# fauxmo Documentation

*Release v0.5.0*

**Nathan Henrie**

**Mar 30, 2020**

# Contents

Contents:

CHAPTER 1

# Fauxmo README

master: master branch build status dev: dev branch build status

Python 3 module that emulates Belkin WeMo devices for use with the Amazon Echo.

Originally forked from https://github.com/makermusings/fauxmo, unforked to enable GitHub code search (which currently doesn't work in a fork), and because the libraries have diverged substantially.

- Documentation: fauxmo.readthedocs.org

## 1.1 Introduction

The Amazon Echo is able to control certain types of home automation devices by voice. Fauxmo provides emulated Belkin Wemo devices that the Echo can turn on and off by voice, locally, and with minimal lag time. Currently these Fauxmo devices can be configured to make requests to an HTTP server or to a Home Assistant instance via its Python API and only require a JSON config file for setup.

As of version v0.4.0, Fauxmo uses several API features and f-strings that require Python 3.6+. I highly recommend looking into pyenv if you're currently on an older Python version and willing to upgrade. Otherwise, check out the FAQ section at the bottom for tips on installing an older Fauxmo version (though note that I will not be continuing development or support for older versions).

For what it's worth, if you're concerned about installing pyenv on a low-resource machine like the Raspberry Pi, I encourage you to review my notes on the size and time required to install Python 3.6 with pyenv on a Raspberry Pi and the nontrivial improvement in speed (with a simple pystone benchmark) using an optimized pyenv-installed 3.6 as compared to the default Raspbian 3.5.3.

## 1.2 Terminology

faux (`\fō\`): imitation

WeMo: Belkin home automation product with which the Amazon Echo can interface

Fauxmo (`\fō-mō\`): Python 3 module that emulates Belkin WeMo devices for use with the Amazon Echo.

Fauxmo has a server component that helps register "devices" with the Echo (which may be referred to as the Fauxmo server or Fauxmo core). These devices are then exposed individually, each requiring its own port, and may be referred to as a Fauxmo device or a Fauxmo instance. The Echo interacts with each Fauxmo device as if it were a separate WeMo device.

## 1.3 Usage

Installation into a venv is *highly recommended*, especially since it's baked into the recent Python versions that Fauxmo requires.

Additionally, please ensure you're using a recent version of pip (>= 9.0.1) prior to installation: `pip install --upgrade pip`

### 1.3.1 Simple install: From PyPI

1. `python3 -m venv .venv`
2. `source ./.venv/bin/activate`
3. `python3 -m pip install fauxmo`
4. Make a `config.json` based on `config-sample.json`
5. `fauxmo -c config.json [-v]`

### 1.3.2 Simple install of dev branch from GitHub

This is a good strategy for testing features in development – for actually contributing to development, clone the repo as per below)

1. `python3 -m venv .venv`
2. `source ./.venv/bin/activate`
3. `pip install [-e] git+https://github.com/n8henrie/fauxmo.git@dev`

### 1.3.3 Install for development from GitHub

1. `git clone https://github.com/n8henrie/fauxmo.git`
2. `cd fauxmo`
3. `python3 -m venv .venv`
4. `source ./.venv/bin/activate`
5. `pip install -e .[dev]`
6. `cp config-sample.json config.json`
7. Edit `config.json`
8. `fauxmo [-v]`

### 1.3.4 Set up the Echo

1. Open the Amazon Alexa webapp to the Smart Home page
2. **With Fauxmo running**, click "Discover devices" (or tell Alexa to "find connected devices")
3. Ensure that your Fauxmo devices were discovered and appear with their names in the web interface
4. Test: "Alexa, turn on [the kitchen light]"

### 1.3.5 Set Fauxmo to run automatically in the background

NB: As discussed in #20, the example files in `extras/` are *not* included when you install from PyPI* (using `pip`). If you want to use them, you either need to clone the repo or you can download them individually using tools like `wget` or `curl` by navigating to the file in your web browser, clicking the `Raw` button, and using the resulting URL in your address bar.

* As of Fauxmo v0.4.0 `extras/` has been added to `MANIFEST.in` and may be included somewhere depending on installation from the `.tar.gz` vs `whl` format – if you can't find them, you should probably just get the files manually as described above.

#### systemd (e.g. Raspbian Jessie)

1. Recommended: add an unprivileged user to run Fauxmo: `sudo useradd -r -s /bin/false fauxmo`
   - NB: Fauxmo may require root privileges if you're using ports below 1024
2. `sudo cp extras/fauxmo.service /etc/systemd/system/fauxmo.service`
3. Edit the paths in `/etc/systemd/system/fauxmo.service`
4. `sudo systemctl enable fauxmo.service`
5. `sudo systemctl start fauxmo.service`

#### launchd (OS X)

1. `cp extras/com.n8henrie.fauxmo.plist ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
2. Edit the paths in `~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
   - You can remove the `StandardOutPath` and `StandardErrorPath` sections if desired
3. `launchctl load ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
4. `launchctl start com.n8henrie.fauxmo`

## 1.4 Plugins

Plugins are small user-extendible classes that allow users to easily make their own actions for Fauxmo to run by way of Alexa commands. They were previously called Handlers and may be referred to as such in places in the code and documentation.

Fauxmo v0.4.0 implements a new and breaking change in the way Handlers were implemented in previous versions, which requires modification of the `config.json` file (as described below).

A few plugins and the ABC from which the plugins are required to inherit may be included and installed by default in the `fauxmo.plugins` package. Any pre-installed plugins, like the rest of the core Fauxmo code, have no third party dependencies.

So far, the pre-installed plugins include:

- `fauxmo.plugins.simplehttpplugin.SimpleHTTPPlugin`
- `fauxmo.plugins.commandlineplugin.CommandLinePlugin`
- `fauxmo.plugins.homeassistantplugin.HomeAssistantPlugin`

`SimpleHTTPPlugin` responds to Alexa's `on` and `off` commands by making requests to URL endpoints by way of `urllib`. Example uses cases relevant to the IOT community might be a Flask server served from localhost that provides a nice web interface for toggling switches, whose endpoints could be added as the `on_cmd` and `off_cmd` args to a `SimpleHTTPPlugin` instance to allow activation by way of Alexa -> Fauxmo.

As of Fauxmo v0.4.5, the `FauxmoPlugin` abstract base class (and therefore all derivate Fauxmo plugins) requires a `get_state` method, which tells Alexa a device's state. If you don't have a way to determine devices state, you can just have your `get_state` method return `"unknown"`, but please review the notes on `get_state` below.

Also, see details regarding plugin configuration in each class's docstring, which I intend to continue as a convention for Fauxmo plugins. Users hoping to make more complicated requests may be interested in looking at `RESTAPIPlugin` in the `fauxmo-plugins repository`, which uses Requests for a much friendlier API.

## 1.4.1 User plugins

Users can easily create their own plugins, which is the motivation behind most of the changes in Fauxmo v0.4.0.

To get started:

1. Decide on a name for your plugin class. I highly recommend something descriptive, CamelCase and a `Plugin` suffix, e.g. `FooSwitcherPlugin`.
2. I strongly recommend naming your module the same as the plugin, but in all lower case, e.g. `fooswitcherplugin.py`.
3. Note the path to your plugin, which will need to be included in your `config.json` as `path` (absolute path recommended, ~ for homedir is okay).
4. Write your class, which must at minimum:
    - inherit from `fauxmo.plugins.FauxmoPlugin`.
    - provide the methods `on()`, `off()`, and `get_state()`.
        - Please note that unless the Echo has a way to determine the device state, it will likely respond that your "device is not responding" after you turn a device on (or in some cases off, or both), but it should still be able to switch the device.
        - If you want to ignore the actual device's state and just return the last successful action as the current state (e.g. if `device.on()` succeeded then return `"on"`), your plugin can return `super().get_state()` as its `get_state()` method. Some of the included plugins can be configured to have this behavior using a `use_fake_state` flag in their configuration (please look at the documentation and source code of the plugins for further details). Note that this means it won't update to reflect state changes that occur outside of Fauxmo (e.g. manually flipping a switch, or toggling with a different program), whereas a proper `get_state` implementation may be able to do so.
5. Any required settings will be read from your `config.json` and passed into your plugin as kwargs at initialization, see below.

In addition to the above, if you intend to share your plugin with others, I strongly recommend that you:

- Include generous documentation as a module level docstring.
- Note specific versions of any dependencies in that docstring.
  - Because these user plugins are kind of "side-loaded," you will need to manually install their dependencies into the appropriate environment, so it's important to let other users know exactly what versions you use.

Be aware, when fauxmo loads a plugin, it will add the directory containing the plugin to the Python path, so any other Python modules in this directory might be loaded by unscrupulous code. This behavior was adopted in part to facilitate installing any plugin dependencies in a way that will be available for import (e.g. `cd "$MYPLUGINPATH"; pip install -t $MYPLUGINDEPS`).

### 1.4.2 Notable plugin examples

NB: You may need to *manually* install additional dependencies for these to work – look for the dependencies in the module level docstring.

- https://github.com/n8henrie/fauxmo-plugins
  - `RESTAPIPlugin`
    - Trigger HTTP requests with your Echo.
    - Similar to `SimpleHTTPPlugin`, but uses Requests for a simpler API and easier modification.
  - `MQTTPlugin`
    - Trigger MQTT events with your Echo
  - User contributions of interesting plugins are more than welcome!

## 1.5 Configuration

I recommend that you copy and modify `config-sample.json`. Fauxmo will use whatever config file you specify with `-c` or will search for `config.json` in the current directory, `~/.fauxmo/`, and `/etc/fauxmo/` (in that order). The minimal configuration settings are:

- `FAUXMO`: General Fauxmo settings
  - `ip_address`: Optional[str] - Manually set the server's IP address. Recommended value: `"auto"`.
- `PLUGINS`: Top level key for your plugins, values should be a dictionary of (likely CamelCase) class names, spelled identically to the plugin class, with each plugin's settings as a subdictionary.
  - `ExamplePlugin`: Your plugin class name here, case sensitive.
    - `path`: The absolute path to the Python file in which the plugin class is defined (please see the section on user plugins above). Required for user plugins / plugins not pre-installed in the `fauxmo.plugins` subpackage.
    - `example_var1`: For convenience and to avoid redundancy, your plugin class can *optionally* use config variables at this level that will be shared for all `DEVICES` listed in the next section (e.g. an api key that would be shared for all devices of this plugin type). If provided, your plugin class must consume this variable in a custom `__init__`.
    - `DEVICES`: List of devices that will employ `ExamplePlugin`
      - `name`: Optional[str] – Name for this device. Optional in the sense that you can leave it out of the config as long as you set it in your plugin code as the `_name` attribute, but it does need to be

set somewhere. If you omit it from config you will also need to override the `__init__` method, which expects a `name` kwarg.

  - `port`: Optional[int] – Port that Echo will use connect to device. Should be different for each device, Fauxmo will attempt to set automatically if absent from config. NB: Like `name`, you can choose to set manually in your plugin code by overriding the `_port` attribute (and the `__init__` method, which expects a `port` kwarg otherwise).
  - `example_var2`: Config variables for individual Fauxmo devices can go here if needed (e.g. the URL that should be triggered when a device is activated). Again, your plugin class will need to consume them in a custom `__init__`.

Each user plugin should describe its required configuration in its module-level docstring. The only required config variables for all plugins is `DEVICES`, which is a `List[dict]` of configuration variables for each device of that plugin type. Under `DEVICES` it is a good idea to set a fixed, high, free `port` for each device, but if you don't set one, Fauxmo will try to pick a reasonable port automatically (though it will change for each run).

Please see `config-sample` for a more concrete idea of the structure of the config file, using the built-in `SimpleHTTPPlugin` for demonstration purposes. Below is a description of the kwargs that `SimpleHTTPPlugin` accepts.

- `name`: What you want to call the device (how to activate by Echo)
- `port`: Port the Fauxmo device will run on
- `on_cmd`: str – URL that should be requested to turn device on.
- `off_cmd`: str – URL that should be requested to turn device off.
- `state_cmd`: str – URL that should be requested to query device state
- `method` / `state_method`: Optional[str] = GET – GET, POST, PUT, etc.
- `headers`: Optional[dict] – Extra headers
- `on_data` / `off_data` / `state_data`: Optional[dict] – POST data
- `state_response_on` / `state_response_off`: str – If this string is in contained in the response from `state_cmd`, then the devices is `on` or `off`, respectively
- `user` / `password`: Optional[str] – Enables HTTP authentication (basic or digest only)
- `use_fake_state`: Optional[bool] – If `True`, override the plugin's `get_state` method to return the latest successful action as the device state. NB: The proper json boolean value for Python's `True` is `true`, not `True` or `"true"`.

## 1.6 Security

I am not a technology professional and make no promises regarding the security of this software. Specifically, plugins such as `CommandLinePlugin` execute arbitrary code from your configuration without any validation. If your configuration can be tampered with, you're in for a bad time.

That said, if your configuration can be tampered with (i.e. someone already has write access on your machine), then you likely have bigger problems.

Regardless, a few reasonable precautions that I recommend:

- run `fauxmo` in a virtulaenv, even without any dependencies
- run `fauxmo` as a dedicated unprivileged user with its own group

- remove write access from the `fauxmo` user and group for your config file and any plugin files (perhaps `chmod 0640 config.json; chown me:fauxmo config.json`)
- consider using a firewall like `ufw`, but don't forget that you'll need to open up ports for upnp (`1900`, UDP) and ports for all your devices that you've configured (in `config.json`).

For example, if I had 4 echo devices at 192.168.1.5, 192.168.1.10, 192.168.1.15, and 192.168.1.20, and Fauxmo was configured with devices at each of port 12345-12350, to configure `ufw` I might run something like:

```
$ for ip in 5 10 15 20; do
    sudo ufw allow \
        from 192.168.1."$ip" \
        to any \
        port 1900 \
        proto udp \
        comment "fauxmo upnp"
    sudo ufw allow \
        from 192.168.1."$ip" \
        to any \
        port 12345:12350 \
        proto tcp \
        comment "fauxmo devices"
done
```

You use Fauxmo at your own risk, with or without user plugins.

## 1.7 Troubleshooting / FAQ

Your first step in troubleshooting should probably be to "forget all devices" (which as been removed from the iOS app but is still available at alexa.amazon.com), re-discover devices, and make sure to refresh your device list (e.g. pull down on the "devices" tab in the iOS app, or just close out the app completely and re-open).

- How can I increase my logging verbosity?
  - `-v[vv]`
  - `-vv` (`logging.INFO`) is a good place to start when debugging
- How can I ensure my config is valid JSON?
  - `python -m json.tool < config.json`
  - Use `jsonlint` or one of numerous online tools
- How can I install an older / specific version of Fauxmo?
  - Install from a tag:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@v0.1.11`
  - Install from a specific commit:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@d877c513ad45cbbbd77b1b83e7a2f03bf0004856`
- Where can I get more information on how the Echo interacts with devices like Fauxmo?
  - Check out `protocol_notes.md`
- Does Fauxmo work with non-Echo emulators like Alexa AVS or Echoism.io?
  - Apparently not.

- How do I find my Echo firmware version?
  - https://alexa.amazon.com -> Settings -> [Device Name] -> Device Software Version

### 1.7.1 Installing Python 3.7 with pyenv

```
sudo install -o $(whoami) -g $(whoami) -d /opt/pyenv
git clone https://github.com/pyenv/pyenv /opt/pyenv
cat <<'EOF' >> ~/.bashrc
export PYENV_ROOT="/opt/pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
EOF
source ~/.bashrc
pyenv install 3.7.3
```

You can then install Fauxmo into Python 3.7 in a few ways, including:

```
# Install with pip
"$(pyenv root)"/versions/3.7.3/bin/python3.7 -m pip install fauxmo

# Show full path to Fauxmo console script
pyenv which fauxmo

# Run with included console script
fauxmo -c /path/to/config.json -vvv

# I recommend using the full path for use in start scripts (e.g. systemd, cron)
"$(pyenv root)"/versions/3.7.3/bin/fauxmo -c /path/to/config.json -vvv

# Alternatively, this also works (after `pip install`)
"$(pyenv root)"/versions/3.7.3/bin/python3.7 -m fauxmo.cli -c config.json -vvv
```

## 1.8 Buy Me a Coffee

## 1.9 Acknowledgements / Reading List

- Tremendous thanks to @makermusings for the original version of Fauxmo!
  - Also thanks to @DoWhileGeek for commits towards Python 3 compatibility
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation
- http://www.makermusings.com/2015/07/18/virtual-wemo-code-for-amazon-echo
- http://hackaday.com/2015/07/16/how-to-make-amazon-echo-control-fake-wemo-devices
- https://developer.amazon.com/appsandservices/solutions/alexa/alexa-skills-kit
- https://en.wikipedia.org/wiki/Universal_Plug_and_Play
- http://www.makermusings.com/2015/07/19/home-automation-with-amazon-echo-apps-part-1
- http://www.makermusings.com/2015/08/22/home-automation-with-amazon-echo-apps-part-2

- https://www.rilhia.com/tutorials/using-upnp-enabled-devices-talend-belkin-wemo-switch

CHAPTER 2

# fauxmo

## 2.1 fauxmo package

### 2.1.1 Subpackages

#### fauxmo.plugins package

#### Submodules

#### fauxmo.plugins.commandlineplugin module

Fauxmo plugin that runs a command on the local machine.

Runs a *shlex‘ed command using ‘subprocess.run*, keeping the default of *shell=False*. This is probaby frought with security concerns, which is why this plugin is not included by default in *fauxmo.plugins*. By installing or using it, you acknowledge that it could run commands from your config.json that could lead to data compromise, corruption, loss, etc. Consider making your config.json read-only. If there are parts of this you don’t understand, you should probably not use this plugin.

If the command runs with a return code of 0, Alexa should respond prompty “Okay” or something that indicates it seems to have worked. If the command has a return code of anything other than 0, Alexa stalls for several seconds and subsequently reports that there was a problem (which should notify the user that something didn’t go as planned).

Note that *subprocess.run* as implemented in this plugin doesn’t handle complex commands with pipes, redirection, or multiple statements joined by *&&*, ||, *;*, etc., so you can’t just use e.g. *“command that sometimes fails* || *true”* to avoid the delay and Alexa’s response. If you really want to handle more complex commands, consider using this plugin as a template for another one using *os.system* instead of *subprocess.run*, but realize that this comes with substantial security risks that exceed my ability to explain.

Example config: “‘ {

> **“FAUXMO”: {** “ip_address”: “auto”
>
> }, “PLUGINS”: {

**"CommandLinePlugin": {** "path": "/path/to/commandlineplugin.py", "DEVICES": [

**{** "name": "output stuff to a file", "port": 49915, "on_cmd": "touch testfile.txt", "off_cmd": "rm testfile.txt", "state_cmd": "ls testfile.txt"

}, {

"name": "command with fake state", "port": 49916, "on_cmd": "touch test-file.txt", "off_cmd": "rm testfile.txt", "use_fake_state": true

}

]

}

}

**}**

**class** `fauxmo.plugins.commandlineplugin.`**`CommandLinePlugin`**(*name: str*, *port: int*, *on_cmd: str*, *off_cmd: str*, *state_cmd: str = None*, *use_fake_state: bool = False*)

Bases: *`fauxmo.plugins.FauxmoPlugin`*

Fauxmo Plugin for running commands on the local machine.

**`__init__`**(*name: str*, *port: int*, *on_cmd: str*, *off_cmd: str*, *state_cmd: str = None*, *use_fake_state: bool = False*) → None

Initialize a CommandLinePlugin instance.

**Parameters**

- **`name`** – Name for this Fauxmo device
- **`port`** – Port on which to run a specific CommandLinePlugin instance
- **`on_cmd`** – Command to be called when turning device on
- **`off_cmd`** – Command to be called when turning device off
- **`state_cmd`** – Command to check device state (return code 0 == on)
- **`use_fake_state`** – If *True*, override *get_state* to return the latest action as the device state. NB: The proper json boolean value for Python's *True* is *true*, not *True* or *"true"*.

**`get_state`**() → str

Get device state.

NB: Return code of *0* (i.e. ran without error) indicates "on" state, otherwise will be off. making it easier to have something like *ls path/to/pidfile* suggest *on*. Many command line switches may not actually have a "state" per se (just an arbitary command you want to run), in which case you could just put "false" as the command, which should always return "off".

**Returns** "on" or "off" if *state_cmd* is defined, "unknown" if undefined

**`off`**() → bool

Run off command.

**Returns** True if command seems to have run without error.

**on**() → bool
Run on command.

**Returns** True if command seems to have run without error.

**run_cmd**(*cmd: str*) → bool
Partialmethod to run command.

**Parameters** **cmd** – Command to be run

**Returns** True if command seems to have run without error

## fauxmo.plugins.homeassistantplugin module

Fauxmo plugin to interact with Home Assistant devices.

One simple way to find your entity_id is to use curl and pipe to grep or jq. Note that modern versions of home-assistant require you to create and include a long-lived access token, which you can generate in the web interface at the */profile* endpoint.

curl –silent –header "Authorization: Bearer YourTokenHere" http://IP:PORT/api/states | jq

NB: This is just a special case of the RESTAPIPlugin (or even SimpleHTTPPlugin, see *config-sample.json* in the main Fauxmo repo), but it makes config substantially easier by not having to redundantly specify headers and endpoints.

Install to Fauxmo by downloading or cloning and including in your Fauxmo config. One easy way to make a long-lived access token is by using the frontend and going to the */profile* endpoint, scroll to the bottom. Documentation on the long-lived tokens is available at https://developers.home-assistant.io/docs/en/auth_api.html#long-lived-access-token

Example config: "' {

**"FAUXMO": {** "ip_address": "auto"

}, "PLUGINS": {

**"HomeAssistantPlugin": {** "ha_host": "192.168.0.50", "ha_port": 8123, "ha_protocol": "http", "ha_token": "abc123", "path": "/path/to/homeassistantplugin.py", "DEVICES": [

**{** "name": "example Home Assistant device 1", "port": 12345, "entity_id": "switch.my_fake_switch"

}, {

"name": "example Home Assistant device 2", "port": 12346, "entity_id": "cover.my_fake_cover"

}

]

}

}

```
}
```

**class** fauxmo.plugins.homeassistantplugin.**HomeAssistantPlugin**(*name: str*, *port: int*, *entity_id: str*, *ha_host: str*, *ha_port: int = 8123*, *ha_protocol: str = 'http'*, *ha_token: str = None*)

Bases: *fauxmo.plugins.FauxmoPlugin*

Fauxmo plugin for HomeAssistant REST API.

Allows users to specify Home Assistant services in their config.json and toggle these with the Echo.

**__init__**(*name: str*, *port: int*, *entity_id: str*, *ha_host: str*, *ha_port: int = 8123*, *ha_protocol: str = 'http'*, *ha_token: str = None*) → None

Initialize a HomeAssistantPlugin instance.

**Parameters**

- **ha_token** – Long-lived HomeAssistant token
- **entity_id** – *entity_id* used by HomeAssistant
- **ha_host** – Host running HomeAssistant
- **ha_port** – Port number for HomeAssistant access
- **ha_protocol** – http or https

**get_state**() → str

Query the state of the Home Assistant device.

Returns: Device state as reported by HomeAssistant

**off**() → bool

Turn the Home Assistant device off.

Returns: Whether the device seems to have been turned off.

**on**() → bool

Turn the Home Assistant device on.

Returns: Whether the device seems to have been turned on.

**send**(*signal: str*) → bool

Send *signal* as determined by service_map.

**Parameters signal** – the signal the service should recongize

**service_map = {'cover': {'off': 'close_cover', 'off_state': 'closed', 'on': 'open_**

## fauxmo.plugins.simplehttpplugin module

simplehttpplugin.py :: Fauxmo plugin for simple HTTP requests.

Fauxmo plugin that makes simple HTTP requests in its *on* and *off* methods. Comes pre-installed in Fauxmo as an example for user plugins.

For more complicated requests (e.g. authentication, sending JSON), check out RESTAPIPlugin in *https://github.com/n8henrie/fauxmo-plugins/*, which takes advantage of Requests' rich API.

**class** fauxmo.plugins.simplehttpplugin.**SimpleHTTPPlugin**(**, headers: dict = None, method: str = 'GET', name: str, off_cmd: str, off_data: Union[Mapping[KT, VT_co], str] = None, on_cmd: str, on_data: Union[Mapping[KT, VT_co], str] = None, state_cmd: str = None, state_data: Union[Mapping[KT, VT_co], str] = None, state_method: str = 'GET', state_response_off: str = None, state_response_on: str = None, password: str = None, port: int, use_fake_state: bool = False, user: str = None*)

Bases: *fauxmo.plugins.FauxmoPlugin*

Plugin for interacting with HTTP devices.

The Fauxmo class expects plguins to be instances of objects that inherit from FauxmoPlugin and have on() and off() methods that return True on success and False otherwise. This class takes a mix of url, method, header, body, and auth data and makes REST calls to a device.

This is probably less flexible than using Requests but doesn't add any non-stdlib dependencies. For an example using Requests, see the fauxmo-plugins repo.

The implementation of the *get_state()* method is admittedly sloppy, trying to be somewhat generic to cover a broad range of devices that may have a state that can be queried by either GET or POST request (sometimes differing from the method required to turn on or off), and whose response often contains the state. For example, if state is returned by a GET request to *localhost:8765/state* with *<p>Device is running</p>* or *<p>Device is not running</p>*, you could use those strings as *state_command_on* and *state_command_off*, respectively.

**__init__**(**, headers: dict = None, method: str = 'GET', name: str, off_cmd: str, off_data: Union[Mapping[KT, VT_co], str] = None, on_cmd: str, on_data: Union[Mapping[KT, VT_co], str] = None, state_cmd: str = None, state_data: Union[Mapping[KT, VT_co], str] = None, state_method: str = 'GET', state_response_off: str = None, state_response_on: str = None, password: str = None, port: int, use_fake_state: bool = False, user: str = None*) → None

Initialize a SimpleHTTPPlugin instance.

**Keyword Arguments**

- **headers** – Additional headers for both *on()* and *off()*
- **method** – HTTP method to be used for both *on()* and *off()*
- **name** – Name of the device
- **off_cmd** – URL to be called when turning device off
- **off_data** – Optional POST data to turn device off
- **on_cmd** – URL to be called when turning device on
- **on_data** – Optional POST data to turn device on
- **state_cmd** – URL to be called to determine device state

- **state_data** – Optional POST data to query device state
- **state_method** – HTTP method to be used for *get_state()*
- **state_response_off** – If this string is in the response to state_cmd, the device is off.
- **password** – Password for HTTP authentication (basic or digest only)
- **port** – Port that this device will run on
- **use_fake_state** – If *True*, override *get_state* to return the latest action as the device state. NB: The proper json boolean value for Python's *True* is *true*, not *True* or *"true"*.
- **user** – Username for HTTP authentication (basic or digest only)

**get_state**() → str
Get device state.

**Returns** "on", "off", or "unknown"

**off**() → bool
Turn device off by calling *self.off_cmd* with *self.off_data*.

**Returns** True if the request seems to have been sent successfully

**on**() → bool
Turn device on by calling *self.on_cmd* with *self.on_data*.

**Returns** True if the request seems to have been sent successfully

**set_state**(*cmd: str*, *data: bytes*) → bool
Call HTTP method, for use by *functools.partialmethod*.

**Parameters**

- **cmd** – Either *"on_cmd"* or *"off_cmd"*, for *getattr(self, cmd)*
- **data** – Either *"on_data"* or *"off_data"*, for *getattr(self, data)*

**Returns** Boolean indicating whether it state was set successfully

## Module contents

fauxmo.plugins :: Provide ABC for Fauxmo plugins.

**class** fauxmo.plugins.**FauxmoPlugin**(**, *name: str*, *port: int*)
Bases: `abc.ABC`

Provide ABC for Fauxmo plugins.

This will become the *plugin* attribute of a *Fauxmo* instance. Its *on* and *off* methods will be called when Alexa turns something *on* or *off*.

All keys (other than the list of *DEVICES*) from the config will be passed into FauxmoPlugin as kwargs at initialization, which should let users do some interesting things. However, that means users employing custom config keys will need to override *__init__* and either set the *name* and "private" *_port* attributes manually or pass the appropriate args to *super().__init__()*.

**__init__**(*, *name: str*, *port: int*) → None
Initialize FauxmoPlugin.

**Keyword Arguments**

- **name** – Required, device name

- **port** – Required, port that the Fauxmo associated with this plugin should run on

Note about *port*: if not given in config, it will be set to an apparently free port in *fauxmo.fauxmo* before FauxmoPlugin initialization. This attribute serves no default purpose in the FauxmoPlugin but is passed in to be accessible by user code (i.e. for logging / debugging). Alternatively, one could accept and throw away the passed in *port* value and generate their own port in a plugin, since the Fauxmo device determines its port from the plugin's instance attribute.

The *_latest_action* attribute stores the most recent successful action, which is set by the *__getattribute__* hackery for successful *.on()* and *.off()* commands.

**close**() → None
Run when shutting down; allows plugin to clean up state.

**get_state**() → str
Run function when Alexa requests device state.

Should return "on" or "off" if it can be determined, or "unknown" if there is no mechanism for determining the device state, in which case Alexa will complain that the device is not responding.

If state cannot be determined, a plugin can opt into this implementation, which falls back on the *_latest_action* attribute. It is intentionally left as an abstract method so that plugins cannot omit a *get_state* method completely, which could lead to unexpected behavior; instead, they should explicitly *return super().get_state()*.

**latest_action**
Return latest action in read-only manner.

Must be a function instead of e.g. property because it overrides *get_state*, and therefore must be callable.

**name**
Return name attribute in read-only manner.

**off**() → bool
Run function when Alexa turns this Fauxmo device off.

**on**() → bool
Run function when Alexa turns this Fauxmo device on.

**port**
Return port attribute in read-only manner.

### 2.1.2 Submodules

### 2.1.3 fauxmo.cli module

cli.py :: Argparse based CLI for fauxmo.

Provides console_script via argparse.

fauxmo.cli.**cli**() → None
Parse command line options, provide entry point for console scripts.

### 2.1.4 fauxmo.fauxmo module

fauxmo.py :: Main server code for Fauxmo.

Emulates a Belkin Wemo for interaction with an Amazon Echo. See README.md at <https://github.com/n8henrie/fauxmo>.

`fauxmo.fauxmo.`**`main`**(*config_path_str: str = None*, *verbosity: int = 20*) → None
Run the main fauxmo process.

Spawns a UDP server to handle the Echo's UPnP / SSDP device discovery process as well as multiple TCP servers to respond to the Echo's device setup requests and handle its process for turning devices on and off.

**Parameters**

- **`config_path_str`** – Path to config file. If not given will search for *config.json* in cwd, *~/.fauxmo/*, and */etc/fauxmo/*.
- **`verbosity`** – Logging verbosity, defaults to 20

### 2.1.5 fauxmo.protocols module

protocols.py :: Provide asyncio protocols for UPnP and SSDP discovery.

**`class`** `fauxmo.protocols.`**`Fauxmo`**(*name: str*, *plugin: fauxmo.plugins.FauxmoPlugin*)
Bases: `asyncio.protocols.Protocol`

Mimics a WeMo switch on the network.

Aysncio protocol intended for use with BaseEventLoop.create_server.

**`NEWLINE = '\r\n'`**

**`__init__`**(*name: str*, *plugin: fauxmo.plugins.FauxmoPlugin*) → None
Initialize a Fauxmo device.

**Parameters**

- **`name`** – How you want to call the device, e.g. "bedroom light"
- **`plugin`** – Fauxmo plugin

**`static add_http_headers`**(*xml: str*) → str
Add HTTP headers to an XML body.

**Parameters** **`xml`** – XML body that needs HTTP headers

**`connection_made`**(*transport: asyncio.transports.BaseTransport*) → None
Accept an incoming TCP connection.

**Parameters** **`transport`** – Passed in asyncio.Transport

**`data_received`**(*data: bytes*) → None
Decode incoming data.

**Parameters** **`data`** – Incoming message, either setup request or action request

**`handle_action`**(*msg: str*) → None
Execute *on*, *off*, or *get_state* method of plugin.

**Parameters** **`msg`** – Body of the Echo's HTTP request to trigger an action

**`handle_event`**() → None
Respond to request for eventservice.xml.

**`handle_metainfo`**() → None
Respond to request for metadata.

**`handle_setup`**() → None
Create a response to the Echo's setup request.

**class** `fauxmo.protocols.`**`SSDPServer`**(*devices: Iterable[dict] = None*)

Bases: `asyncio.protocols.DatagramProtocol`

UDP server that responds to the Echo's SSDP / UPnP requests.

**`__init__`**(*devices: Iterable[dict] = None*) → None

Initialize an SSDPServer instance.

**Parameters** **`devices`** – Iterable of devices to advertise when the Echo's SSDP search request is received.

**`add_device`**(*name: str*, *ip_address: str*, *port: int*) → None

Keep track of a list of devices for logging and shutdown.

**Parameters**

- **`name`** – Device name
- **`ip_address`** – IP address of device
- **`port`** – Port of device

**`connection_lost`**(*exc: Exception*) → None

Handle lost connections.

**Parameters** **`exc`** – Exception type

**`connection_made`**(*transport: asyncio.transports.BaseTransport*) → None

Set transport attribute to incoming transport.

**Parameters** **`transport`** – Incoming asyncio.DatagramTransport

**`datagram_received`**(*data: Union[bytes, str], addr: Tuple[str, int]*) → None

Check incoming UDP data for requests for Wemo devices.

**Parameters**

- **`data`** – Incoming data content
- **`addr`** – Address sending data

**`respond_to_search`**(*addr: Tuple[str, int], discover_pattern: str, mx: float = 0.0*) → None

Build and send an appropriate response to an SSDP search request.

**Parameters** **`addr`** – Address sending search request

### 2.1.6 fauxmo.utils module

utils.py :: Holds utility functions for Fauxmo.

`fauxmo.utils.`**`get_local_ip`**(*ip_address: str = None*) → str

Attempt to get the local network-connected IP address.

**Parameters** **`ip_address`** – Either desired ip address or string or "auto"

**Returns** Current IP address as string

`fauxmo.utils.`**`get_unused_port`**() → int

Temporarily binds a socket to an unused system assigned port.

**Returns** Port number

`fauxmo.utils.`**`make_serial`**(*name: str*) → str

Create a persistent UUID from the device name.

Returns a suitable UUID derived from *name*. Should remain static for a given name.

**Parameters** **`name`** – Friendly device name (e.g. "living room light")

**Returns** Persistent UUID as string

`fauxmo.utils.`**`make_udp_sock`**`()` → socket.socket

Make a suitable udp socket to listen for device discovery requests.

I would *love* to get rid of this function and just use the built-in options to *create_datagram_endpoint* (e.g. *allow_broadcast* with appropriate local and remote addresses), but having no luck. Would be thrilled if someone can figure this out in a better way than this or <https://github.com/n8henrie/fauxmo/blob/c5419b3f61311e5386387e136d26dd8d4a55518c/src/fauxmo/protocols.py#L149>.

**Returns** Socket suitable for responding to multicast requests

`fauxmo.utils.`**`module_from_file`**`(`*modname: str*, *path_str: str*`)` → module

Load a module into *modname* from a file path.

**Parameters**

- **`modname`** – The desired module name
- **`path_str`** – Path to the file

**Returns** Module read in from path_str

### 2.1.7 Module contents

fauxmo :: Emulated Belkin Wemo devices for use with the Amazon Echo.

# CHAPTER 3

# Credits

## 3.1 Development Lead

- Nathan Henrie nate@n8henrie.com

## 3.2 Contributors

- Originally based on work by Maker Musings

# CHAPTER 4

# Changelog

Will not contain minor changes – feel free to look through `git log` for more detail.

## 4.1 v0.5.0 :: 20191212

- Add py38 support
- Add `use_fake_state` option to accommodate situations that state can't be properly determined (thanks @johngo7470)
- Bugfix: fix unexpected behavior with a switch's state logic was true for both `on` and `off`
- Migrated HomeAssistantPlugin and CommandLinePlugin from fauxmo-plugins repo
- Update tests, pytest fixtures, and add some mocks

## 4.2 v0.4.9 :: 20190527

- Add py37 support (including Travis workaround)
- Fix bug in content-length calculation (thanks @tim15)
- Replace `find_unused_port` with local function (thanks @schneideradam)
- Use black for formatting
- Update `config-sample.txt` for changes in HomeAssistant API

## 4.3 v0.4.8 :: 20180804

- Add `.close()` method to `FauxmoPlugins`, allowing for cleanup (thanks @howdypierce) discussion, e907245

- Append plugins directory to `sys.path` for more convenient loading of additional modules (thanks @howdypierce) discussion, 03f2101
- Add HTTP headers to `/eventservice.xml` and `/metainfoservice.xml` endpoints 5a53268

## 4.4 v0.4.7 :: 20180512

- Minor dev-side changes
  - Use pipenv for dev dependency management
- Add utf-8 to readme parsing (5 days ago) (thanks @hestela!) 49d2c57
- Change newline to `\r\n` in HTTP responses (thanks @GlennPegden2) 239bc79
- Match `MAN:` case insensitive (thanks @wingett) 8307096
- Add GetBinaryState and GetFriendlyName commands including test cases (thanks @howdypierce!) 71392de
- Make comparison of the "SOAPACTION" header case-insensitive, per UPnP spec (thanks @howdypierce!) a5cdf82
- Add fallback for determining IP address when DNS resolution is a problem (thanks @howdypierce!) c2d7f13
- Bugfix: ~/.fauxmo/ not being read as a location for config file (thanks @howdypierce!) c322c9b

## 4.5 v0.4.6 :: 20180212

- Mostly changes to try to fix compatibility with newer generation Echos / Echo Plus, see #38

## 4.6 v0.4.5 :: 20171114

- Support new GetBinaryState command (fixes n8henrie/fauxmo#31)

## 4.7 v0.4.3 :: 20170914

- Add `--version` to cli
- Add `python_requires` specifier to `setup.py`
- Bind to specific address in `make_udp_sock` (`fauxmo.utils`), seems to fix some intermittent failing tests on MacOS.

## 4.8 v0.4.2 :: 20170601

- Add additional linters to tests
- Set reuseaddr and reuseport before binding socket

## 4.9 v0.4.0 :: 20170402

- Rename handlers to plugins
- Add interface for user plugins
- Add type hints
- Require Python 3.6
- Eliminate third party dependencies
- Make sure to close connection when plugin commands fail / return False

## 4.10 v0.3.3 :: 20160722

- Added compatibility for `rollershutter` to `handlers.hass`
- Changed `handlers.hass` to send values from a dict to make addition of new services easier in the future

## 4.11 v0.3.2 :: 20160419

- Update SSDPServer to `setsockopt` to permit receiving multicast broadcasts
- `sock` kwarg to `create_datagram_endpoint` no longer necessary, restoring functionality to Python 3.4.0 - 3.4.3 (closes #6)
- `make_udp_sock()` no longer necessary, removed from `fauxmo.utils`
- Tox and Travis configs switched to use Python 3.4.2 instead of 3.4.4 (since 3.4.2 is the latest available in the default Raspbian Jessie repos)

## 4.12 v0.3.1 :: 20160415

- Don't decode the UDP multicast broadcasts (hopefully fixes #7)
  - They might not be from the Echo and might cause a `UnicodeDecodeError`
  - Just search the bytes instead
- Tests updated for this minor change

## 4.13 v0.3.0 :: 20160409

- Fauxmo now uses asyncio and requires Python >= 3.4.4
- *Extensive* changes to codebase
- Handler classes renamed for PEP8 (capitalization)
- Moved some general purpose functions to `fauxmo.utils` module
- Both the UDP and TCP servers are now in `fauxmo.protocols`
- Added some rudimentary pytest tests including tox and Travis support

- Updated documentation on several classes

## 4.14 v0.2.0 :: 20160324

- Add additional HTTP verbs and options to `RestApiHandler` and Indigo sample to config
  - **NB:** Breaking change: `json` config variable now needs to be either `on_json` or `off_json`
- Make `RestApiHandler` DRYer with `functools.partialmethod`
- Add `SO_REUSEPORT` to `upnp.py` to make life easier on OS X

## 4.15 v0.1.11 :: 20160129

- Consolidate logger to `__init__.py` and import from there in other modules

## 4.16 v0.1.8 :: 20160129

- Add the ability to manually specify the host IP address for cases when the auto detection isn't working (https://github.com/n8henrie/fauxmo/issues/1)
- Deprecated the `DEBUG` setting in `config.json`. Just use `-vvv` from now on.

## 4.17 v0.1.6 :: 20160105

- Fix for Linux not returning local IP
  - restored method I had removed from Maker Musings original / pre-fork version not knowing it would introduce a bug where Linux returned 127.0.1.1 as local IP address

## 4.18 v0.1.4 :: 20150104

- Fix default verbosity bug introduced in 1.1.3

## 4.19 v0.1.0 :: 20151231

- Continue to convert to python3 code
- Pulled in a few PRs by @DoWhileGeek working towards python3 compatibility and improved devices naming with dictionary
- Renamed a fair number of classes
- Added kwargs to several class and function calls for clarity
- Renamed several variables for clarity
- Got rid of a few empty methods

- Import devices from `config.json` and include a sample
- Support `POST`, headers, and json data in the RestApiHandler
- Change old debug function to use logging module
- Got rid of some unused dependencies
- Moved license (MIT) info to LICENSE
- Added argparse for future console scripts entry point
- Added Home Assistant API handler class
- Use "string".format() instead of percent
- Lots of other minor refactoring

# CHAPTER 5

# 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:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/n8henrie/fauxmo/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.

### 5.1.2 Fix Bugs

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

### 5.1.3 Implement Features

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

### 5.1.4 Write Documentation

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

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/n8henrie/fauxmo/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 :)

### 5.1.6 Create a new Plugin

Please refer to https://github.com/n8henrie/fauxmo-plugins

## 5.2 Get Started!

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

1. Start by making an issue to serve as a reference point for discussion regarding the change being proposed.
2. Fork the fauxmo repo on GitHub.
3. Clone your fork locally:

~~~
```shell_session
$ git clone git@github.com:your_name_here/fauxmo.git
```
~~~

4. Install your local copy into a virtualenv. Assuming you have python >= 3.6 installed, this is how you set up your fork for local development:

~~~
```shell_session
$ cd fauxmo
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install -e .[dev]
```
~~~

5. Create a branch for local development:

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

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass all tests configured for each Python version with tox:

~~~
```shell_session
$ tox
```
~~~

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

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

8. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

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

1. Pull requests of any substance should reference an issue used for discussion regarding the change being considered.
2. The style should pass `tox -e lint`, including docstrings, type hints, and `black --line-length=79 --target-version=py37` for overall formatting.
3. The pull request should include tests if I am using tests in the repo.
4. 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.md
5. The pull request should work for Python 3.7. If I have included a `.travis.yml` file in the repo, check https://travis-ci.org/n8henrie/fauxmo/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests: `pytest tests/test_your_test.py`

CHAPTER 6

# protocol_notes.md

Details on the Echo's interaction with Fauxmo, and how to examine it for debugging.

Tons of information gathered by @makermusings, I *strongly* recommend you start by reading these:

- https://github.com/makermusings/fauxmo/blob/master/protocol_notes.txt
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation

In summary:

1. User tells Echo to "find connected devices" or clicks corresponding button in webapp
2. Echo broadcasts "device search" to `239.255.255.250:1900` (UDP)
3. Fauxmo response includes `LOCATION` of `setup.xml` endpoint for each "device" in config (UDP)
4. Echo requests `setup.xml` endpoint at above `LOCATION` (HTTP) for each device
5. Fauxmo responds with setup information for each device (HTTP)
6. Alexa verbally announces any discovered devices (*really* wish I could mute this – set volume to 1 beforehand if I'll be doing it a bunch), and they also show up in the webapp

Once you understand the basic model of interaction, the next step in debugging is to inspect the actual requests and responses.

The following commands require some tools you might not have by default; you can get them with: `sudo apt-get install tcpdump tshark nmap`. Doesn't matter what you choose regarding the wireshark question you'll get during installation; just read the warning and make a good decision. On OSX, use homebrew to install the same.

First, get the IP address of your Echo. If you don't know it:

```
# Assuming your local subnet is 192.168.27.*
sudo nmap -sP 192.168.27.1/24 | grep -i -B 2 amazon
```

You should get `Nmap scan report for 192.168.27.XXX` – your Echo IP address. For the examples below, I'll use `192.168.27.100` as the Echo IP address, and `192.168.27.31` as the Pi's IP address (31 as in 3.14, easier to remember).

Next, we'll check out the info being sent to and from the Echo and Fauxmo. In one window, run Fauxmo in verbose mode. In a second window, run the commands below, and check their output when you tell the Echo to find connected devices.

To get an overview of what's going on, start with `tshark`:

```
sudo tshark -f "host 192.168.27.100" -Y "udp"

# Alternatively, only show the Echo's SEARCH requests:
sudo tshark -f "host 192.168.27.100" -Y "udp contains SEARCH"

# Only show the Fauxmo responses (note still using the Echo's IP):
sudo tshark -f "host 192.168.27.100" -Y "udp contains LOCATION"
```

Example output for the first command, showing a few sets of SSDP SEARCH sent by the Echo followed by 4 responses by Fauxmo (1 for each device in sample config).

```
Capturing on 'eth0'
  1   0.000000 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  2   0.046414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  3   0.064351 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  4   0.082011 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  5   0.101093 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  6   0.104016 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  7   0.151414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  8   0.171049 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  9   0.191602 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 10   0.199882 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 11   0.231841 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
 12   0.333406 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
```

To get a raw look at all the info, use `tcpdump`. I've cleaned up a bunch of garbage in the below output, but you should still be able to recognize each of the critical components.

```
sudo tcpdump -s 0 -i eth0 -A host 192.168.27.100
```

This should show a ton of detailed info, including all responses sent to / from the Echo. Replace `etho` with your network interface (check with `ip link`) and `192.168.27.100` with your Echo's IP address.

The output should start with several of the Echo's UDP based discovery requests, where you can recognize the `UDP` protocol being sent from the Echo `192.168.27.100` to the network's multicast broadcast `239.255.255.250.1900`, something like:

```
15:48:39.268125 IP 192.168.27.100.50000 > 239.255.255.250.1900: UDP, length 122
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: 15
ST: urn:Belkin:device:**
```

Below that, you should see Fauxmo's responses, also UDP, one for each device in the config. This response provides the Echo with the `LOCATION` of the device's `setup.xml`.

```
15:48:39.513741 IP 192.168.27.31.1900 > 192.168.27.100.50000: UDP, length 386
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=86400
DATE: Sun, 24 Apr 2016 21:48:39 GMT
EXT:
```

(continues on next page)

(continued from previous page)

```
LOCATION: http://192.168.27.31:12340/setup.xml
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: c66d1ad0-707e-495e-a21a-1d640eed4547
SERVER: Unspecified, UPnP/1.0, Unspecified
ST: urn:Belkin:device:**
USN: uuid:Socket-1_0-2d4ac336-8683-3660-992a-d056b5382a8d::urn:Belkin:device:**
```

Somewhere below that, you'll see the Echo request each device's `setup.xml` (based on the `LOCATION` from the prior step), this time TCP instead of UDP.

```
15:48:39.761878 IP 192.168.27.100.39720 > 192.168.27.31.12341: Flags [P.], seq 1:68,␣
↪ack 1, win 274, options [nop,nop,TS val 619246756 ecr 140303456], length 67
GET /setup.xml HTTP/1.1
Host: 192.168.27.31:12341
Accept: */*
```

And somewhere below that, Fauxmo's setup response, for each device in the config, also TCP:

```
15:48:39.808164 IP 192.168.27.31.12342 > 192.168.27.100.59999: Flags [P.], seq 1:608,␣
↪ack 68, win 453, options [nop,nop,TS val 140303462 ecr 619246754], length 607
HTTP/1.1 200 OK
CONTENT-LENGTH: 375
CONTENT-TYPE: text/xml
DATE: Sun, 24 Apr 2016 21:48:39 GMT
LAST-MODIFIED: Sat, 01 Jan 2000 00:01:15 GMT
SERVER: Unspecified, UPnP/1.0, Unspecified
X-User-Agent: Fauxmo
CONNECTION: close

<?xml version="1.0"?>
<root>
<device>
<deviceType>urn:Fauxmo:device:controllee:1</deviceType>
<friendlyName>fake hass switch by REST API</friendlyName>
<manufacturer>Belkin International Inc.</manufacturer>
<modelName>Emulated Socket</modelName>
<modelNumber>3.1415</modelNumber>
<UDN>uuid:Socket-1_0-cbc4bc63-e0e2-3a78-8a9f-f0ff7e419b79</UDN>
</device>
</root>
```

Then, to get a *really* close look at a request, well go back to `tshark`. For example, we can add the `-V` flag to get a **ton** more info, and add `-c 1` (count) to limit to capturing a single packet, and further refine the capture filter by specifying that we only want to look at packets sent **from** the Pi **to** the Echo.

```
sudo tshark -f "src 192.168.27.31 and dst 192.168.27.100" -c 1 -V
```

At the bottom, you should find the `Hypertext Transfer Protocol` section contains the same `setup.xml` response we found in the `tcpdump` output above.

You can also send requests from another device on the network to check out Fauxmo's responses and ensure that they're getting through the network. For example, to simulate the Echo's device search, run the following from another device on the network, in two different windows:

```
# Seems to work with `nc.traditional` on Raspberry Pi, not yet working for me on OSX
# Window 1: Listen for response on port 12345 (should show up once second command is␣
↪sent)
```

(continues on next page)

(continued from previous page)

```
nc.traditional -l -u -p 12345

# Window 2: Send simulated UDP broadcast device search (from port 12345)
echo -e '"ssdp:discover"urn:Belkin:device:**' | nc.traditional -b -u -p 12345 239.255.
↪255.250 1900
```

To request a device's `setup.xml`, using the device's `port` from `config.json`:

```
# Send a request for the `setup.xml` of a device from the sample config
curl -v 192.168.27.31:12340/setup.xml
```

The above commands may seem a little complicated if you're unfamiliar, but they're immensely powerful and indispensable for debugging these tricky network issues. If you're not already familiar with them, learning the basics will serve you well in your IoT endeavors!

To verify that Fauxmo is working properly, check for a few things:

1. Is the Pi consistently seeing the Echo's `M-SEARCH` requests?
2. Is Fauxmo consistently replying with the `LOCATION` responses?
3. Is the Echo then requesting the `setup.xml` (for each device)?
4. Is Fauxmo consistently replying with the setup info?

If you can confirm that things seem to be working through number 4, then it would seem that Fauxmo is working properly, and the issue would seem to be elsewhere.

## 6.1 On and Off Commands

One way to examine exactly what the Echo sends to one of your connected Fauxmo devices (i.e. one that *already* works as expected) is to first **stop** Fauxmo (to free up the port), then use netcat to listen to that port while you trigger the command. E.g. for a Fauxmo device configured to use port `12345`, run `nc.traditional -l 12345` and then tell the Echo to "turn on [device name]". The Echo will notify you that the command failed, obviously, because Fauxmo isn't running, but you should be able to see exactly what the Echo sent.

These are the requests that the Echo sends to Fauxmo when you ask it to turn a device…

### 6.1.1 On

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState>
</u:SetBinaryState>
```

(continues on next page)

(continued from previous page)

```
</s:Body>
</s:Envelope>
```

### 6.1.2 Off

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>0</BinaryState>
</u:SetBinaryState>
</s:Body>
</s:Envelope>
```

Several similar terms can be used instead of `On` and `Off`, e.g. `Open` and `Close`; the response looks identical. This Reddit post has a good number more that work. NB: the `Dim` commands in the post don't seem to work (likely incompatible with Wemo devices, so the Echo doesn't even try to send them).

As of sometime around 20171030, it will also now send a `GetBinaryState` action when viewing a device, which can be problematic for earlier versions of Fauxmo (prior to v0.4.5).

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#GetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:GetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState>
</u:GetBinaryState>
</s:Body>
</s:Envelope>
```

I think I have a good idea what the Fauxmo response should look like, thanks to help from:

- u/romanpet
- https://github.com/go-home-iot/belkin/blob/7b62ec854e9510f4857bb9eceeb8fef3d8b55fb4/device.go

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
```

(continues on next page)

(continued from previous page)

```
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#GetBinaryState"
Content-Length: 299

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:GetBinaryStateResponse xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>0</BinaryState>
</u:GetBinaryStateResponse>
</s:Body>
</s:Envelope>
```

# CHAPTER 7

# Indices and tables

- genindex
- modindex

# Python Module Index

# Index