## Contents

1 Installation .............................................. 3
2 Source ................................................. 5
3 License .................................................. 7
4 Contents ................................................ 9
   4.1 Running IoT Relay .................................................. 9
   4.2 Plugins ....................................................... 9
   4.3 API Documentation .............................................. 11

Python Module Index ........................................ 15
Release 1.2.2

In greater and greater number, “Things” are capable of gathering data about their environment. These things have an interface to retrieve the measurements being taken but contain no way of pushing this data to the Internet. For example, home weather stations often contain only a USB interface and no network capability. Other devices may have network capability, such as ZigBee®, but still don’t have a direct way to send data to Internet connected hosts.

Internet of Things Relay is an application and framework for gathering data from sources and relaying it to destinations. It is somewhat like publish/subscribe except that it’s geared more toward devices that are unable to initiate a connection (they must be polled to get at their data).

IoT Relay provides basic setup and matches data sources with interested handlers. The rest of the work is left to plugins.
IoT Relay is available via PyPI.

```
$ pip install iotrelay
```

It is also necessary to create an (initially empty) ini-style file: `~/.iotrelay.cfg`.

```
[itorelay]
```

Now that IoT Relay is installed, add plugins!
The source for the IoT Relay project is hosted on GitHub. https://github.com/eman/iotrelay
License

The IoT Relay project is licensed under The BSD 2-Clause License.
4.1 Running IoT Relay

Start IoT Relay with the following command:

$ iotrelay

4.1.1 Usage

$ iotrelay --help

usage: iotrelay [-h] [-c CONFIG_FILE] [--log-level {debug,info,warning,info}]

Internet of Things Relay

optional arguments:
  -h, --help             show this help message and exit
  -c CONFIG_FILE, --config-file CONFIG_FILE
                         Configuration Filename
  --log-level {debug,info,warning,info}
                         Log Level

4.2 Plugins

Before IoT Relay can do anything useful, it needs plugins. There are plugin types: source and handler. Source plugins generate data. Handler plugins handle or do something with data that source plugins produce. These definitions are intended to be open-ended. Although IoT Relay was developed with the intention of relaying time-series type data between remote sources and remote destinations, a handler could instead view each datum as an event and trigger some action. Likewise, data source plugins do not have to simply pass the data they are collecting. They may process the data in some way before making it available to interested handlers.

4.2.1 Available Plugins

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotrelay-influxdb</td>
<td>A handler for sending data to InfluxDB</td>
</tr>
<tr>
<td>iotrelay-eagle</td>
<td>Pull data from an Eagle™ Home Energy Gateway</td>
</tr>
<tr>
<td>iotrelay-pywws</td>
<td>Pull weather data from a weather station via pywws</td>
</tr>
<tr>
<td>iotrelay-nest</td>
<td>Pull temperature and humidity from a Nest thermostat</td>
</tr>
</tbody>
</table>
4.2.2 Plugin Configuration

Each plugin will typically have its own configuration options. All plugins share the same .iotrelay.cfg configuration file. On plugin initialization, each plugin is provided with the options contained in its section.

Each plugin must at least be represented by a section in the configuration file. If no section exists for a given plugin, it is assumed to be disabled.

Plugin section names match the plugin name. For example, the following configuration file would enable (but not necessarily configure) the iotrelay-sample-source and iotrelay-sample-handler plugins.

```
; ~/.iotrelay.cfg

[iotrelay]

[iotrelay-sample-source]

[iotrelay-sample-handler]
```

Handler plugins must list the types of readings they’re interested in. If a reading_type for a handler plugin is left unset, that plugin will not receive any readings. The following example shows that the iotrelay-sample-handler is interested in receiving weather and power readings. Note that this handler may receive more than two time series with this configuration. Reading type represents a category of possible readings.

```
; ~/.iotrelay.cfg

[iotrelay]

[iotrelay-sample-handler]
reading types = weather, power, random
```

This example configuration file would cause the iotrelay-sample-handler to receive data from three reading types: weather, power, and random. Weather and power reading types are produced by the plugins referenced earlier: iotrelay-pywws and iotrelay-eagle. Readings of type random are produced by the data source sample plugin shown in the next section.

4.2.3 Data Source Sample Plugin

A data source definition is a class which provides a get_reading() method and a constructor which accepts a config parameter. The get_reading() method must return one or more instances of the Reading() class or None. In this example, create a file called iotrelay_sample_source.py and enter the following code.

```
# source/iotrelay_sample_source.py

import random
from iotrelay import Reading

class DataSource(object):
    def __init__(self, config):
        self.config = config

    def get_readings(self):
        value = random.randint(1, 100)
        return Reading(reading_type='random', value=value)
```

IoT Relay uses setuptools to find plugins registered in the iotrelay group. Data-sources should use the entry-point name source. The following configuration should be placed in setup.py and in the same directory as iotrelay_sample_source.py.
# source/setup.py

```python
from setuptools import setup

setup(name='iotrelay-sample-source',
      install_requires=['iotrelay'],
      py_modules=['iotrelay_sample_source'],
      entry_points={
        'iotrelay': ['source=iotrelay_sample_source:DataSource']
      })
```

Install the source plugin by typing:

```
$ python setup.py install
```

### 4.2.4 Data Handler Sample Plugin

Like the previous example, create a new directory with two files:

# handler/iotrelay_sample_handler.py

```python
class Handler(object):
    def __init__(self, config):
        self.config = config

    def set_reading(self, reading):
        print(reading)
```

# handler/setup.py

```python
from setuptools import setup

setup(name='iotrelay-sample-handler',
      install_requires=['iotrelay'],
      py_modules=['iotrelay_sample_handler'],
      entry_points={
        'iotrelay': ['handler=iotrelay_sample_handler:Handler']
      })
```

Install the handler plugin by typing:

```
$ python setup.py install
```

### 4.3 API Documentation

#### 4.3.1 Constants

`iotrelay.version`

The version number of the iotrelay module, as a string.
The configuration file name and path to use if one is not specified.

The setuptools group to inspect for available plugins.

### 4.3.2 Classes

**class iotrelay.Reading**

`iotrelay.Reading(reading_type, value, timestamp, series_key, tags)`

Reading provides a container for passing a datum, or “Reading”, between sources and handlers.

- **reading_type**
  - Represents a category of readings. For example, a weather station might produce temperature, rainfall, and wind speed. Because all of these are related to weather, they could be categorized with a reading type of “weather”. `reading_type` is used to match data sources with data handlers. If a data source generates readings with a `reading_type` of weather, data handlers that have registered an interest in weather will receive those readings.

- **value**
  - Contains the datum being communicated.

- **timestamp**
  - A `datetime` object containing the time stamp at which the reading was taken.

  If timestamp is not specified in the constructor, timestamp is set to the time the `Reading` object was created.

- **series_key**
  - Identifies an individual time series. A weather station may produce multiple data streams, one for each sensor. Each of these streams should have their own series key.

  If a `series_key` is not specified in the constructor, `series_key` is set to `reading_type`.

- **tags**
  - A dictionary of key value pairs, describing the reading.

**class iotrelay.DataSource**

`iotrelay.DataSource(config)`

DataSource is an abstract class for implementing data source plugins.

- **config**
  - A dict containing key/value pairs corresponding to options taken from the plugin’s section in iotrelay’s `config` file, `~/.iotrelay.cfg`.

**get_readings()**

Get readings from a data source.

- **Returns**
  - One or more Readings or no Reading

- **Return type**
  - `Reading`, an iterable of `Reading` instances, or None

Example Data Source:

```python
# source/iotrelay_sample_source.py

import random
from iotrelay import Reading

class DataSource(object):  
    def __init__(self, config):
        self.config = config
```
def get_readings(self):
    value = random.randint(1, 100)
    return Reading(reading_type='random', value=value)

class iotrelay.Handler(config):
    Handler is an abstract class for implementing data handler plugins.

    config
    A dict containing key/value pairs corresponding to options taken from the plugin’s section in iotrelay’s config file, ~/.iotrelay.cfg.

    set_reading(reading)
    Send a reading to a handler.

    Parameters reading(iotrelay.Reading) – The Reading instance being sent to the handler.

    flush()
    Optional: Flush any readings that have not been sent or otherwise processed.

Example Data Handler:

    # handler/iotrelay_sample_handler.py
    
    class Handler(object):
        def __init__(self, config):
            self.config = config

        def set_reading(self, reading):
            print(reading)
| iotrelay | 11 |
Index

C
config (iotrelay.DataSource attribute), 12
config (iotrelay.Handler attribute), 13

D
DataSource (class in iotrelay), 12
DEFAULT_CONFIG (in module iotrelay), 11

G
get_readings() (iotrelay.DataSource method), 12
GROUP (in module iotrelay), 12

H
Handler (class in iotrelay), 13
Handler.flush() (in module iotrelay), 13

I
iotrelay (module), 11

R
Reading (class in iotrelay), 12
reading_type (iotrelay.Reading attribute), 12

S
series_key (iotrelay.Reading attribute), 12
set_reading() (iotrelay.Handler method), 13

T
tags (iotrelay.Reading attribute), 12
timestamp (iotrelay.Reading attribute), 12

V
value (iotrelay.Reading attribute), 12
version (in module iotrelay), 11