Contents

General API Concept Overview

Resource Representation

All resources are specified with a URI. When presented to the client, the resource is represented as a JSON object with at minimum an `_href` property that contains its URI, a `_disp` property that gives a short string representation of that resource suitable for use as link text, and a `_type` property that uniquely identifies that resource type. Note that in the future the `_type` property will be a URL string that provides a link to that type's documentation.

When a parent resource (such as a Collection Resource or some other related resource) contains embedded child resources, those resources may be fully embedded or may only contain the `_href` and `_disp` properties, in which case the client should follow the given link to access the child resource. Clients should be written to handle both cases transparently, to support future server-side changes and optimizations. That is, when accessing related resources the client should check for the presence of a `_type` field, which will signify whether the related resource is embedded or linked.

Example of an embedded resource

```
GET http://example.com/api/books/483
```

```
{
    "_href": "http://example.com/api/books/483",
    "_type": "book",
    "_disp": "Jane Eyre",
    "title": "Jane Eyre",
    "author": {
        "_href": "http://example.com/api/authors/910",
        "_type": "author",
        "_disp": "Charlotte Bronte",
        "firstName": "Charlotte",
        "lastName": "Bronte",
        "books": {
            "_href": "/api/authors/910/books/"
            "_disp": "Books by Charlotte Bronte"
        }
    }
}
```

Example of a linked resource

```
GET http://example.com/api/book/483
```

```
{
    "_href": "http://example.com/api/books/483",
    "_type": "book",
    "_disp": "Jane Eyre",
    "title": "Jane Eyre",
    "author": {
        "_href": "http://example.com/api/authors/910",
        "_disp": "Charlotte Bronte"
    }
}
```

A Parent resource might also contain a list of related resources, such as the list of books by an author. In this case the "books" field would be itself a Collection Resource, so the Author resource would look like:

```
GET http://example.com/api/authors/910

{
    "_href": "http://example.com/api/authors/910",
    "_type": "author",
    "_disp": "Charlotte Bronte",
    "firstName": "Charlotte",
    "lastName": "Bronte"
    "books": {
        "_href": "http://example.com/api/authors/910/books/",
        "_disp": "Books by Charlotte Bronte"
    }
}
```

Or if the "books" resource is expanded:

```
GET http://example.com/api/authors/910

{
    "_href": "http://example.com/api/authors/910",
    "_type": "author",
    "firstName": "Charlotte",
    "lastName": "Bronte"
    "_disp": "Charlotte Bronte",
    "books": {
        "_href": "http://example.com/api/authors/910/books/",
        "meta": {
          "totalCount": 24,
          "last": {
```

```
      "_disp": "21 through 23",
      "_href": "http://localhost:8000/api/devices/?limit=3&site_id=2&offset=21"
     },
     "next": {
      "_disp": "3 through 5",
      "_href": "http://localhost:8000/api/devices/?limit=3&site_id=2&offset=3"
     }
    },
    "_type": "resource-list",
    "data": [
      {
        "_href": "http://example.com/api/books/483",
        "_type": "book",
        "_disp": "Jane Eyre",
        "title": "Jane Eyre",
        "author": {
          "_href": "http://example.com/api/authors/910",
          "_disp": "Charlotte Bronte"
        }
      },
      {
        "_href": "http://example.com/api/books/918",
        "_type": "book",
        "_disp": "Shirley",
        "title": "Shirley",
        "author": {
          "_href": "http://example.com/api/authors/910",
          "_disp": "Charlotte Bronte"
        }
      },
      {
        "_href": "http://example.com/api/books/710",
        "_type": "book",
        "_disp": "The Professor",
        "title": "The Professor",
        "author": {
          "_href": "http://example.com/api/authors/910",
          "_disp": "Charlotte Bronte"
        }
      },
    ]
  }
}
```

Collection Resources

Collection resources are fully-formed resources so all the formatting specifications above apply. Currently collection resources are homogenous, that is all the resources contained in the collection are the same type. Brand new resources can be created by posting them to a collection resource. In the case that the resource to be posted has some sort of parent relation that indicates which resource is its parent, that field can be omitted and will be ignored because it is defined by the collection.

Getting Resources

The URIs given at the API entry point point to Collection Resources. Sending an HTTP GET request to a Resource List will return a response with the requested resources as well as metadata about the response. This metadata includes the total number of resources represented by this collection. If there are more resources than will fit into a single response, the meta field may also contain links to the first, last, previous, and next pages.

```
GET http://example.com/api/some_resources/

{
    "_href": "http://example.com/api/some_resources/",
    "_type": "resource-list",
    "_disp": "Some Resources",
    "meta": { "total_count": 2 },
    "data": [
        {
            "_href": "http://example.com/api/some_resources/192",
            "_type": "some_resource",
            "_disp": "Idaho",
            "name": "A great resource",
            "state": "Idaho"
        },
        {
            "_href": "http://example.com/api/some_resources/193",
            "_type": "some_resource",
            "_disp": "New York",
            "name": "Another Resource",
            "state": "New York"
        }
    ]
}
```

Sending a GET to the URI for a specific resource will return only that resource, e.g.

```
GET http://example.com/api/some_resources/193
```

```
{
    "_href": "http://example.com/api/some_resources/193",
    "_type": "some_resource",
    "_disp": "New York",
    "name": "Another Resource",
    "state": "New York"
}
```

The Tidmarsh API

Entry Point

The API entry point is at `http://tidmarsh.media.mit.edu/api/`. A `GET` request will give you links to the available sites in the following format:

```
{
    "_href": "http://tidmarsh.media.mit.edu/api/",
    "_type": "api-root",
    "_disp": "Tidmarsh API",
    "sites": {
        "_href": "http://tidmarsh.media.mit.edu/api/sites/",
        "_type": "resource-list",
        "_disp": "site",
        "meta": { "total_count": 2 },
        "data": [
            {
                "_href": "http://tidmarsh.media.mit.edu/api/sites/92",
                "_type": "site",
                "_disp": "DoppelLab",
                "name": "DoppelLab",
                "latitude": 42.360461,
                "longitude": -71.087347,
            },
            {
                "_href": "http://tidmarsh.media.mit.edu/api/sites/12",
                "_type": "site",
                "_disp": "TidMarsh",
                "name": "TidMarsh",
                "latitude": 39.948171,
                "longitude": -70.827105,
            }
        }
    }
}
```

Currently the only top-level resource available is the collection of Sites. The client first selects which site they are interested in and can navigate from there to explore that site. Clients should not assume hard-coded URIs, but should instead get the proper URI from the API entry point.

Base Resource Types

NOTE: At some point this information should be moved into the docstrings of the resources themselves, so we can generate the documentation as well as the information at the `_type` URLs from the same info.

Site

An installation of Doppel2, usually on the scale of several or many buildings.

Resource Fields

- `_href` (string) - URI of this resource
- `_type` (string) - Type of this resource
- `_disp` (string) - Short display string for this resource
- `name` (string) - Name of this site
- `latitude` (float) - The latitude of the site
- `longitude` (float) - The longitude of the site
- `devices` (related resource) - A collection of all the devices in this site. New devices can be POSTed to this collection to add them to this site.

Example

```
{
    "_href": "http://tidmarsh.media.mit.edu/api/sites/758",
    "_type": "site",
    "_disp": "TidMarsh",
    "name": "TidMarsh",
    "latitude": 39.948171,
    "longitude": -70.827105,
    "devices": {
        "_disp": "devices",
        "_href": "http://tidmarsh.media.mit.edu/api/sites/758/devices"
    }
}
```

Device

A device that may contain several sensor channels.

Resource Fields

- `_href` (string) - URI of this resource
- `_type` (string) - Type of this resource
- `_disp` (string) - Short display string for this resource
- `name` (string) - Name of this device
- `site` (related resource) - The site this device is a part of
- `description` (string) - A longer description of this device
- `building` (string) - The building the device is in
- `floor` (string) - The floor of the building
- `room` (string) - The room containing the device
- `sensors` (related resource) - A collection of all the sensors in this device. New sensors can be POSTed to this collection to add them to this device.

Example

```
{
    "_href": "http://tidmarsh.media.mit.edu/api/devices/129",
    "_type": "device",
    "_disp": "Bathroom Thermostat",
    "name": "Bathroom Thermostat",
    "site": {
        "_href": "http://tidmarsh.media.mit.edu/api/sites/928",
        "_disp": "Summer Cabin"
    },
    "description": "Thermostat in the pool house bathroom",
    "building": "Pool House",
    "floor": "2",
    "room": "Bathroom",
    "sensors": {
        "_href": "http://tidmarsh.media.mit.edu/api/devices/129/sensors",
        "_disp": "sensors"
    }
}
```

Sensor

A sensor captures a single channel of data. There may be multiple sensors on a single device. Issuing a GET request for a Sensor resource also includes the current value for that sensor. The value could be a scalar value or some other TBD data types.

Resource Fields

- `_href` (string) - URI of this resource
- `_type` (string) - Type of this resource
- `_disp` (string) - Short display string for this resource
- `device` (related resource) - The device this sensor is part of
- `history` (related resource) - Collection of data from this sensor
- `metric` (string) - What the sensor is measuring (e.g. “temperature”)
- `unit` (string) - The unit the data is in (e.g. “kWh”)

Example

```
{
    "_href": "http://tidmarsh.media.mit.edu/api/sensors/758",
    "_disp": "temperature",
    "_type": "sensor",
    "device": {
        "_href": "http://tidmarsh.media.mit.edu/api/devices/358",
        "_disp": "Bathroom Thermostat"
    },
    "history": {
        "_href": "http://tidmarsh.media.mit.edu/api/sensordata?sensor=758",
        "_disp": "data"
    },
    "metric": "temperature",
    "unit": "C"
}
```

Sensor Data

Sensor Data is the raw data captured by the sensors. Each Sensor Data resource represents a single data point, so the client is often looking at large aggregations of this data.

Resource Fields

- `_href` (string) - URI of this resource
- `_type` (string) - Type of this resource
- `_disp` (string) - Short display string for this resource
- `value` (float) - The value of the sensor data
- `timestamp` (ISO 8601 timestamp) - Timestamp marking when the data was captured

Example

```
{
    "_href": "http://tidmarsh.media.mit.edu/api/scalar_data/193",
    "_type": "scalar_data",
    "_disp": "2013-04-12T03:30:00Z",
    "value": 25.2,
    "timestamp": "2013-04-12T03:30:00Z",
}
```

Dev Server Initial Setup

To develop for doppel2 the best way is with a virtual machine. The following instructions should guide you through setting up a development environment that should match the production machine.

First you'll want to make sure that both Virtualbox and Vagrant are installed, as well as mercurial.

Then clone the code from the repository with:

```
hg clone http://simonetti.media.mit.edu/hg/doppel2
```

Mercurial (hg) will create a folder called doppel2 and check out the code into it.

from within that folder run "vagrant up" and it should instantiate the virtual machine.

after the machine is up you can run "vagrant ssh" to ssh into the new VM. From there you can follow the below instructions on setting up a Doppel2 server.

Doppel2 Server Setup Instructions

Install puppet and the proper puppet modules with

```
sudo apt-get install puppet
(or on Arch use yaourt -S puppet)
sudo puppet module install puppetlabs/postgresql -v3.2.0
```

NOTE FOR ARCH:

The puppet manifest doesn't work well on Arch, so just install the python packages with pip system packages with pacman that are listed in the manifest, and comment those stanzas out. I also had to set the ownership of /var/lib/postgres with:

```
sudo chown -R postgres:users /var/lib/postgres
```

END OF ARCH-SPECIFIC NOTE

edit manifest.pp and change the database username and password.

Edit settings.py and set the username and password, as well as the `SECRET_KEY` setting.

Then set up the server with

```
sudo puppet apply manifest.pp
```

Now you can initialize your django environment

```
./manage.py syncdb
./manage.py migrate
```

Now you should be able to run the server with:

```
./manage.py runserver 0.0.0.0:8000
```

and access it from your host machine's browser at

```
http://localhost:8000/admin
```