hueman Documentation

Release 0.3.0

Will Boyce

January 02, 2017

Hueman is a application and library for managing Philips Hue Connected bulbs, written in Python, designed for Humans.

CHAPTER 1

User Guide

This part of the documentation, which is mostly prose, takes you step-by-step through initial configuration and getting to know Hueman.

1.1 Installation

This part of the documentation covers the installation of Hueman.

1.1.1 Distribute & Pip (recommended)

Installing hueman is simple with pip:

```
% pip install hueman
```

Or you can install directly from GitHub:

```
% pip install git+https://github.com/wrboyce/hueman.git#egg=hueman
```

1.1.2 Get the Code

Hueman is actively developed and maintained on GitHub, you can either:

Clone the public repository:

```
% git clone https://github.com/wrboyce/hueman.git
```

or download the tarball:

```
% curl -OL https://github.com/wrboyce/hueman/tarball/master
```

1.2 Initial Configuration

1.2.1 Adding Your Bridges

The bare minimum configuration for Hueman is letting it know where your bridges are and providing a username for each one. Knowing these details is currently an exercise for the reader.

By default (and who doesn't like defaults?) your configuration file is stored at `~/.hueman.yml`, add your bridges like so:

```
bridges:
  - hostname: limelight01.example.com
    username: your-app-hash-goes-here
  - hostname: limelight02.example.com
    username: your-app-hash-goes-here
```

That's it, you're ready to go!

1.3 Quick Start

Ok, time to get your hands dirty. This section assumes that you already have Hueman installed and configured. If you do not, head over to the Installation or Configuration section.

1.3.1 Finding the Light

Now hueman can talk to your bridges, listing the available lights is simple:

```
% hueman -Ll
bedroom.hers
bedroom.his
office.desklamp
office.floorlamp
```

Now we know what we're dealing with, there are a few ways to direct commands to lights, the most direct of which of which is `-l`/`--light`:

```
% hueman -l bedroom.hers,bedroom.his on
% hueman -l bedroom.hers -l bedroom.his off
```

Next up is the `-f`/`--find`, which allows you to match lights with more flexibility, such as:

- Full name:

```
    % hueman -f office.desklamp,office.floorlamp on
```

- Simple Wildcards:

```
    % hueman -f office.* on
```

- Regular Expressions:

```
    % hueman -f /office/ on
    % hueman -f /^office\.(desk|floor)lamp$/ on
```

And finally, sometimes all you need is a sledgehammer:

```
% hueman -a on
```

1.3.2 The Colour of Magic

The Hue bulbs are mighty versatile things, and Hueman aims to match that. You can control any aspect of the bulbs via Hueman with a very simply syntax: `key:val`.

The bulbs currently expose the following attributes:

- bri
- hue
- sat
- xy
- ct
- transitiontime
- effect
- colormode

In addition, Hueman provides these aliases:

- brightness (bri)
- saturation (sat)
- time (transitiontime)
- mode (colormode)

```
% hueman -a brightness:100 sat:100 time:10
```

Refer to the Hue API docs for more information on these attributes.

1.4 Groups

1.4.1 Brighter Together

Due to limitations in the Philips Hue API, we have to define groups client-side. Groups are defined in your config file:

```
groups:
  bedroom:
    - bedroom.hers
    - bedroom.his
  office:
    - office.desklamp
    - office.floorlamp
```

1.4.2 Introducing –group

Now you have groups configured, you can reference them with the `-g/--group` argument:

```
% hueman -g office on
% hueman -g bedroom,office off
```

1.4.3 Another Trick up –find's Sleeve

Find can also reference groups:

```
% hueman -f office on
% hueman -f bedroom,office on
```

Note that when using find, groups will take precedence over lights.

1.5 Presets

Presets allow you to save a lamp state and recall it by name. Hueman comes with some presets, but you can define your own in your config file.

1.5.1 Defining Presets

Presets can change any attribute normally available:

```
presets:
  bright:
    bri: 255
  red:
    sat: 255
```

1.5.2 Invoking Presets

Invoking presets is easy, just say their name:

```
% hueman -a bright
% hueman -g bedroom red
```

Or even combine them:

```
% hueman -l office bright red
```

1.5.3 Transitions

Presets can also hold two states and have the lights transition into the second state over a predefined time:

```
presets:
  slowly_bright:
    - bri: 100
    - bri: 255
      time: 100
```

```
% hueman -a slowly bright
```

1.6 Scenes

Scenes are a culmination of Groups and Presets. They allow you apply a set of actions to predefined a light. As usual, scenes are defined in your config file.

1.6.1 Setting the Scene

Scenes are defined like so, mapping lights/groups to states:

```
scenes:
  work_mode:
    office:
      bri: 240
      ct: 100
    bedroom:
      on: No
```

Each state is applied in the same manner as a preset, and targets are resolved using the same rules as `find`.

1.6.2 Reliving the Moment

Brining a scene back is the same as invoking a prefix, only you do not need to (and it would not make sense to) define a target:

```
% hueman work mode
```

Advanced Usage

This section covers advanced topics to help you make the most out of Hueman.

2.1 Attributes

We touched on modifying light attributes in the getting started section; in this section we will look at some more advanced means Hueman can modify the state of lights offering a greater degree of flexibility.

2.1.1 Relative Values and Percentages

The following attributes (and their aliases) can be modified using percentages and relative values:

- bri
- hue
- sat
- ct

Supported operations are:

- Absolute percentage:

```
% hueman -a bri:100%  # set brightness to 255
```

- Relative percentage:

```
% hueman -a bri:~50%  # set brightness to 50% of current value, 127
```

- Addition and Subtraction:

```
% hueman -a bri:+23  # increase brightness by 23 -> 150
% hueman -a bri:-50  # decreate brightness by 50 -> 100
```

- You can also do relative percentage addition and substraction:

```
% hueman -a bri:-50%  # decreate brightness by 50% -> 50
% hueman -a bri:+200%  # increase brightness by 200% -> 150
```

These values are also clippped within their known ranges, to avoid any weird behaviour:

```
% hueman -a bri:500  # brightness is set to known max, 255
```

2.1.2 Fancy Durations

The default unit for `transitiontime` is tenths of a second, not very Hueman at all. Hueman understands seconds *and* minutes:

```
% hueman -a bright transitiontime:900  # 900 tenths of a second (or 90 seconds, or one and a half mi
% hueman -a bright time:1m30s  # as above, but much more readable
```

Note that `time` is simply an alias of `transitiontime`, both keys will accept either format of duration.

2.2 Plugins

Plugins are where the real fun begins. Hueman comes with a few plugins, but you can also add your own (more on that later). Plugins can be invoked one of two ways, either just the name like a plugin, or name:args - if the plugin requires arguments:

```
% hueman -a plugin
% hueman -a plugin:args
```

2.2.1 Colour

The colour plugin allows you to set your bulbs to a “natural” colour:

```
% hueman -a colour:red
```

2.2.2 RGB

If you can’t quite find the colour you’re looking for, you can instead provide an RGB hex:

```
% hueman -a rgb:ff0000
```

2.2.3 Always Take the Weather With You

The weather plugin requires some configuration before you can use it. Plugin settings are defined in your config file:

```
plugins:
  weather:
    settings:
      latitude: 0.0
      longitude: 0.0
```

:: % hueman -a weather

2.2.4 Mixing it Up

And, of course, you can mix plugins and attributes (and scenes, and presets...):

```
% hueman -a colour:red bri:75%
% hueman -a slowly bright colour:red
% hueman work mode colour:red
```

2.2.5 Rolling Your Own

Plugins are a simple class which implement the `__call__` function, with settings being passed to the `__init__` function:

```python
class MyPlugin1(object):
    def __call__(self, controller, value):
        controller.brightness(value)


class MyPlugin2(object):
    def __init__(self, default_brightness):
        self.default_brightness = default_brightness

    def __call__(self, controller):
        self.controller.brightness(default_brightness)
```

The `__call__` method should accept at a minimum one argument (in addition to `self`) which is the target of the action. Plugins must be registered in your config file:

```
presets:
  myplug1: module.path.MyPlugin1
  myplug2:
    path: module.path.MyPlugin2
    settings:
      default_brightness: 255
```

`path` should be a standard Python import path, and settings is a dictionary passed to `__init__` as kwargs. Once registered plugins can be invoked by name:

```
% hueman -g bedroom myplug1:100
% hueman -g office myplug2
```

Refer to the API Documentation for more information on Plugin authoring.

2.3 Inventory

With so much going on, its useful to be able to see what is available - and what it does.

2.3.1 Plugins

Gathering information about registered plugins is still undecided, but it'll look something like this:

```
% hueman -LP
colour
rgb
weather

% hueman -LPv
colour:colour_name
```

```
    Lookup and set a colour by name
rgb:rgb_hex
    Set a colour by hex value
weather
    Pick a colour based on daily weather forecast

% hueman -h weather
Pick a colour based on the daily weather forecast
Required settings: latitude, longitude
```

2.3.2 Presets

You can list all available presets with `hueman -Lp`:

```
% hueman -Lp
bright
red
```

And view their states by adding the `verbose` flag:

```
% hueman -Lpv
bright  {bri: 100%}
red     {sat: 255}
```

2.3.3 Scenes

Likewise with scenes:

```
% hueman -Ls
work mode
% hueman -Lsv
work mode:
    office:     {bri: 240, ct: 100}
    bedroom:    {on: No}
```

CHAPTER 3

API Documentation

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

CHAPTER 4

Class Index

4.1 hueman Package

4.1.1 `hueman` Package

4.1.2 `entities` Module

class `hueman.entities.`**`Bridge`**(*hostname*, *username*, *groups={}*, *plugins={}*, *presets={}*, *scenes={}*)

Bases: *`hueman.entities.Group`*

`find`(**names*)

`group`(*name*)

Lookup a group by name, if name is None return all groups.

class `hueman.entities.`**`Controller`**(*bridge*, *id*, *name*, *cstate=None*, *nstate=None*)

Bases: `object`

`commit`()

Send any outstanding changes to the Endpoint.

`preset`(*name*, *commit=False*)

Load a preset state

`reset`()

Drop any uncommitted changes.

`state`

Return the current state

class `hueman.entities.`**`Group`**(*bridge*, *id*, *name*, *cstate=None*, *nstate=None*)

Bases: *`hueman.entities.Controller`*

Mostly useless currently, until we can create new Groups using the Hue API.

`light`(*name*)

Lookup a light by name, if name is None return all lights.

`lights`(**names*)

class `hueman.entities.`**`Light`**(*bridge*, *id*, *name*, *cstate=None*, *nstate=None*)

Bases: *`hueman.entities.Controller`*

A light, a bulb... The fundamental endpoint.

4.1.3 `groups` Module

class `hueman.groups.`**`GroupController`**(*name=''*)

Bases: `object`

Dispatches calls to its member Controllers (recursively!). Members can be Lights, Groups, Bridges or GroupControllers.

`add_member`(*obj*)
Add a single Light/Group/Bridge or GroupController to the current GroupController.

`add_members`(*iter*)
Shortcut to *add_member* when you want to add many, will consume any iterable.

`find`(**names*)
Find members by name

`group`(**names*)
Find members by name

`light`(**names*)
Find members by name

`members`
Return a `list` of Group members.

class `hueman.groups.`**`Hueman`**(*cfg*)

Bases: *`hueman.groups.GroupController`*

Top level *GroupController* for managing all your Bridges and Configurations

`scene`(*scene*, *commit=False*)

4.1.4 `util` Module

`hueman.util.`**`cli`**(*args=None*)
Commandline Entrypoint

`hueman.util.`**`loader`**(*cfg_file='~/.hueman.yml'*)
Shortcut function to furnish you with a configured *Hueman*.

Python Module Index