PHILIPS

hue PERSONAL WIRELESS LIGHTING

How to **develop for Hue?**

Develop   Get Started   Application Design Guidance   **Hue API**   Hue Entertainment   Tools and SDKs

Overview

Datatypes and Time Patterns

1. Lights API

2. Groups API

3. Schedules API

4. Scenes API

5. Sensors API

6. Rules API

7 Configuration API

8. Info API (deprecated as of 1.15)

9. Resourcelinks API

10. Capabilities API

Remote Authentication

Applications have to authenticate with the Hue Remote API and allow users to authorize them to use their Hue lighting system remotely. When you register with Hue, an appid, clientid and clientsecret are provided to authenticate your applications with the Hue Remote API. With these credentials you are to ask users to allow your application access to their system remotely. If the user grants your application access, the Hue Remote API will grant an authorization_code that can be used to request an access_token. This access_token is to be included as an HTTP Authorization header as a Bearer token for every call that you make to the remote API.

Example:

```
> Authorization: Bearer Y2xpZW50X2lkOnNlY3JldA==
```

The steps for succesfully authenticating your application and using the Hue Remote API succesfully are described below in greater detail.

Authorization request

URL	`https://api.meethue.com/oauth2/auth`
Method	`GET`
Permission	valid clientid

Sample Request:

```
 >GET https://api.meethue.com/oauth2/auth?clientid=<clientid>&
appid=<appid>&deviceid=<deviceid>&devicename=<devicename>&state=
<state>&response_type=code
```

On this page:

Get Token

Refresh Token

Data Rate Limits

Remote API Quick start guide

Remote Authentication

Remote Hue API – Error Messages

Error messages

Message Structure and Response

Supported Devices

API Documentation Changelog

Glossary terms

Description

This is the initial step in the authorization flow, in which there will be a redirect to the meethue login portal for a user to grant permissions to the resources. As query parameters a valid clientid , and a response_type should be provided. The clientid will be supplied by the Hue team as soon a developer is registered and accepted the terms of use. The only allowed response_type is " code "

Query parameters

Name	Value	Description	Required
clientid	The clientid you obtain from Hue	Identifies the client that is making the request. The value passed in this parameter must exactly match the value you receive from hue. Note that the underscore is not used in the clientid name of this parameter.	Required
state	any string	Provides any state that might be useful to your application upon receipt of the response. The Hue Authorization Server roundtrips this parameter, so your application receives the same value it sent. To mitigate against cross-site request forgery (CSRF), it is strongly recommended to include an anti-forgery token in the state, and confirm it in the response. One good choice for a state token is a string of 30 or so characters constructed using a high-quality random-number generator.	Required
deviceid	string	The device identifier must be a unique identifier for the app or device accessing the Hue Remote API.	Required
devicename	string	The device name should be the name of the app or device accessing the remote API. The devicename is used in the user's "My Apps"	Optional

Name	Value	Description	Required
		overview in the Hue Account (visualized as: “<app name> on <devicename>”). If not present, deviceid is also used for devicename. The <app name> is the application name you provided to us the moment you requested access to the remote API.	
appid*	The appid you obtain from Hue	Identifies the app that is making the request. The value passed in this parameter must exactly match the value you receive from hue.	Required
response_type	code	The response_type value must be “code”.	Required

* This parameter might be removed in the future.

Example

The response will be sent to the redirect_uri which you specified the moment you registered for access. The user is first directed to the meethue.com website where he can approve the access to his Hue lighting system from the application. If the user approves the access request, then the response contains an authorization code (e.g. ?code=oMsCeLvIaQm6bTrgtp7) and the state parameter. If the user does not approve the request, the response contains an error message.

Sample Request:

```
 > GET https://api.meethue.com/oauth2
/auth?clientid=jq5AQ0zHi5uwhKqzbviNfiG4WXAaqd4F&
response_type=code&state=xUvdhs&appid=myappid&deviceid=mydeviceid&
devicename=mydevicename
```

Sample Response:

```
 < HTTPS/1.1 302 Moved
 < Content-Type: text/plain
 < Location: https://<redirect-
uri>/login?clientid=jq5AQ0zHi5uwhKqzbviNfiG4W&response_type=code
```

Get Token

URL	`https://api.meethue.com/oauth2/token`
Method	`POST`

URL	`https://api.meethue.com/oauth2/token`
Permission	valid authentication code

Description

This endpoint is intended to exchange the code obtained in 10.1.1 for a set of access and refresh tokes. The returned access_token can be used by the application to access the user's Hue resources remotely. As query parameters valid code and grant_type parameters must be provided. The code parameter is the authentication code as received at the callback uri. The grant_type must be "authorization_code ". With these two parameters you will be able to complete a *Digest* or *Basic* authorization flow, which we will explain in detail.

The response will contain an access_token and a refresh_token . The access_token will be only valid for a short time, which means that the application has to refresh the access_token after expiration of the access_token and before the expiration time of the refresh_token , otherwise the user has to go through the authorization step again. The expire times of the access_token and the refresh_token are part of the response

Sample Request:

```
 > POST https://api.meethue.com/oauth2/token?code=pP5J8YN8&
grant_type=authorization_code
```

Sample Response:

```
 < HTTPS/1.1 401 Unauthorized
 WWW-Authenticate: Digest realm="oauth2_client@api.meethue.com",
nonce="7b6e45de18ac4ee452ee0a0de91dbb10"
```

In this example you'll notice that you have not received an access_token in response to your request, even though a valid authentication code was sent as a query parameter. Hue still needs to *verify that it is in fact your application* requesting the access_token on the user's behalf! You will have to add an *Authorization header* to the call to /oauth2/token so Hue knows it really is your application that is making the request.

There are two types authorization headers possible for getting an access_token : the Hue Remote API supports *Digest* and *Basic* authentication methods. We recommend using Digest for your applications, as with this method you will be able to handle your credentials in a more secure way.

Digest Authentication

HTTP Digest authentication is based on a challenge-response handshake. In the example response above you'll find that the Hue Remote API response contains additional information in a WWW-Authenticate header. This information can be used for constructing a

Digest Authorization header.

Requesting Challenge:

```
 > POST https://api.meethue.com/oauth2/token?code=pP5J8YN8&
grant_type=authorization_code
```

Note: This post to /oauth2/token endpoint should not contain an *Authorization header* or credentials as *form parameters.*

Response:

```
 < HTTPS/1.1 401 Unauthorized
 WWW-Authenticate: Digest realm=”oauth2_client@api.meethue.com”,
nonce=”7b6e45de18ac4ee452ee0a0de91dbb10”
```

Note: nonce numbers will only stay valid for a limited period.

With this nonce , we now have all information we need to build a Digest header to accompany our token request. The Digest header contains a response variable that only your applications can build.

Get token with Digest:

```
 > POST https:api.meethue.com/oauth2/token?code=pP5J8YN8&
grant_type=authorization_code
 > Authorization: Digest username="<clientid>",
realm="oauth2_client@api.meethue.com", nonce="<nonce>",
uri="/oauth2/token", response="<response>"
```

Location	Parameter	Value
Query	code	The code you received in “authorization request” step. This code is only valid for about 10 minutes and 1 time use only.
Query	grant_type	Must be “authorization_code”
Header	Authorization	Digest username=”<clientid>”, realm=”oauth2_client@api.meethue.com”, nonce=”<nonce>”, uri=”/oauth2/token”, response=”<response>”

The Digest header above consists of comma-separated parameters in one single Authorization header:

- The username value is the clientid Hue provided you with.
- The nonce is the value you got from the challenge.
- The response parameter in the Digest header is unique for every token request and must be calculated.

Calculating digest response

The response variable in the Authorization header is calculated from a set of MD5 hashed *string concatenations* . The response is calculated as follows:

Parameter	Value
HASH1	MD5(“CLIENTID” + “:” + “REALM” + “:” + “CLIENTSECRET”)
HASH2	MD5(“VERB” + “:” + “PATH”)
response	MD5(HASH1 + “:” + “NONCE” + “:” + HASH2)

In pseudo code, this would translate into the following:

```
 var HASH1 =
MD5("kVWjgzqk8hayM38pAudrA6psflju6k0T:oauth2_client@api.meethue.com:
 var HASH2 = MD5("POST:/oauth2/token");
 var response = MD5(HASH1 + ":" + "7b6e45de18ac4ee452ee0a0de91dbb10"
```

The values needed for performing these MD5 hashing operations should look familiar:

Parameter	Value
CLIENTID	The clientid you have received from Hue when registering for the Hue Remote API.
REALM	The realm provided in the challenge “401 Unauthorized” response (i.e. “oauth2_client@api.meethue.com”).
CLIENTSECRET	The clientsecret you have received from Hue when registering for the Hue Remote API.
VERB	The HTTPS verb you are using to request the token (i.e. “POST”).
PATH	The path you are making your request to (i.e. “/oauth2/token”).
NONCE	The nonce provided in the challenge “401 Unauthorized” response.

Sample Response:

```
 < HTTPS/1.1 200 OK
 < Content-Type: application/json
 {
     "access_token":"jWH1al4ncKzu41u40dWckZFAAUxU",
     "access_token_expires_in":"3599",
     "refresh_token":"AaVBPYuxs6MxGTFasV7QdZ20Yq7unwVo",
     "refresh_token_expires_in":"7199",
     "token_type":"BearerToken"
 }
```

Note: For security reasons the nonce provided will only be valid for a short period of time. In case you are doing all things right, but still get a 401 Unauthorized, completing this flow might take too long.

Basic Authentication

Besides the Digest method, the Hue Remote API supports Basic authentication via both form and query parameters. For Basic, instead of sending a Digest header as described above (i.e. Authorization: Digest etc…) you would need to send a Basic authorization header that includes your base64 encoded clientid and clientsecret. Note that by doing this you are relying heavily on TLS/SSL encryption for hiding your clientid and clientsecret. This comes with additional security risks, particularly for mobile apps or other environments that are not under your complete control, and is therefore discouraged.

For obtaining an access_token with Basic Authentication this header is required: Authorization: Basic <base64(clientid:clientsecret)>

Sample Request:

```
> POST /oauth2/token?code=pP5J8YN8&grant_type=authorization_code
> Authorization: Basic Y2xpZW50X2lkOnNlY3JldA==
```

Sample Response:

```
< HTTPS/1.1 200 OK
< Content-Type: application/json
{
    "access_token":"jWH1al4ncKzu41u40dWckZFAAUxU",
    "access_token_expires_in":"86399",
    "refresh_token":"AaVBPYuxs6MxGTFasV7QdZ20Yq7unwVo",
    "refresh_token_expires_in":"172799",
    "token_type":"BearerToken"
}
```

Refresh Token

URL	`https://api.meethue.com/oauth2/refresh`
Method	`POST`
Permission	valid refresh token

Description

Exchange a valid refresh token previously received with a new set of access and refresh tokens.

Similar as with requesting an access_token, there are two methods available for refreshing the access_token, i.e. Digest and Basic authorization. Refreshing the token using Digest is based on the same challenge-response handshake flow as the Digest method for requesting an access_token, and is preferred to be used.

An example of Digest and Basic authorization is given below.

As query parameter a valid grant_type should be provided. The provided grant_type should be set to the string “refresh_token”. Additionally, two headers (Content-Type and Authorization) and a form parameter (refresh_token) must be set.

Location	Parameter	Value
Header	Content-Type	Must be “application/x-www-form-urlencoded”
Header	Authorization	*Digest* or *Basic* authentication
Query	grant_type	Must be “refresh_token”
Form	refresh_token	The obtained refresh token

Sample Request (Digest):

```
 > POST /oauth2/refresh?grant_type=refresh_token
 > Authorization: Digest
username="kVWjgzqk8hayM38pAudrA6psf1ju6k0T",
 realm="oauth2_client@api.meethue.com",
 nonce="7b6e45de18ac4ee452ee0a0de91dbb10",
 uri="/oauth2/refresh",
 response="39fcfbbea89b3cf9d0547f0c838d1e27"
 > Content-type: application/x-www-form-urlencoded
 refresh_token=ArgjAYZnLnaqQ94SgR8waZl2t78Q8dTr
```

Sample Response:

```
 < HTTPS/1.1 200 OK
 < Content-Type: application/json
 {
     "access_token":"AtVzDnS7ALBNtqzcTpzdiCSOoIXb",
     "access_token_expires_in":"86399",
     "refresh_token":"ZntBGpAAyIptBL47tKFmUimqDRErqWH3",
     "refresh_token_expires_in":"172799",
     "token_type":"BearerToken"
 }
```

Sample Request (Basic):

```
 > POST /oauth2/refresh?grant_type=refresh_token
 > Authorization: Basic Y2xpZW50X2lkOnNlY3JldA==
 > Content-type: application/x-www-form-urlencoded
 refresh_token=ArgjAYZnLnaqQ94SgR8waZl2t78Q8dTr
```

Sample Response:

```
 < HTTPS/1.1 200 OK
 < Content-Type: application/json
 {
     "access_token":"AtVzDnS7ALBNtqzcTpzdiCSOoIXb",
     "access_token_expires_in":"86399",
     "refresh_token":"ZntBGpAAyIptBL47tKFmUimqDRErqWH3",
     "refresh_token_expires_in":"172799",
     "token_type":"BearerToken"
 }
```

Data Rate Limits

We want developers to create compelling user experiences, but we also want the Hue remote services to always be available for the users. Clients that make a large number of requests in a given period of time can impact hue services, so we apply rate limits. Rate limiting restricts the number of requests for a given time period. If you exceed the limit, you will get a response code 429 (too many request) for subsequent requests. As we learn more about client usage patterns and their impact on the hue remote service we may find it necessary to modify rate limits. We strongly encourage you to build your client apps to use the minimum number of calls required to build a compelling user experience, and to deal with the rate limit violations appropriately.

Connect with us

Contact Terms & Conditions Privacy Product Security