ouimeaux Documentation

Release 0.7.2

Ian McCracken

July 31, 2016

Contents

Contents:

CHAPTER 1

ouimeaux

Open source control for Belkin WeMo devices

- Free software: BSD license
- Documentation: http://ouimeaux.rtfd.org.

1.1 Features

- Supports WeMo Switch, Light Switch, Insight Switch and Motion
- Command-line tool to discover and control devices in your environment
- REST API to obtain information and perform actions on devices
- Simple responsive Web app provides device control on mobile
- Python API to interact with device at a low level

CHAPTER 2

Installation

2.1 Basic

At the command line:

```
$ easy_install ouimeaux
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv ouimeaux
$ pip install ouimeaux
```

If you want to enable `wemo server` functionality, specify the `server` feature to install the necessary dependencies:

```
$ pip install ouimeaux[server]
```

2.2 Linux

ouimeaux requires Python header files to build some dependencies, and is installed normally using pip or easy_install.

Debian/Ubuntu:

```
sudo apt-get install python-setuptools python-dev
```

RHEL/CentOS/Fedora:

```
sudo yum -y install python-setuptools python-devel
```

If you wish to build from a local copy of the source, you can of course always execute:

```
python setup.py install
```

2.3 Windows

ouimeaux requires gevent version 1.0rc2 or higher. If you don't have the ability to compile gevent and greenlet (a sub-dependency) locally, you can find and download the binary installers for these packages here:

- gevent: https://github.com/SiteSupport/gevent/downloads
- greenlet: https://pypi.python.org/pypi/greenlet

CHAPTER 3

`wemo` Command

The `wemo` script will discover devices in your environment and turn switches on and off. To list devices:

```
$ wemo list
```

Default is to search for 5 seconds; you can pass `--timeout` to change that.

You can also print the status of every device found in your environment (the `-v` option is available to print on/off instead of 0/1):

```
$ wemo status
```

To turn a switch on and off, you first have to know the name. Then:

```
$ wemo switch "TV Room" on
$ wemo switch "TV Room" off
```

You can also toggle the device:

```
$ wemo switch "TV Room" toggle
```

Or check its current status (the `-v` option will print the word on/off instead of 0/1):

```
$ wemo -v switch "TV Room" status
on
```

WeMo LED Bulbs are supported on the command line as well. Control them like switches with `wemo light`:

```
$ wemo light lamp on
```

Or set them to a dimness level from 1 to 255:

```
$ wemo light lamp on 45
```

The `wemo` script will do fuzzy matching of the name you pass in (this can be disabled with the `-e` option):

```
$ wemo switch tvrm on
```

Aliases configured in the file will be accessible on the command line as well:

```
aliases:
    tv: TV Room Lights

$ wemo switch tv on
```

Note: If an alias is used on the command line, fuzzy matching will not be attempted.

You can also clear the device cache from the command line:

```
$ wemo clear
```

The `wemo` script will obey configured settings; they can also be overridden on the command line:

-b, --bind IP:PORT Bind to this host and port when listening for responses

-d, --debug Enable debug logging to stdout

-e, --exact-match Disable fuzzy matching

-f, --no-cache Disable the device cache

-v, --human-readable Print statuses as human-readable words

CHAPTER 4

Server

`wemo server` starts a process serving up both a Web app providing basic device control and a REST API allowing integration with any number of services.

The necessary dependencies to run the server are not installed unless the `server` feature is specified at installation:

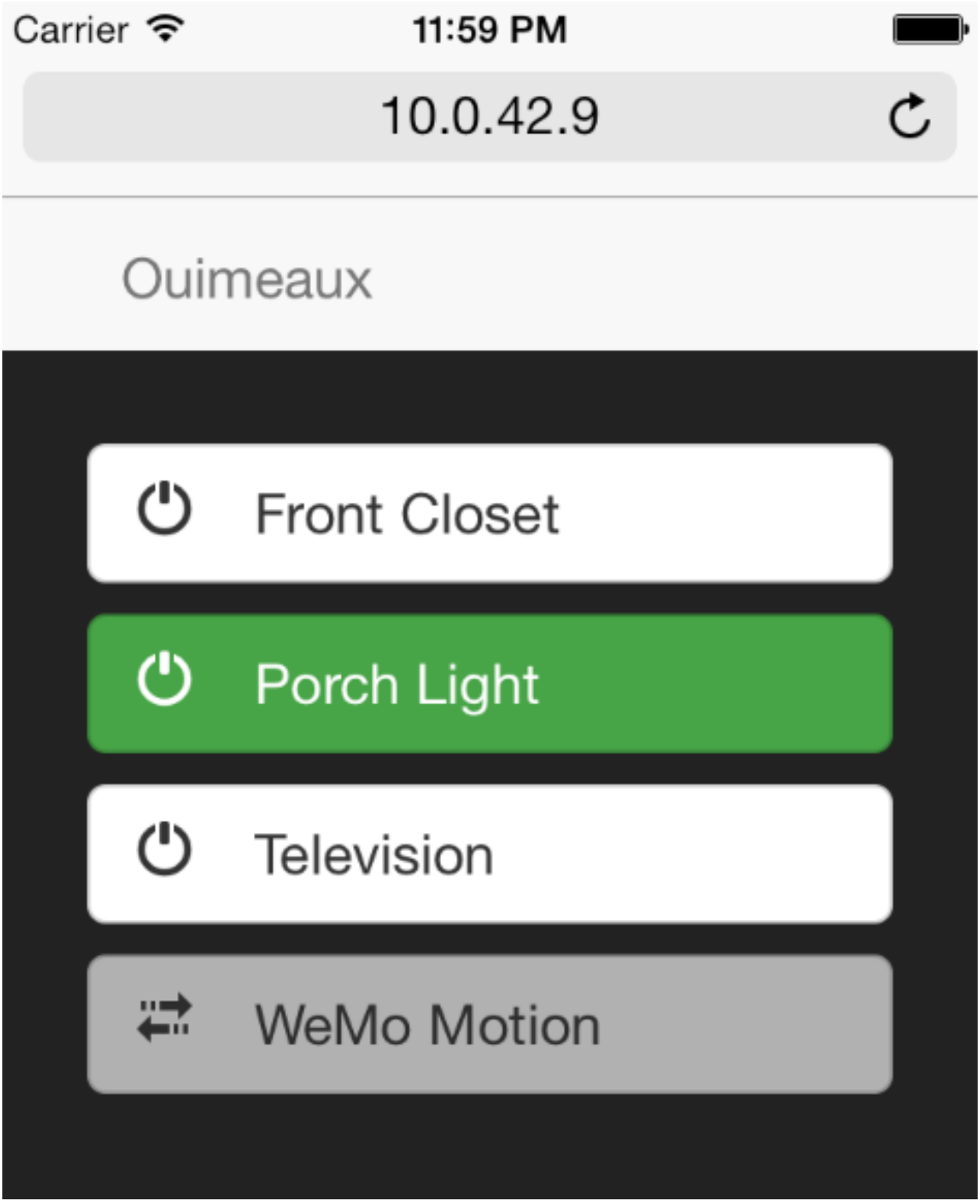
```
$ pip install ouimeaux[server]
```

4.1 Configuration

The IP and port to which the server will bind are specified by the `listen` parameter in the configuration file. Default: `0.0.0.0:5000`.

4.2 Web App

The Web app very simply presents buttons allowing control of devices and indicating current state. Motions appear as disabled buttons but will turn green when activated.


Carrier
11:59 PM
10.0.42.9
Ouimeaux
Front Closet
Porch Light
Television
WeMo Motion

4.3 REST API

A vaguely RESTful API is provided to control devices. Arguments, where specified, may be passed either as query arguments or as JSON:

```
curl -X POST http://localhost:5000/api/environment -d '{"seconds":10}'
```

is as valid as:

```
curl -X POST http://localhost:5000/api/environment?seconds=10
```

Resource	Description
GET /api/environment	Returns a JSON description of all devices in the environment
POST /api/environment	Initiates a discovery of the environment. Optional `seconds` argument (default: 5) determines length of discovery.
GET /api/device/NAME	Returns a JSON description of the device named NAME. NAME will be fuzzy-matched against device names, as on the command line (e.g., "closet" will match "Hall Closet").
POST /api/device/NAME	Toggle switch state, specified by optional `state` argument (default: "toggle"). Valid values are "on", "off" or "toggle".

CHAPTER 5

Configuration

A configuration file in YAML format will be created at `~/.wemo/config.yml`:

```
# ip:port to bind to when receiving responses from discovery.
# The default is first DNS resolution of local host, port 54321
#
# bind: 10.1.2.3:9090

# Whether to use a device cache (stored at ~/.wemo/cache)
#
# cache: false

aliases:
# Shortcuts to longer device names. Uncommenting the following
# line will allow you to execute 'wemo switch lr on' instead of
# 'wemo switch "Living Room Lights" on'
#
#    lr: Living Room Lights

# Web app bind address
#
# listen: 0.0.0.0:5000
```

CHAPTER 6

Python API

6.1 Environment

The main interface is presented by an `Environment`, which optionally accepts functions called when a Switch or Motion device is identified:

```
>>> from ouimeaux.environment import Environment
>>>
>>> def on_switch(switch):
...     print "Switch found!", switch.name
...
>>> def on_motion(motion):
...     print "Motion found!", motion.name
...
>>> env = Environment(on_switch, on_motion)
```

Start up the server to listen for responses to the discovery broadcast:

```
>>> env.start()
```

Discovery of all WeMo devices in an environment is then straightforward; simply pass the length of time (in seconds) you want discovery to run:

```
>>> env.discover(seconds=3)
Switch found! Living Room
```

During that time, the `Environment` will continually broadcast search requests and parse responses. At any point, you can see the names of discovered devices:

```
>>> env.list_switches()
['Living Room', 'TV Room', 'Front Closet']
>>> env.list_motions()
['Front Hallway']
```

Devices can be retrieved by using `get_switch` and `get_motion` methods:

```
>>> switch = env.get_switch('TV Room')
>>> switch
<WeMo Switch "TV Room">
```

6.2 Devices

All devices have an `explain()` method, which will print out a list of all available services, as well as the actions and arguments to those actions on each service:

```
>>> switch.explain()

basicevent
----------
  SetSmartDevInfo(SmartDevURL)
  SetServerEnvironment(ServerEnvironmentType, TurnServerEnvironment, ServerEnvironment)
  GetDeviceId()
  GetRuleOverrideStatus(RuleOverrideStatus)
  GetIconURL(URL)
  SetBinaryState(BinaryState)
...
```

Services and actions are available via simple attribute access. Calling actions returns a dictionary of return values:

```
>>> switch.basicevent.SetBinaryState(BinaryState=0)
{'BinaryState': 0}
```

6.3 Events

Warning: This events framework is deprecated and will be removed prior to the 1.0 release. Please use the signals framework.

By default, ouimeaux subscribes to property change events on discovered devices (this can be disabled by passing `with_subscribers=False` to the `Environment` constructor). You can register callbacks that will be called when switches and motions change state (on/off, or motion detected):

```
>>> def on_motion(value):
...     print "Motion detected!"
...
>>> env.get_motion('Front Hallway').register_listener(on_motion)
>>> env.wait(timeout=60)
```

Note the use of `Environment.wait()` to give control to the event loop for events to be detected. A timeout in seconds may optionally be specified; default is no timeout.

6.4 Signals

A simple signals framework (using pysignals) is included to replace the rudimentary events in earlier releases. These are found in the *ouimeaux.signals* module. Signal handlers may be registered using the `receiver` decorator and must have the signature `sender, **kwargs`:

```
@receiver(devicefound)
def handler(sender, **kwargs):
    print "Found device", sender
```

Where `sender` is the relevant object (in most cases, the device). Signals may also have handlers registered using `signal.connect(handler)`:

```
def handler(sender, **kwargs):
    pass

statechange.connect(sender)
```

Available signals:

discovered

Fires when a device responds to the broadcast request. Includes:

- `sender`: The UPnP broadcast component
- `address`: The address of the responding device
- `headers`: The response headers

devicefound

Sent when a device is found and registered into the environment. Includes:

- `sender`: The device found

subscription

Sent when a device sends an event as the result of a subscription. Includes:

- `sender`: The device that sent the event
- `type`: The type of the event send (e.g., `BinaryState`)
- `value`: The value associated with the event

statechange

Sent when a device indicates it has detected a state change. Includes:

- `sender`: The device that changed state
- `state`: The resulting state (0 or 1)

See the pysignals documentation for further information.

Example: Registering a handler for when a Light Switch switches on or off:

```
from ouimeaux.signals import statechange, receiver

env = Environment(); env.start()
env.discover(5)

switch = env.get_switch('Porch Light')

@receiver(statechange, sender=switch)
def switch_toggle(device, **kwargs):
    print device, kwargs['state']

env.wait()  # Pass control to the event loop
```

See the examples for a more detailed implementation.

6.5 Switches

Switches have three shortcut methods defined: `get_state`, `on` and `off`. Switches also have a `blink` method, which accepts a number of seconds. This will toggle the device, wait the number of seconds, then toggle it again. Remember to call `env.wait()` to give control to the event loop.

6.6 Motions

Motions have one shortcut method defined: `get_state`.

6.7 Insight

In addition to the normal Switch methods, Insight switches have several metrics exposed:

```
insight.today_kwh
insight.current_power
insight.today_on_time
insight.on_for
insight.today_standby_time
```

6.8 Device Cache

By default, device results are cached on the filesystem for quicker initialization. This can be disabled by passing `with_cache=False` to the `Environment` constructor. On a related note, if you want to use the cache exclusively, you can pass `with_discovery=False` to the `Environment` constructor to disable M-SEARCH requests.

You can clear the device cache either by deleting the file `~/.wemo/cache` or by using the `wemo clear` command.

6.9 Examples

Detailed examples are included in the source demonstrating common use cases. Suggestions (or implementations) for more are always welcome.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/iancmcc/ouimeaux/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

7.1.4 Write Documentation

ouimeaux could always use more documentation, whether as part of the official ouimeaux docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/iancmcc/ouimeaux/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *ouimeaux* for local development.

1. Fork the *ouimeaux* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ouimeaux.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ouimeaux
$ cd ouimeaux/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ouimeaux tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/iancmcc/ouimeaux/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_ouimeaux
```

CHAPTER 8

Credits

8.1 Development Lead

- Ian McCracken <ian.mccracken@gmail.com>

8.2 Contributors

- Aron Parsons
- Brian Peiris
- nschrenk
- Gabriel M. Schuyler
- Markus Stenberg
- aktur
- Justin Eldridge
- logjames
- jgstew
- Bob Gardner
- FRITZ|FRITZ

CHAPTER 9

History

9.1 Release 0.8.0 (July 30, 2016)

- Randomize subscription ports to enable simultaneous ouimeaux scripts (thanks @bennytheshap)
- Fix for WeMo LED Light support (thanks @sstangle73)
- #32: Removed address cache, broke server out into optional feature
- Fix for Maker state reporting (thanks @pavoni)
- Filter by SSDP location, fixing case where multiple devices respond from the same IP (thanks @szakharchenko)
- Fix Maker event handlers, which were being passed as bridges (thanks @maxlazarov)
- Work around gevent-socketio bug by explicitly casting header value as string
- Fix for inconsistent Light state (thanks @canduuk)
- StateChange signals are now a separate class and do not fire if value is unchanged (thanks @esecules)
- Python 3 support (thanks to @drock371)

9.2 Release 0.7.9 (March 17, 2015)

- Command line support for WeMo LED Light (thanks @fritz-fritz)
- Command line support for WeMo Maker (thanks @logjames)
- Support for 2.0.0 firmware (thanks @fritz-fritz)
- Bug fixes

9.3 Release 0.7.3 (August 10, 2014)

- Fixed #18: Error when run as root
- Fixed #26: Evict devices from cache when unreachable
- Fixed #29: GetPower stopped working for Insight devices
- Fixed #31: Add blink method on switches, include in REST API
- Fixed #33, #37: Handle invalid devices without dying

- Fixed #35: Require requests >= 2.3.0
- Fixed #40: Retry requests in the event of failure
- **Fixed #47: Don't choke on invalid newlines in XML returned by switches** (thanks to @fingon)

9.4 Release 0.7.2 (January 28, 2014)

- Fix a bug with using query parameters on /api/device

9.5 Release 0.7 (January 27, 2014)

- Added REST API
- Added Web app

9.6 Release 0.6 (January 25, 2014)

- Added signals framework
- Fixed #16, #19, #22: Defensively resubscribe to events when device responds with an error
- Fixed #15: Signals framework includes relevant device when sending signal
- Refactored structure, added Sphinx docs

9.7 Release 0.5.3 (January 25, 2014)

- Fixed #20: Allow timeout in environment.wait()
- Fixed #21: Add Insight support

9.8 Release 0.5.2 (November 23, 2013)

- Fixed #14: Indicate Connection:close header to avoid logging when WeMo sends invalid HTTP response.

9.9 Release 0.5.1 (November 9, 2013)

- Fixed #10: Updated subscriber listener to use more reliable method of retrieving non-loopback IP address; updated docs to fix typo in listener registration example (thanks to @benhoyle, @francxk)
- Fixed #11: Remove instancemethod objects before attempting to pickle devices in the cache (thanks @piperde, @JonPenner, @tomtomau, @masilu77)

9.10 Release 0.5 (October 14, 2013)

- Added fuzzy matching of device name when searching/toggling from command line
- Added `status` mode to print status for all devices
- Added `switch status` mode to print status for specific device
- Added flags for all command-line options
- Fixed #9: Removed unused fcntl import that precluded Windows usage (thanks to @deepseven)

9.11 Release 0.4.3 (August 31, 2013)

- Used new method of obtaining local IP for discovery that is less likely to return loopback
- Exit with failure and instructions for solution if loopback IP is used
- Updated installation docs to include python-dev and pip instructions (patch by @fnaard)
- Fixed README inclusion bug that occasionally broke installation via pip.
- Added `--debug` option to enable debug logging to stdout

9.12 Release 0.4 (August 17, 2013)

- Fixed #7: Added support for light switch devices (patch by nschrenk).
- Fixed #6: Added "wemo clear" command to clear the device cache.

9.13 Release 0.3 (May 25, 2013)

- Fixed #4: Added ability to specify ip:port for discovery server binding. Removed documentation describing need to disable SSDP service on Windows.
- Fixed #5: Added device cache for faster results.
- Added configuration file.
- Added ability to configure aliases for devices to avoid quoting strings on the command line.
- Added 'toggle' command to command line switch control.

9.14 Release 0.2 (April 21, 2013)

- Fixed #1: Added ability to subscribe to motion and switch state change events.
- Added Windows installation details to README (patch by @brianpeiris)
- Cleaned up UDP server lifecycle so rediscovery doesn't try to start it back up.

9.15 Release 0.1 (February 2, 2013)

- Initial release.
- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search