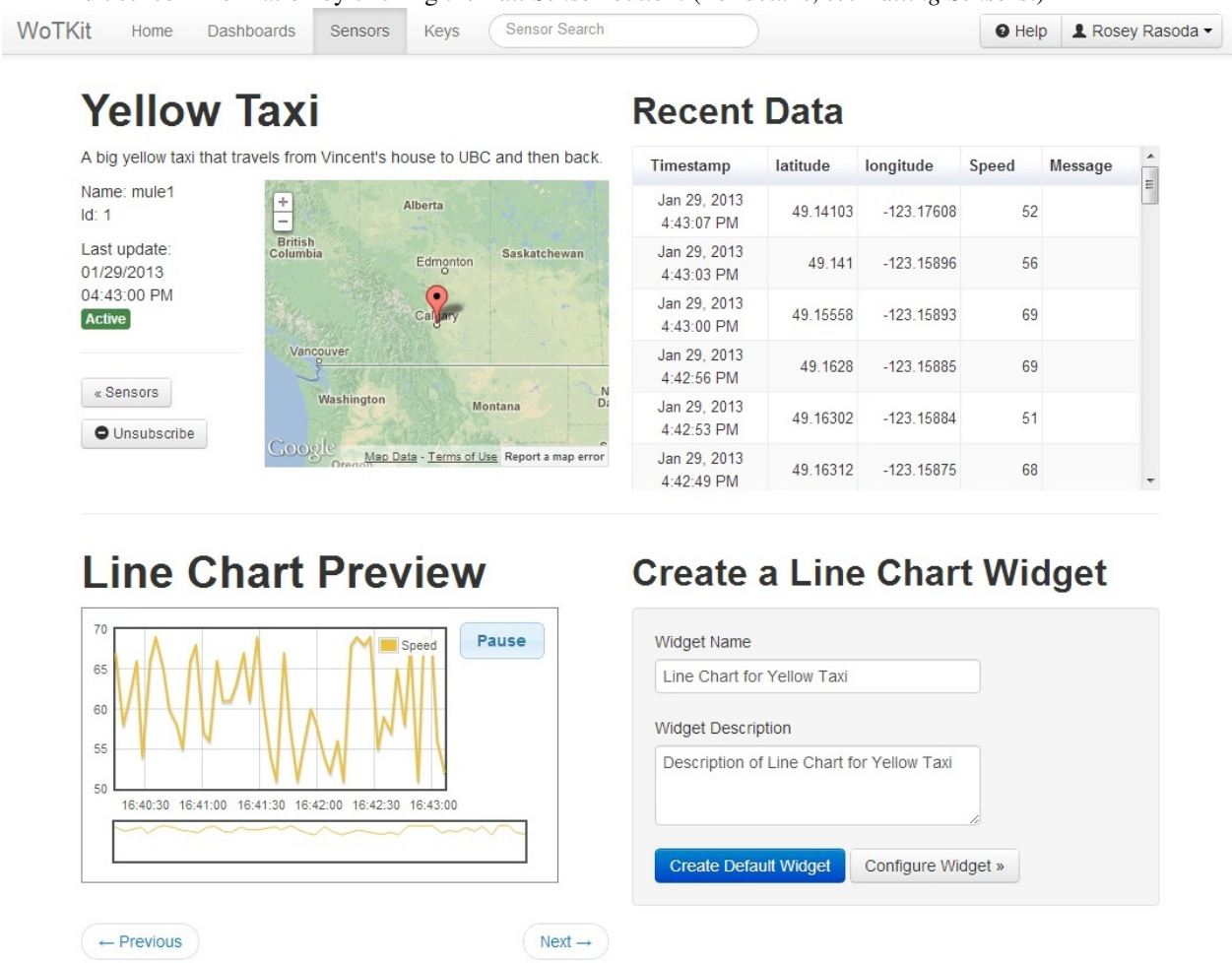
To monitor a sensor, select the *Sensors* tab, find the sensor you want to monitor, and click on the sensor.

This page allows you to:

- *Subscribe* or *Unsubscribe* from this sensor data feed by clicking the corresponding button.
- View sensor data using a visualization for your sensor data.
- Customize and create a new dashboard widget. (For details, see see *Creating a Widget* and *Adding Widgets to a Dashboard*.)

If you contributed the sensor to the WoTKit, you can also:

- Delete the sensor by clicking the *Delete* button.
- Edit sensor information by clicking the *Edit Sensor* button. (For details, see *Editing Sensors*.)

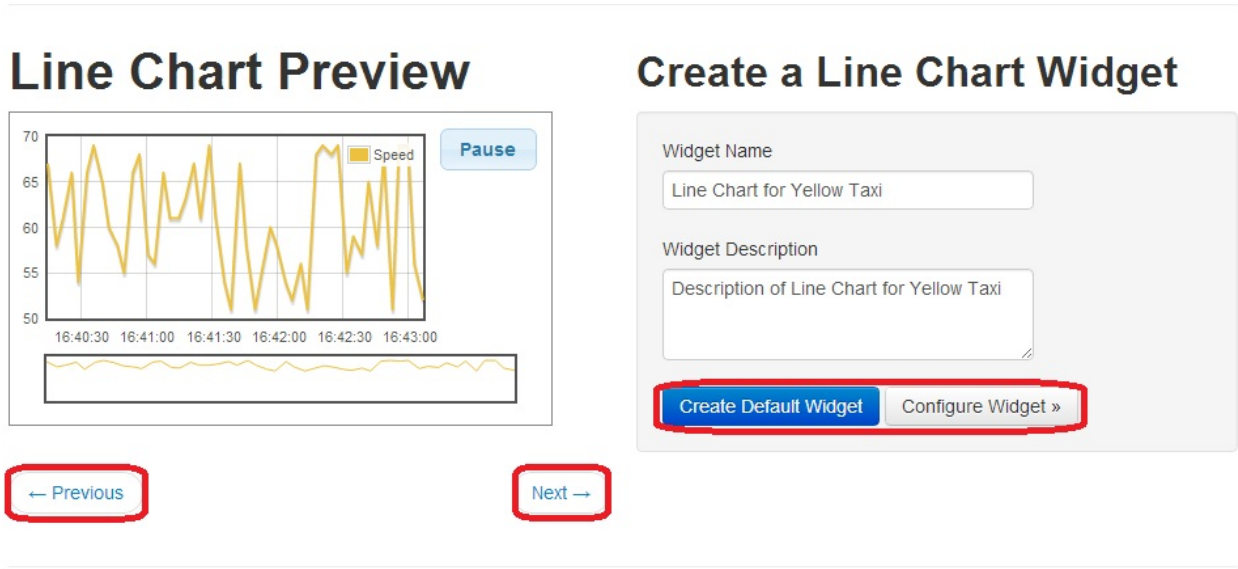


2.3.3 Creating a Widget Visualization

To create a widget for a sensor:

- **On the *Monitor View*, choose a widget.**
  - Using the *Previous* and *Next* buttons to find the correct visualization.
- **Create the widget using the *Create Default Widget* or *Configure Widget* button.**
  - Provide as much information for the widget as you would like.

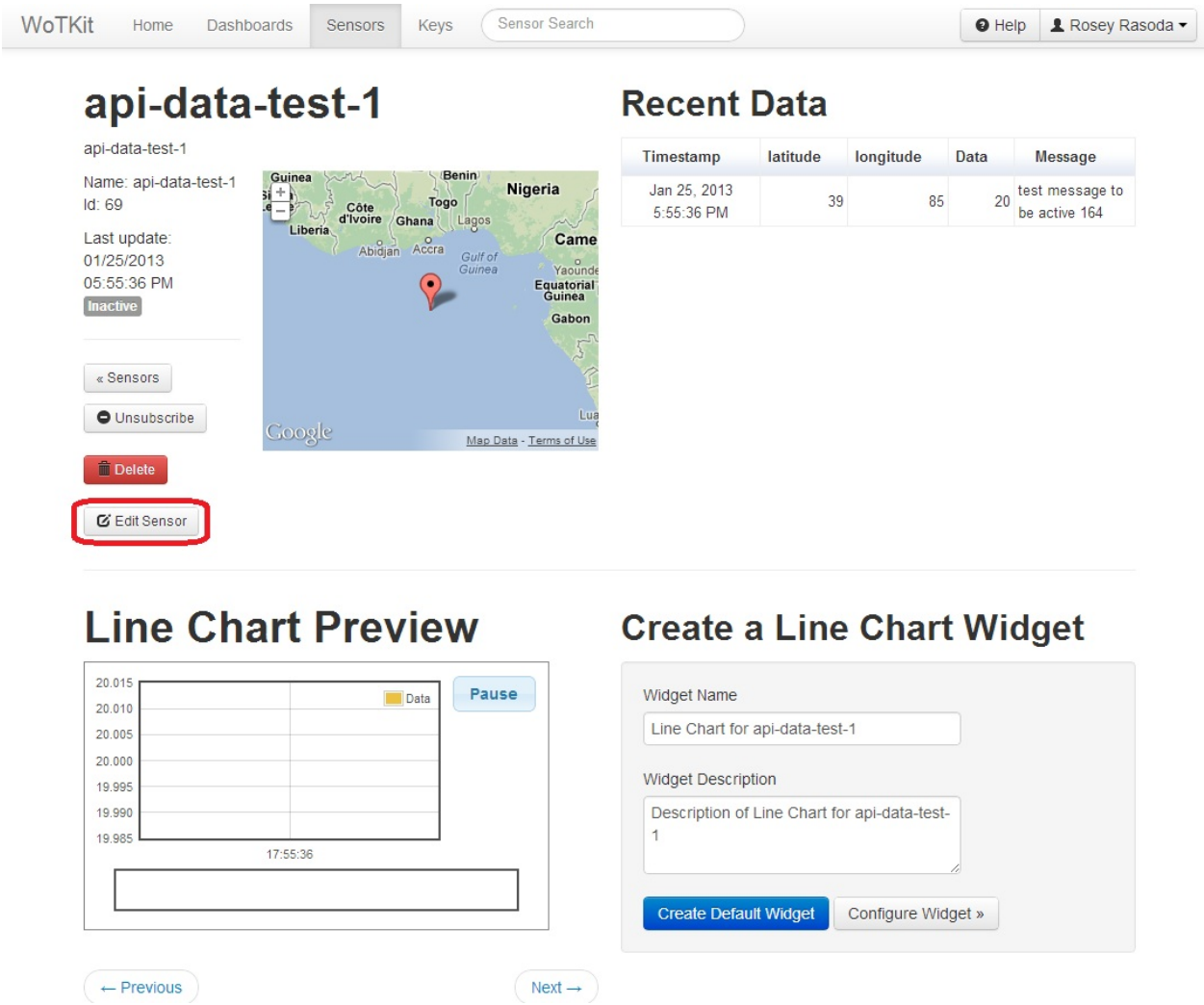
Once the widget is created, you will be taken to the *Widgets View*.



2.3.4 Editing Sensor Information

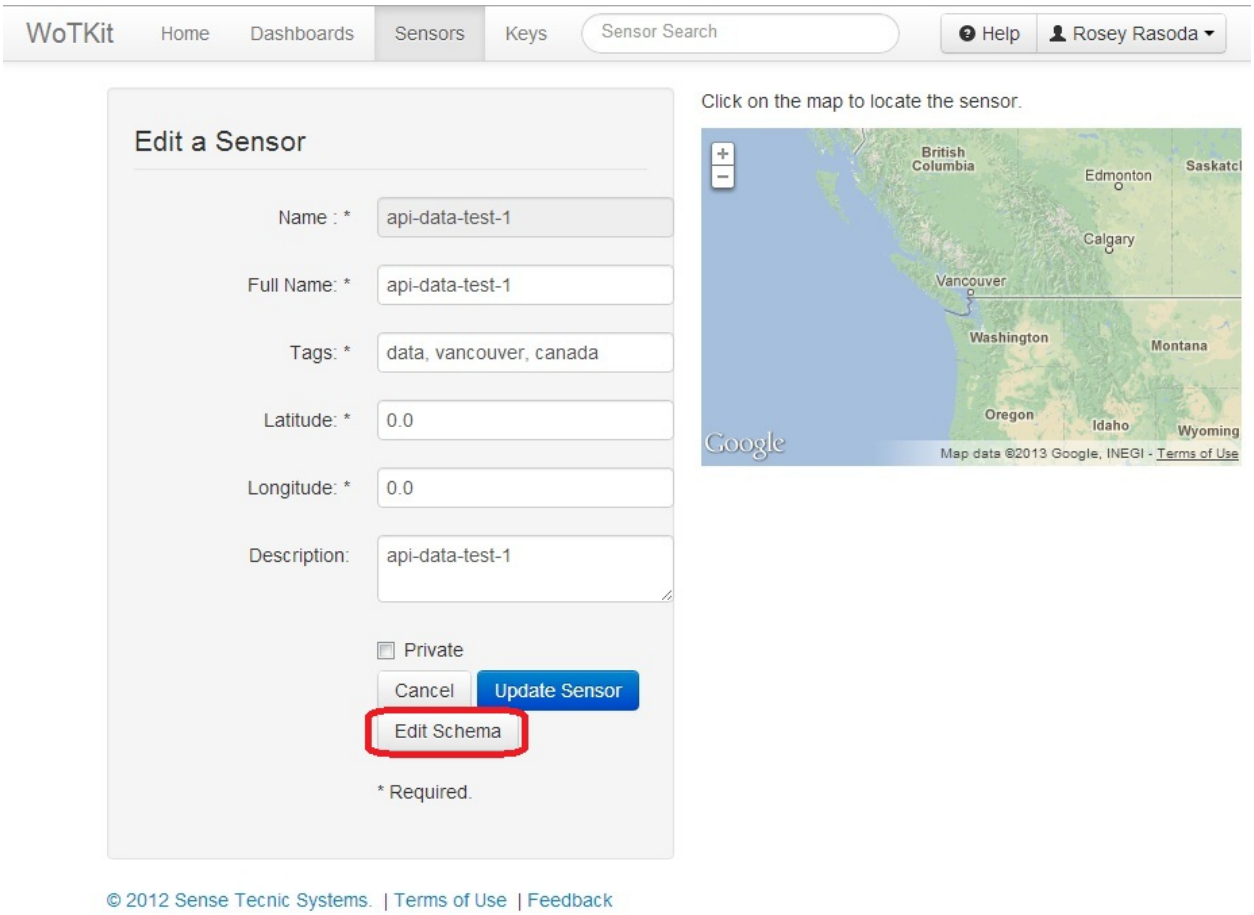
Note: You can only edit sensors you have contributed to the WoTKit.

On the *Monitor View*, select the *Edit Sensor* button.



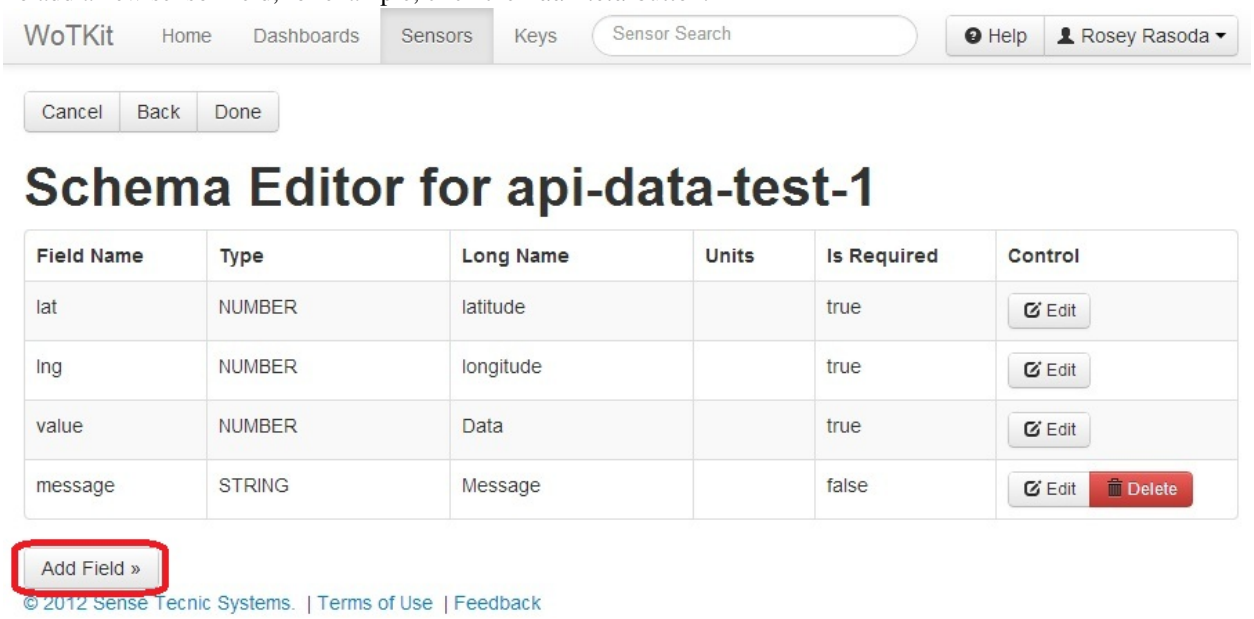
—

By clicking the *Edit Sensor* button, you can change the information you initially registered for the sensor. (The existing information for the sensor will be present to help you edit what is there.) Additionally, you may edit the fields for sensor data using the *Edit Schema* button.



—

By clicking the *Edit Schema* button, you may add, modify, or delete schema data fields properties for the sensor's data. To add a new sensor field, for example, click the *Add Field* button.



2.4 Dashboards

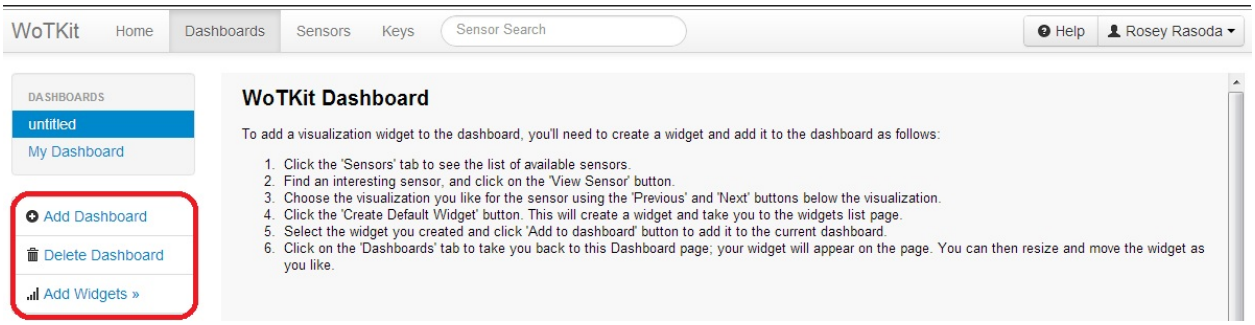
Dashboards allow you to view multiple widgets displaying sensor data.

To view your dashboards, select the *Dashboards* tab.

When viewing a new dashboard, you will see the following view with a help message. By default, there will be an empty dashboard labeled 'untitled'.

From here, you can: * Select an existing dashboards by clicking another dashboard on the left of the page. * Add a new dashboard by clicking ''Add Dashboard''. * Rename a dashboard by clicking on the Edit link. * Delete a new dashboard by clicking ''Delete Dashboard''.

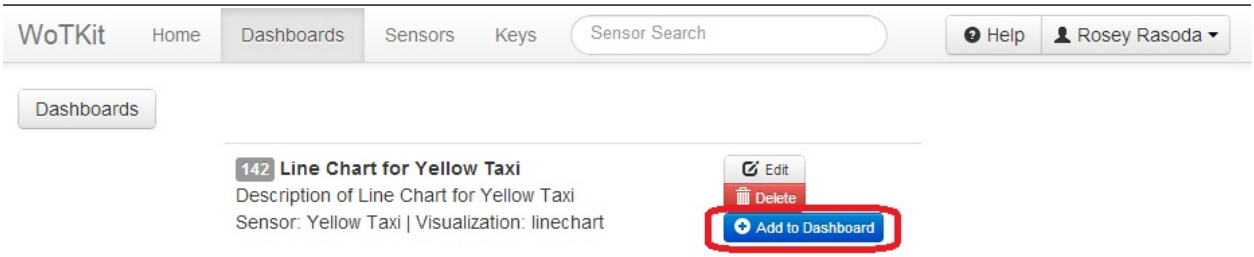
Note: You must have at least one dashboard at all times. If you only have one dashboard, it cannot be deleted.



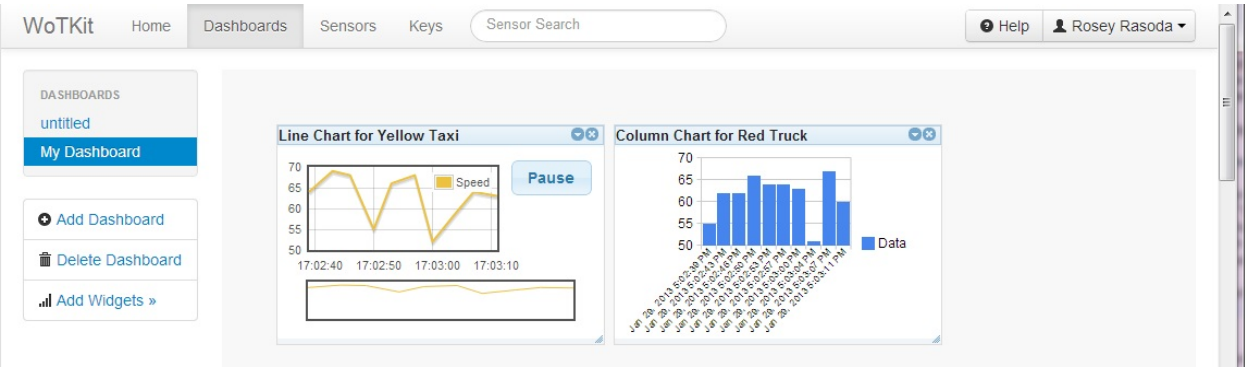
2.4.1 Adding Widgets to a Dashboard

To add a sensor widget to a dashboard: * Select the "Dashboard" tab. * Click on the dashboard that you wish to add a widget to. * Click on ''Add Widgets''. * Choose which widget to add, and click on its ''Add to Dashboard" button.

Note: This will add the widget to the last dashboard you viewed. If there are no widgets, you must create a widget by viewing a sensor. See ...



After adding a widget to a dashboard, the widgets will be displayed on the dashboard as shown. You can drag and resize them to any position on the dashboard.



2.5 Managing WoTKit API Clients

Clients of the WoTKit API include third partly applications, sensor gateways, and scripts.

You can manage the access these clients have to your sensor data and remove the need for external clients to share your personal WoTKit name and password in one of two ways:

- *Keys and Basic Authentication*.
- *Applications and OAuth2 Authorization*.

Once granted access, WoTKit clients can create, modify, or delete sensors and sensor data on your behalf.

2.5.1 Keys and Basic Authentication

A user can generate a key id and key password for WoTKit API clients as follows:

- Select the "Keys" tab and click on the ''New Key" button.
- Fill out the form with information to identify how the key is used, and click ''Add''. Once created, a 'key id' and 'key password' will be generated.
- To view the key, click ''View Key''.

The generated 'key id' and 'key password' can be used as the name and password in the basic authentication headers used when accessing the WoTKit API.

`images/Keys.jpg`

2.5.2 Applications and OAuth2 Authorization

Applications are clients of the WoTKit that can access the WoTKit API on behalf of more than one user. Application credentials provided during the registration process are unique to that application. All applications appear in the WoTKit application list. They can connect to the WoTKit on behalf of a WoTKit user using the OAuth2 authorization process.

To register a new WoTKit application:

- Select the user menu (located at the top right).
- Click on your user name in the top right, and select "Applications" from the drop down list. Click on the ''New Application" button.
- Fill out the form, and click ''Add''. Once registered with an 'application client id', an 'application secret' will be automatically generated.
- To view application information, click ''View details''.

Using the supplied 'application client id' and 'application secret' applications obtain an *access token* to access WoTKit sensors on behalf of a user.

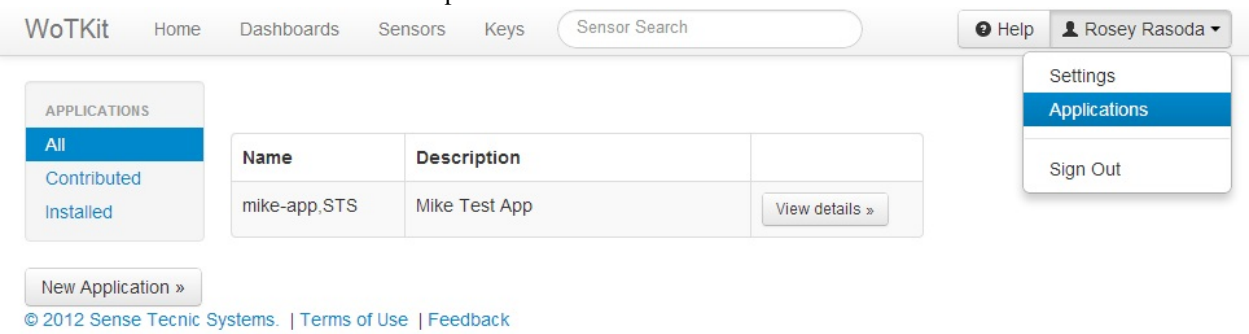
For an application to obtain an access token it requests authorization.

1. The application first requests an authorization code by providing its 'application client id' to the WoTKit using its OAuth2 endpoint:

   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 provide their user name and password to continue.
3. A page will then ask the user to authorize the application to connect to the WoTKit on their behalf. Once authorized, the authorization code is provided to the application by redirection.
4. The application receives the authorization code and exchanges it along with the application credentials for an access token to use the WoTKit API.

Please see the *API Documentation* and in particular *Authentication* for more details.



2.6 Processor

The Processor is a component of the WoTKit that lets you create "*Pipes*" - visual scripts that take data from one or more input sources, process that data in any way, and output that data to one or more sensors and *actuators*.

2.6.1 Creating a Pipe

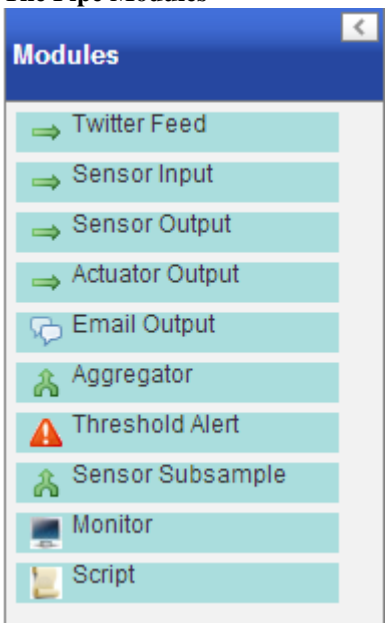
1. Click on **Pipes** and then on **New Pipe**.
   - If you do not have any existing pipes, clicking on Pipes will redirect you to the New Pipe page automatically.
2. Add modules to the pipe by dragging them from the **Modules** component and dropping them on the workspace.
   - Each module will have an input dot on top and/or an output dot below

3. To pipe information between modules, drag the output dot of one module to the input dot of another.

Create/Edit Pipe screen components

Processor UI

The Pipe Modules



- **Menu** (on top) - Actions to save, delete, and start or stop the pipe
- **Modules** (on the left) - Pipe modules that can be dragged and dropped into the workspace
- **Workspace** (centre) - The modules and their connections are displayed here
- **Properties** (on the right) - Basic properties of the pipe, such as name and description
- **Monitor** (on the right) - Custom ''Monitor" modules in the pipe can be viewed in this component
- **Error Monitor** (on the right) - WoTKit's API responses to input/output actions (success or error messages)
- **Minimap** (on the right) - A minimap of the workspace
- **Help** (on the right) - A link to this document

Module types

- **Input**:
  - **Twitter feed - The name of a twitter feed to poll**
    - Updated every time a new tweet is tweeted by the account
    - Fields that will be outputted by this module:
    - *message* - The content of the tweet

  - **Sensor input - The name or ID of a WoTKit sensor that you have access to (either public, or private and owned by yo**
    - Updated every time a new datum is posted to the sensor
    - Fields that will be outputted by this module:
    - *value* - The value of the datum
    - *{any other field}* - If the sensor has any other fields, they will appear by name here
- **Process:**
  - **Aggregator - Aggregates multiple sensors together**
    - Accepts multiple inputs
    - Outputs the input values verbatim whenever an input is updated, and adds a `_sensor` field with the input sensor's name
  - **Threshold Alert** - TODO
  - **Sensor Subsample - Will only pass data from input to output at least that many seconds have passed after the las**
    - Define the ''period" in seconds
    - Any input sent during the defined period after the last input is suppressed
  - **Script - A custom Python script**
    - The input and output to/from the script are in the python dictionaries named called *input* and *output*
    - **For example, if you input a sensor to the script and output to another sensor and you want to multiply the**
      `output['value'] = input['value'] * 2 + 1`
    - Any entry in the *output* dictionary will be sent as the output of this script
  - **{named script} - A copy of the template that you defined in the *Scripts* page**
    - Note that this creates a copy of the template. Any changes made to this script will not reflect in the original template
- **Log:**
  - **Monitor** - A debugging module. Will display everything that is sent to it's input as a table in the Monitor component on this screen
- **Output:**
  - **Sensor output** - Post the input to the named sensor
  - **Actuator output** - Post the input to the named actuator
  - **Email output - Email the input to the provided email address**
    - To replace text with a value use the following syntax: `${value}`

CHAPTER

THREE

API DOCUMENTATION

3.1 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 *apps-oauth-label* for details.

3.1.1 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

3.1.2 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**.

(Please replace the `{key_id}` and `{key_password}` in the code with appropriate values copied from the WoTKit UI.)

example

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


3.1.3 Registered Applications and OAuth2

Applications are registered on the WoTKit UI (http://wotkit.sensetecnic.com/wotkit/apps). They can be installed by many users, but the credentials are unique to the contributor.

To grant a 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 the user's behalf, using OAuth2 authorization.

The OAuth2 authorization asks 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, authorizing access. (No further id/passwords are needed.)

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

example

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

In order to obtain an access token for your client, the following steps are taken:

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

   ```
   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. Using the application credentials, this code is exchanged for an access token. This access token is then appended to the end of each URL, authorizing access.

Example: PHP file pointed to by `{redirect_uri}`

```php
<?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;
        }
        ?>
```

3.1.4 Access Token Facts

When obtaining an access token, the 'response' field holds the following useful information:

- `response->access_token`
- `response->expires_in`
  - default value is approx. 43200 seconds (or 12 hrs)

3.2 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.

```
HTTP/1.1 404 Not Found

{
  "error" : {
        "status" : 404,
        "code" : 0,
        "message" : "No sensor with that id",
        "developerMessage" : "user: mike sensor:gfhghjhj is not in the database"
  }
}
```

3.3 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 **ORGANIZATION**: The sensor is visible to everyone in the same organization as the sensor **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.
tags	the list of tags for the sensor
data	sensor data (not shown yet)

** Required when creating a new sensor.

3.3.1 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.
tags	list of comma separated tags
orgs	list of comma separated organization names
private	****true** - private sensors only; **false** - public only** deprecated, use **visibility** instead
visibility	filter by the visibility of the sensors, either of **public**, **organization**, or **private**
text	text to search for in the name, long name and description
active	when true, only returns sensors that have 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**

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

URL	http://wotkit.sensetecnic.com/api/sensors?{query}
Privacy	Public or Private
Format	json
Method	GET
Returns	On error, an appropriate HTTP status code; On success, OK 204 and a list of sensor descriptions matching the query.

example

```
curl --user {id}:{password}
"http://wotkit.sensetecnic.com/api/sensors/sensetecnic.mule1?tags=canada"
```

Output:


```
[
  {
        "tags":["data","vancouver","canada"],
                "latitude":0.0,
                "longitude":0.0,
                "longName":"api-data-test-1",
                "lastUpdate":"2013-01-26T01:55:36.514Z",
                "name":"api-data-test-1",
                "fields":
                        [{"required":true, "longName":"latitude",
                          "lastUpdate":"2013-01-26T01:55:36.514Z",
                          "name":"lat", "value":39.0, "type":"NUMBER","index":0},
                        {"required":true,"longName":"longitude",
                         "lastUpdate":"2013-01-26T01:55:36.514Z",
                         "name":"lng","value":85.0,"type":"NUMBER","index":1},
                        {"required":true,"longName":"Data",
                         "lastUpdate":"2013-01-26T01:55:36.514Z
                         "name":"value","value":20.0,"type":"NUMBER","index":2},
                        {"required":false,"longName":"Message",
                         "lastUpdate":"2013-01-26T01:55:36.514Z",
                         "name":"message","value":"test message to be active 164",
                         "type":"STRING","index":3}],
        "id":69,
        "visibility":"PUBLIC",
        "owner":"roseyr",
        "description":"api-data-test-1"
  },

  {
        "tags":["data","canada","edmonton"],
        "latitude":0.0,
        "longitude":0.0,
        "longName":"api-data-test-2",
        "lastUpdate":"2013-01-26T01:55:42.400Z",
        "name":"api-data-test-2",
        "fields":
                [{"required":true,"longName":"latitude",
                  "lastUpdate":"2013-01-26T01:55:37.537Z",
                  "name":"lat","value":65.0,"type":"NUMBER","index":0},
                {"required":true,"longName":"longitude",
                 "lastUpdate":"2013-01-26T01:55:37.537Z",
                 "name":"lng","value":74.0,"type":"NUMBER","index":1},
                {"required":true,"longName":"Data",
                 "lastUpdate":"2013-01-26T01:55:37.537Z",
                 "name":"value","value":82.0,"type":"NUMBER","index":2},
                {"required":false,"longName":"Message",
                 "lastUpdate":"2013-01-26T01:55:37.537Z",
                 "name":"message","value":"test message to be active 110",
                 "type":"STRING","index":3}],
        "id":70,
        "visibility":"PUBLIC",
```

```
        "owner":"roseyr",
        "description":"api-data-test-1"
  },

  {
        "tags":["data","canada","winnipeg"],
        "latitude":0.0,
        "longitude":0.0,
        "longName":"api-data-test-3",
        "lastUpdate":"2013-01-26T01:55:34.488Z",
        "name":"api-data-test-3",
        "fields":
                [{"required":true,"longName":"latitude","name":"lat","value":0.0,
                  "type":"NUMBER","index":0},
                {"required":true,"longName":"longitude","name":"lng","value":0.0,
                 "type":"NUMBER","index":1},
                {"required":true,"longName":"Data","name":"value","value":0.0,
                 "type":"NUMBER","index":2},
                {"required":false,"longName":"Message","name":"message",
                 "type":"STRING","index":3}],
        "id":71,
        "visibility":"PUBLIC",
        "owner":"roseyr",
        "description":"api-data-test-3"
  }
]
```


3.3.2 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/sensors/{sensorname}
Privacy	Public or Private
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

---

example

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

Output:


```
{
        "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"}
}
```


3.3.3 Creating/Registering a Sensor

To register a sensor, you POST a sensor resource to the url `/sensors`.

- The sensor resources is a JSON object.
- The "name", "longName", and "description" fields are required when creating a sensor.
- The "latitude" and "longitude" fields are optional and will default to 0 if not provided.
- The "visibility" field is optional and will default to "PUBLIC" if not provided.
- The "tags", "fields" and "organization" information are optional.
- If "visibility" is set to ORGANIZATION, a valid "organization" must be supplied.
- 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.

To create a sensor:

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

example

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

For this example, the file *test-sensor.txt* contains the following. This is the minimal information needed to register a sensor resource.

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

3.3.4 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/sensors
Privacy	Private
Format	json
Method	PUT
Returns	HTTP status code; Created 201 if successful; Bad Request 400 if sensor is invalid; Conflict 409 if sensor with the same name already exists ; On Created 201 or some errors (not all) you will receive a JSON dictionary where the keys are the sensor names and the values are true/false depending on whether creating the sensor succeeded. For Created 201 all values will be true.

3.3.5 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 "visibility" is set to ORGANIZATION, a valid "organization" must be supplied.
- 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/sensors/{sensorname}
Privacy	Private
Format	json
Method	PUT
Returns	HTTP status code; No Content 204 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/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"]
}
```

3.3.6 Deleting a Sensor

Deleting a sensor is done by deleting the sensor resource.

To delete a sensor owned by the current user:

URL	http://wotkit.sensetecnic.com/api/sensors/{sensorname}
Privacy	Private
Format	not applicable
Method	DELETE
Returns	HTTP status code; No Response 204 if successful

example

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

3.4 Sensor Subscriptions

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

3.4.1 Get Sensor Subscriptions

To view sensors that the current user is subscribed to:

URL	http://wotkit.sensetecnic.com/api/subscribe
Privacy	Private
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

3.4.2 Subscribe

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

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

3.4.3 Unsubscribe

To unsubscribe from a sensor:

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

3.5 Sensor Fields

Each sensor has the following default fields:

Field Name	Information
value	The numerical data for the sensor. Required.
lat	The latitude of the sensor. Required.
lng	The longitude of the sensor. Required.
message	The string message for the sensor. Not Required.

Each pieces of sensor field data has the following sub-fields:

Sub-Field Name	Information
name	The unique identifier for the field. It is required when creating/updating a field and cannot be changed.
longName	The display name for the field.
type	Can be "NUMBER" or "STRING". It is required when creating/updating a field.
required	Is a boolean field. If true, data sent to a sensor must include this field or an error will result. Optional.
units	Is a string. Optional.
index	The numerical identifier of the field. Automatically populated.
value	The last value sent to the field. Automatically populated.
lastUpdate	The time stamp of the last value sent to the field. Automatically populated.

3.5.1 Querying Sensor Fields

To query all sensor fields for a specific sensor:

URL	http://wotkit.sensetecnic.com/api/sensors/{sensorname}/fields
Privacy	Public or Private
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

To query a single sensor field for a specific sensor:

URL	http://wotkit.sensetecnic.com/api/sensors/{sensorname}/fields/{fieldName}
Privacy	Public or Private
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

3.5.2 Updating a Sensor Field

You can update an existing sensor field or add a new sensor field by performing a PUT and including the field name in the URL. The field information is supplied in a JSON format.

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

- When inputting field data, the sub-fields "name" and "type" are required-both for adding a new field or updating an existing one.
- 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.
- For the default fields (lat, lng, value, message), only the "longName" and "unit" sub-fields may be updated.

To update/add a sensor field:

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

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

example

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

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

```
{
        "name"=>"test-field",
        "type"=>"STRING"
}
```

To then update "test-field" sub-fields, the same curl command would be used, and ''field-data.txt" could now contain the following.

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

3.5.3 Deleting a Sensor Field

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

None of the existing default fields (lat, lng, value, message) can be deleted.

To delete a sensor field:

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

3.6 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:

Name	Value Description
timestamp	**the time that the sensor data was collected. This is a long integer repre** Optional; if not supplied, a server-supplied timestamp will be used.
sensor_id	the globally unique sensor id that produced the data.
sensor_name	the globally unique sensor name, in the form `{username}.{sensorname}`
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 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)`. Using Java: `System.currentTime()`.

3.6.1 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.

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.

To send new data:

URL	http://wotkit.sensetecnic.com/api/sensors/{sensorname}/data
Privacy	Private
Format	not applicable
Method	POST
Returns	HTTP status code; No Response 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/sensors/test-sensor/data'
```

3.6.2 Sending Bulk Data

To send a range of 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

- The data sent must contain a list of JSON objects containing a timestamp and a value.
- 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, the list must be ordered from earliest to most recent timestamp. Any existing data within this timestamp range will be deleted and replaced by the new data.

To update data:

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

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

3.6.3 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/sensors/{sensorname}/data/{timestamp}
Privacy	Private
Format	not applicable
Method	DELETE
Returns	HTTP status code; No Response 204 if successful

3.6.4 Raw Data Retrieval

To retrieve raw data use the following:

URL	http://wotkit.sensetecnic.com/api/sensors/{sensor-name}/data?{query-params}
Privacy	Public or Private
Format	json
Method	GET
Returns	On success, OK 200 with 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.)
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

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

3.6.5 Formatted Data Retrieval

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

URL	http://wotkit.sensetecnic.com/api/sensors/{sensor-name}/dataTable?{query-params}
Privacy	Public or Private
Format	json
Method	GET
Returns	On success, OK 200 with a list of timestamped data records.

In addition to the above query parameters, the following parameters are also supported:

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 "test-sensor", select all data where value was greater than 20, and display the output as an html table.

example

```
curl --user {id}:{password} http://wotkit.sensetecnic.com/api/sensors/test-sensor/
dataTable?tq=select%20*%20where%20value%3E20&reqId=1&out=html
```

3.6.6 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 following parameters may be added to the `/data` url:

- scope
- tags
- private (deprecated, use visibility instead)
- visibility
- text
- active
- start
- end
- after
- afterE
- before
- beforeE
- **orderBy**
  - **sensor**: which groups data by sensor_id
  - **time** (default): which orders data by timestamp, regardless of the sensor it comes from.

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

URL	http://wotkit.sensetecnic.com/api/data?{query-param}={query-value}&{param}={value}...
Privacy	Public or Private
Format	json
Method	GET
Returns	On success, OK 200 with a list of timestamped data records.

example

```
curl --user {id}:{password}
"http://wotkit.sensetecnic.com/api/data?subscribed=all&beforeE=20&orderBy=sensor"
```

3.7 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.

To demonstrate actuators, the control visualization that comes with the WoTKit sends three type 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.

Any name/value pair can be sent to an actuator in a message, these are just the names sent by the visualization.

3.7.1 Sending Actuator Messages

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

URL	http://wotkit.sensetecnic.com/api/sensors/{sensorname}/message
Privacy	Public or Private
Format	json
Method	POST
Returns	On success, OK 200 (no content).

3.7.2 Receiving Actuator Messages

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/control/sub/{sensor-name}
Privacy	Private
Format	json
Method	POST
Returns	On success, OK 200 with JSON 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` 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/control/sub/{subscription-id}?wait={wait-time}
Privacy	Private
Format	json
Method	GET
Returns	On success, OK 200 with JSON containing control messages.

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
headers = {'Authorization': authheader}

#subscribe to the controller and get the subscriber ID
conn.request("POST", "/api/control/sub/" + actuator, 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/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
```

3.8 Tags

You can get a list of tags, either the most used by public sensors or by a sensor query.

3.8.1 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	**true - private sensors only; false - public only** (Deprecated, use visibility instead)
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 sensors URL as follows:

URL	http://wotkit.sensetecnic.com/api/tags?{query}
Privacy	Public or Private
Format	json
Method	GET
Returns	On error, an appropriate HTTP status code; On success, OK 200 and a list of tag count objects matching the query.

example

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

3.9 Users

Admins can list, create and delete users from the system.

3.9.1 List/Query Users

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

Name	Value Description
text	text to search for in the username, first name and/or last name
reverse	**true** to get the oldest users first; **false** (default) to get newest first
offset	offset into list of users for paging
limit	limit to show for paging. The maximum number of users to display is 1000.

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

URL	http://wotkit.sensetecnic.com/api/users?{query}
Privacy	Admin
Format	json
Method	GET
Returns	On error, an appropriate HTTP status code; On success, OK 200 and a list of users matching the query.

3.9.2 Viewing a Single User

To view a single user, query by username or id as follows:

URL	http://wotkit.sensetecnic.com/api/users/{username}
Privacy	Admin
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

example

```
curl --user {id}:{password}
"http://wotkit.sensetecnic.com/api/users/1"
```

Output:

```
{
        'id': 1,
        'username': 'sensetecnic',
        'email': 'info@sensetecnic.com',
        'firstname': 'Sense',
        'lastname': 'Tecnic',
        'enabled': True,
        'accountNonExpired': True,
        'accountNonLocked': True,
        'credentialsNonExpired': True
}
```

3.9.3 Creating/Registering a User

To register a user, you POST a user resource to the url */users*.

- The user resources is a JSON object.
- The "username", "firstname", "lastname", "email", and "password" fields are required when creating a user.
- The "timeZone" field is optional and defaults to UTC.
- The username must be at least 4 characters long.

To create a user:

URL	http://wotkit.sensetecnic.com/api/users
Privacy	Admin
Format	json
Method	POST
Returns	HTTP status code; Created 201 if successful; Bad Request 400 if user is invalid; Conflict 409 if user with the same username already exists

3.9.4 Updating a User

- You may only update the following fields: "firstname", "lastname", "email", "timeZone" and "password".
- Only fields that will be present in the JSON object will be updated. The rest will remain unchanged.

To update a user:

URL	http://wotkit.sensetecnic.com/api/users/{username}
Privacy	Admin
Format	json
Method	PUT
Returns	HTTP status code; No Content 204 if successful

3.9.5 Deleting a User

Deleting a user is done by deleting the user resource.

To delete a user:

URL	http://wotkit.sensetecnic.com/api/users/{username}
Privacy	Admin
Format	not applicable
Method	DELETE
Returns	HTTP status code; No Response 204 if successful

3.10 Organizations

All users can see all organizations, and admins can manipulate them.

3.10.1 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/orgs?{query}
Privacy	Public
Format	json
Method	GET
Returns	On error, an appropriate HTTP status code; On success, OK 200 and a list of organizations matching the query from newest to oldest.

3.10.2 Viewing a Single Organization

To view a single organization, query by name:

URL	http://wotkit.sensetecnic.com/api/orgs/{org-name}
Privacy	Public
Format	json
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

example

```
curl "http://wotkit.sensetecnic.com/api/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"
}
```

3.10.3 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.
- The "name" and "longName" fields are **required** and must both be at least 4 characters long.
- The "imageUrl" and "description" fields are **optional**.

To create an organization:

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

3.10.4 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/orgs/{org-name}
Privacy	Admin
Format	json
Method	PUT
Returns	HTTP status code; No Content 204 if successful

3.10.5 Deleting an Organization

Deleting an organization is done by deleting the organization resource.

To delete a user:

URL	http://wotkit.sensetecnic.com/api/orgs/{org-name}
Privacy	Admin
Format	not applicable
Method	DELETE
Returns	HTTP status code; No Content 204 if successful

3.10.6 Organization Membership

List all members of an Organization

To query for organization members:

URL	http://wotkit.sensetecnic.com/api/orgs/{org-name}/members
Privacy	Admin
Format	not applicable
Method	GET
Returns	On error, an appropriate HTTP status code; On success, OK 200 and 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/orgs/{org-name}/members
Privacy	Admin
Format	json
Method	POST
Returns	On error, an appropriate HTTP status code; On success, OK 204.

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/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/orgs/{org-name}/members
Privacy	Admin
Format	json
Method	DELETE
Returns	On error, an appropriate HTTP status code; On success, OK 204.

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

3.11 Sensor Groups

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

Currently, all Sensor Groups have **public** visibility, but **only** the **owner** (creator) can add/remove sensors from the group.

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

3.11.1 Sensor Group Format

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

Field Name	Type	Required	Notes
id	Integer	true	The id contains a unique number which is used to identify the group
name	String[4,50]	true	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
longName	String[,255]	optional	A readable name used for visual interfaces.
owner	String[4,50]	true	The name of the group's owner. This field is set by the system and cannot be modified.
description	String[,255]	optional	A simple description of the group
imageUrl	String[,255]	optional	A string url to an image which can be used to represent this group
sensors	Array[Sensor]	optional	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 follows:

```
{
  id: 49,
  name: "water-sensor",
  longName: "A water sensor",
  owner: "robertl",
  description: "This is a short description",
  imageUrl: "http://someurl.com/water-sensor.jpg"
  sensors: []
}
```

3.11.2 List Groups

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

URL	http://wotkit.sensetecnic.com/api/groups/
Method	GET
Returns	OK 200, along with list

example

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

3.11.3 Viewing a Single Sensor Group

Similar to listing a group, but retrieving only a single sensor. Replace `{group-name}` with `group.id` or `group.name`. The API accepts both formats

URL	http://wotkit.sensetecnic.com/api/groups/{group-name}
Method	GET
Returns	OK 200

example

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

3.11.4 Creating a Sensor Group

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

On creation, the **id** and **owner** fields are **ignored** because they are system generated.

URL	http://wotkit.sensetecnic.com/api/groups
Method	POST
Returns	If a sensor with the same name exists, ERROR 409. Otherwise, OK 204.

3.11.5 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}` in the URL with either `group.id` or `group.name`. The API accepts both formats.

URL	http://wotkit.sensetecnic.com/api/groups/{group-name}
Method	PUT
Returns	If user has no permissions to edit group, returns UNAUTHORIZED 401, otherwise OK 204

3.11.6 Deleting a Sensor Group

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

URL	http://wotkit.sensetecnic.com/api/groups/{group-name}
Method	DELETE
Returns	If user has no permissions to edit group, returns UNAUTHORIZED 401, otherwise OK 204

3.11.7 Adding a Sensor to Sensor Group

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

URL	http://wotkit.sensetecnic.com/api/groups/{group-name}/sensors/{sensor-id}
Method	POST

The response will contain one of the following response codes.

Return Code	Description
OK 204	No Content is given.
400	Sensor is already a member of sensor group
401	User is unauthorized to edit group.

3.11.8 Removing a Sensor from Sensor Group

The format is the same as *Adding a Sensor to Sensor Group* except replacing `method` with `DELETE`

URL	http://wotkit.sensetecnic.com/api/groups/{group-name}/sensors/{sensor-id}
Method	DELETE

The response will contain one of the following codes.

Return Code	Description
OK 204	No Content is given. If a sensor does not exist in a group, this is also returned.
401	User is unauthorized to edit group

3.12 News

To get "news" (a list of interesting recent things that happened in the system):

URL	http://wotkit.sensetecnic.com/api/news
Privacy	Public
Format	not applicable
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

---

example

```
curl "http://wotkit.sensetecnic.com/api/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'
}]
```

3.13 Statistics

To get some statistics (eg. number of public sensors, active sensors, new sensors, etc...):

URL	http://wotkit.sensetecnic.com/api/stats
Privacy	Public
Format	not applicable
Method	GET
Returns	Appropriate HTTP status code; OK 200 - if successful

example

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

Output:

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

3.14 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.

3.14.1 Authenticating using Smart Streets Developer Keys

More on this to come

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*