PHILIPS

hue PERSONAL WIRELESS LIGHTING

How to **develop for Hue?**

Core Concepts

The hue system is built around the idea of everything in your system having a unique URL served by the bridge. Interacting with these URLs lets you modify them or find out their current state as explained above.

Hue Web Addresses

A hue resource web address will always start with the following.

```
http://<bridge IP address>/api
```

This is the RESTful root and is how your app or controller talks to the hue bridge interface.

Hue Username

In most of the commands (the exceptions are creating a username and getting basic bridge information – Login Required) you'll also include a username after this:

```
http://<bridge IP address>/api/<username>
```

This username determines which resources you have access to. If you provide a username that isn't known to the bridge then most resources won't be available to you. Using one that is authorized, as shown in the getting started section, will allow you to interact with pretty much everything interesting.

Each new instance of an app should use a unique username which you generate using the Create New User command.

Lights/Groups and More

There are different kinds of resources to interact with and we've clustered them together with similar objects. The seven available currently are:

`/lights` resource which contains all the light resources
`/groups` resource which contains all the groups
`/config` resource which contains all the configuration items
`/schedules` which contains all the schedules
`/scenes` which contains all the scenes
`/sensors` which contains all the sensors
`/rules` which contains all the rules

We'll be adding more as we add features to the system

You can query resources available in your bridge by doing a GET on its local URL. For example the following returns all configuration items in your bridge.

Address	`http://<bridge IP address>/api/<username>/config`
Method	`GET`

Change a Setting or Resource

The principle for changing a setting, attribute or other 'resource' on the device is to send a `PUT` request to the URL of its parent. The desired new value is attached to the request in the Message Body in JSON format.

For example to change the name of the bridge `(/config/name)` we address the parent of this setting `(/config)` and send the new name with the request in the message body.

Address	`http://<bridge IP address>/api/<username>/config`
Body	`{"name":"Developer Bridge"}`
Method	`PUT`

If you're doing something that isn't allowed, maybe setting a value out of range or typo in the resource name, then you'll get an error message letting you know what's wrong. For much more information see the error section.

Sometimes, when you do a `GET`, the JSON response will have attributes that contain an object for a value – in other words a resource has a set of child resources.

For example:

Address	`http://<bridge IP address>/api/<username>/lights/1`
Method	`GET`

gives a JSON response with an attribute called "state". The value of state is an object which in turn contains a whole bunch of other attributes! To modify these we would need to address `/lights/1/state`.

Representations

The hue system is built on the principle of representations. Each object in the hue system has a corresponding resource in the hue interface (a representation of the real object). If you interact directly with these representations and afterwards the necessary Zigbee commands are sent out to lights to effect the change.

This lets us do things fast without worrying about the wireless messages bouncing between the lights, but it can mean that things are temporarily inconsistent. By this we mean the state of the lights and the state of their representation are not the same for a short period. It corrects itself after a while, but for example if you set all your lights to red and then power cycle a light so that it goes to white, it could take a few minutes for the bridge to update its representation.

The same thing is true for our portal with a representation there, which may be temporarily inconsistent with the bridge representation. This all lets us do things fast and deal with lots of concurrent users.

Controlling Light

There are many properties that can be controlled with hue – more than you might expect. All of these properties are in the `/state` resource of a light. Let's go through them and explain.

The Easy Ones

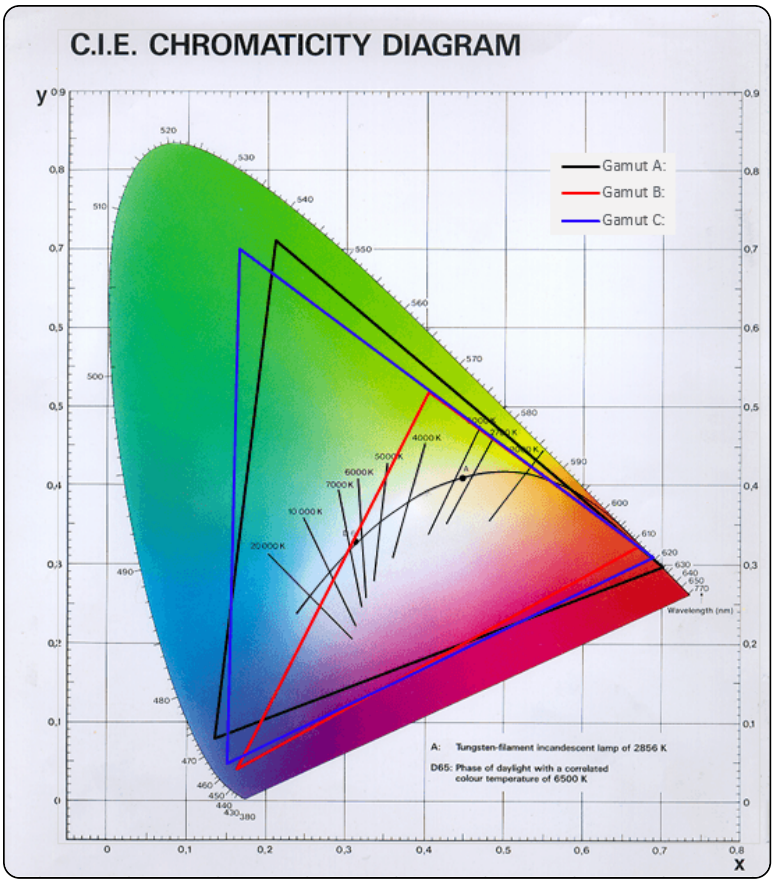
on – This is the easiest example. A light can be on or off. So setting this resource to true turns a light on to its last setting. Setting the resource to false turns the light off.

bri – This is the brightness of a light from its minimum brightness 0 to its maximum brightness 254 (note minimum brightness is not off, and the light will actually return 1 when set to 0 and return 254 when set to 255). This range has been calibrated so there a perceptually similar steps in brightness over the range. You can set the "bri" resource to a specific value e.g. the following commands sets the lights to 42/254 of their maximum brightness.

Address	`http://<bridge IP address>/api/<username>/lights/1/state`
Body	`{"bri":42}`
Method	`PUT`

Colors Get More Complicated

The color point of light has lots of ways of being quantified. The diagram below shows a plot of what is known as the CIE color space – the 3 triangles outline the colors which hue can address.



All points on this plot have unique xy coordinates that can be used when setting the color of a hue bulb. If an xy value outside of bulbs relevant Gamut triangle is chosen, it will produce the closest color it can make. To control lights with xy use the “xy” resource which takes the value of an array of two values between 0 and 1 e.g. `"xy":[0.675,0.322]` is red.

We can also choose to address the color point of light in a different way, using colors on the black curved line in the center of the diagram. This is the line that follows white colors from a warm white to a cold white. hue supports color temperatures from 2000K (warm) to 6500K (cold) with great white light. To set the light to a white value you need to interact with the “ct” (color temperature) resource, which takes values in a scale called “reciprocal megakelvin” or “mirek”. Using this scale, the warmest color 2000K is 500 mirek `("ct":500)` and the coldest color 6500K is 153 mirek `("ct":153)`. As with xy, the light will go to the closest value it can produce if the specified color temperature is outside of the achievable range.

The final way to choose the color of your lights is with hue and saturation. These values will depend on the color capabilities of the bulb you are using, but since all hue bulbs are the same that’s not a problem. The hue property at maximum saturation basically goes around all points on the edge of the bulbs Gamut triangle (all the most saturated colors).

Reducing the saturation takes this hue and moves it in a straight line towards the white point. So `"sat":25` always gives the most saturated colors and reducing it to `"sat":200` makes them less intense and more white.

Below are some common hue values and the corresponding xy values (for a Gamut B bulb). See the XY Values page for a more comprehensive table of the

145 most common colors showing XY values for each of the 3 Gamuts.

Extra Fun Stuff

Hue	Final-X	Final-y
0	0.3972	0.4564
12750	0.5425	0.4196
25500	0.41	0.51721
46920	0.1691	0.0441
56100	0.4149	0.1776
65280	0.6679	0.3181

As with bri you can choose to set “hue” and “sat” to an absolute value.

So this command sets a light to blue:

Address	`http://<bridge IP address>/api/<username>/lights/1/state`
Body	`{"bri":46920}`
Method	`PUT`

Limitations

With these commands you can control any number of your lights in any way you want, just address the correct light resource and send it the command you want.

We have some limitations to bear in mind:

We can’t send commands to the lights too fast. If you stick to around 10 commands per second to the `/lights` resource as maximum you should be fine. For `/groups` commands you should keep to a maximum of 1 per second.

If you try and control multiple conflicting parameters at once e.g. `{"ct":250,"xy":[0.5,0.5]}` the lights can only physically do one, for this the following rule applies: xy beats ct beats hue, sat – Simple.

All your lights have to be in range. If you can control them from the hue app then you’re fine to play with them via the developer interface. Remember all lights act as a signal repeater so if you want more range put a light in between.

Connect with us