DAIKIN
AIR INTELLIGENCE

IM-DKNAPI-201908

API Integration Manual

For

Daikin DKN Cloud Wi-Fi Adaptor

TABLE OF CONTENTS

1 INTRODUCTION

The following document describes the use of the DKN Cloud NA third party API, further on called *Open API*, including the authentication flow and all available requests/actions.

2 AUTHENTICATION

The authentication mechanism, which allows a third party service to connect to the DKN Cloud NA ecosystem, is based on the *OAuth2* standard implementation. *OAuth2* is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. The following sections will provide an overview of *OAuth2* roles, the *Authorization Code* grant type (Authorization flow used for the Open API) and how it's been adopted to work with the Open API. This section is guided towards developers.

2.1 Oauth2 Fundamentals

2.1.1 Oauth Roles

Oauth defines four roles:

- **Resource Owner,** *User*: The resource owner is the *user* who authorizes an application to access their account. The application's access to the user's account is limited to the "scope" of the authorization granted (e.g. read or write access). In this case, the user corresponds to a registered user in the DKN Cloud NA App, and the allowed "scopes" by the Open API are the user's devices and groups information.
- **Resource / Authorization Server**, *API*: The resource server hosts the protected user accounts, and the authorization server verifies the identity of the user then issues access tokens to the application. In the Open API implementation, the DKN Cloud NA ecosystem fulfills both the resource and authorization server roles.
- **Client**, *Third Party Application*: The client is the *third party application* that wants to access the user's account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the Server.

2.1.2 Third Party Client Application Registration

Before using Oauth, the third party client application must be registered with the Open API service, where the following details of the application must be provided:

- Application Name
- List of Redirect URI or Callback URL

The redirect URI is where the Open API will redirect the user after they authorize (or deny) the application, and therefore the part of the application that will handle authorization codes or access tokens. This must be seen as a list of valid URLs. As we will see next, on the authentication process using a web interface based flow, the application must specify a redirect URL. This URL must match one of the registered URLS. This is a safety measure used to ensure that the user will only be directed to appropriate locations.

As of now, this registration is done exclusively on demand, and requires manual interaction of the system administrator of the DKN Cloud NA ecosystem.

2.1.3 Client ID and Client Secret

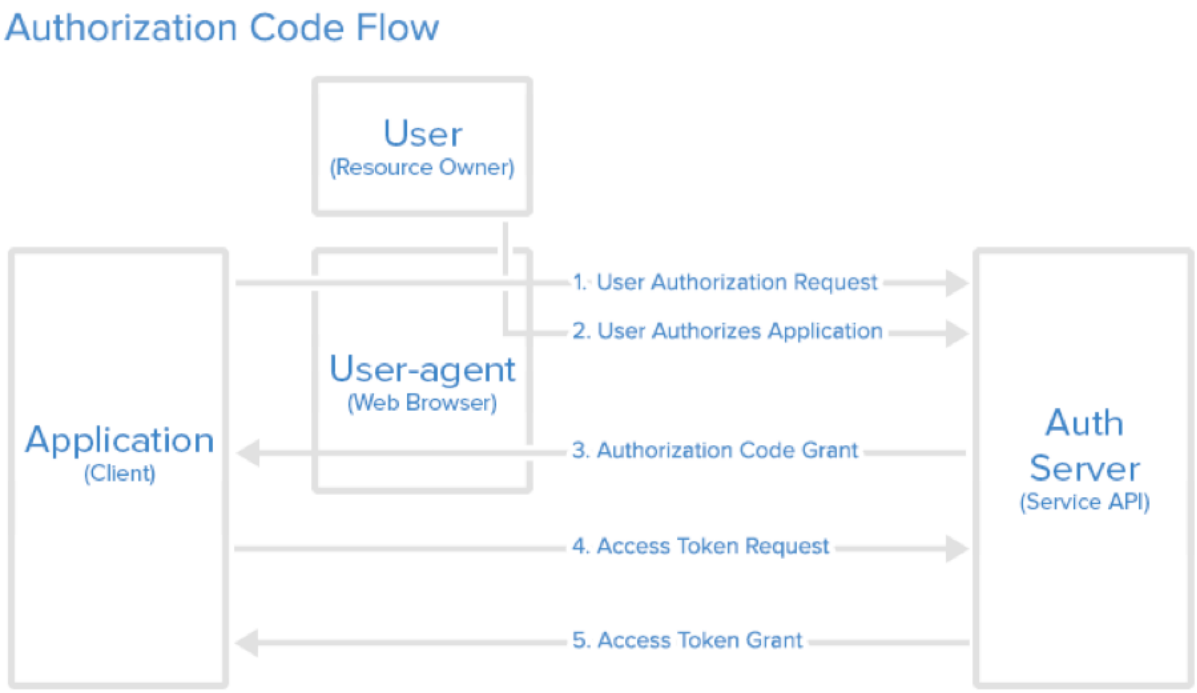
Once the third party application is registered, the Open API service will issue "client credentials" in the form of a *client identifier* and a *client secret*. The Client ID is a publicly exposed string that is

used by the service API to identify the application, and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user's account, and must be kept private between the application and the API.

2.2 Oauth2 Authorization Code Grant Type

OAuth 2 defines four grant types, each of which is useful in different cases. The one used for the Open API corresponds to the *Authorization Code Grant Type*. It's the most commonly used because it is optimized for *server-side* applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained. This is a redirection-based flow, which means that the application must be capable of interacting with the user-agent (i.e. the user's web browser) and receiving API authorization codes that are routed through the user-agent. Nevertheless, it can also be used with other server-side services which lack of web interface (*programmatic interface*).

Here describes the authorization code flow:



The above diagram represents a typical authorization flow.

Here is a more detailed explanation of the steps in the diagram:

1. The application requests authorization to access service resources from the user
2. The user must log in to the Open API environment and authorize the request.
3. If the user authorized the request, the application receives an authorization grant (Code)
4. The application requests an access token from the authorization server (API) by presenting authentication of its own identity (Client Secret and Client ID), and the authorization grant (Code).
5. If the application identity is authenticated (registered in the Open API environment) and the authorization grant is valid (Code hasn't expired or already been used), the authorization server (API) issues a pair of access token/refresh token to the application. Authorization is complete.

From now on, the application is authorized to make requests to the API in behalf of the user, provided the request presents the access token for authentication. If the access token is valid, the resource server (API) serves the resource to the application.

2.2.1 Refresh Token

The authorization response contains a pair of tokens: an access token and a refresh token. The access token is the token used in every request made to the API to authenticate the request in behalf of the user. The refresh token on the other hand is used for requesting a new access token when this one expires.

Once the application has an access token, it may use the token to access the user's account via the API, limited to the scope of access, until the token expires or is revoked. After an access token expires, using it to make a request from the API will result in an "Invalid Token Error" (401 status). At this point, the refresh token can be used to request a fresh access token from the authorization server.

Now that the Oauth2 foundations have been explained, we will describe in detail the different requests and responses handled by the Open API. The following must be used as an implementation reference by the third party services for a successful integration with the DKN Cloud NA's Open API.

2.3 Open API OAuth2 Implementation

As prerequisites, users must have a valid account in both environments (third party application and DKN Cloud NA), the third party application must be registered as an authorized entity by DKN Cloud NA (meaning that the third party will have a valid client_id/client_secret token pair), and for third party applications with web interface, a valid Redirect URI.

Since the authentication flow involves the linking of accounts between services, the user must first log in to DKN Cloud NA and authorize the third party service to access his information. This can be done in two ways: through the DKN Cloud NA website (for third party services with their own websites/applications) or through the programmatic interface (recommended for integration and automation systems).

All requests must be done using HTTPS protocol, and will throw an error if trying to use plain HTTP.

Note: *the following HTTP request examples will be done using the cURL agent format.*

2.3.1 Web Interface

Step 1. If the third party application wants to make the authorization through a web interface based flow, the application must redirect the user to the following authorization URL:

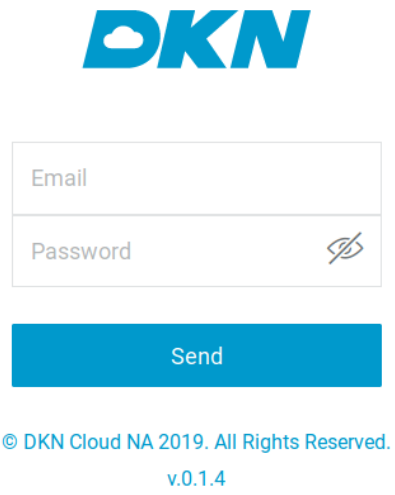
https://dkncloudna.com/#/oauth?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=installations+devices&state=STATE_TOKEN

Here is an explanation of the link components. The parameters are set in *querystring* format:

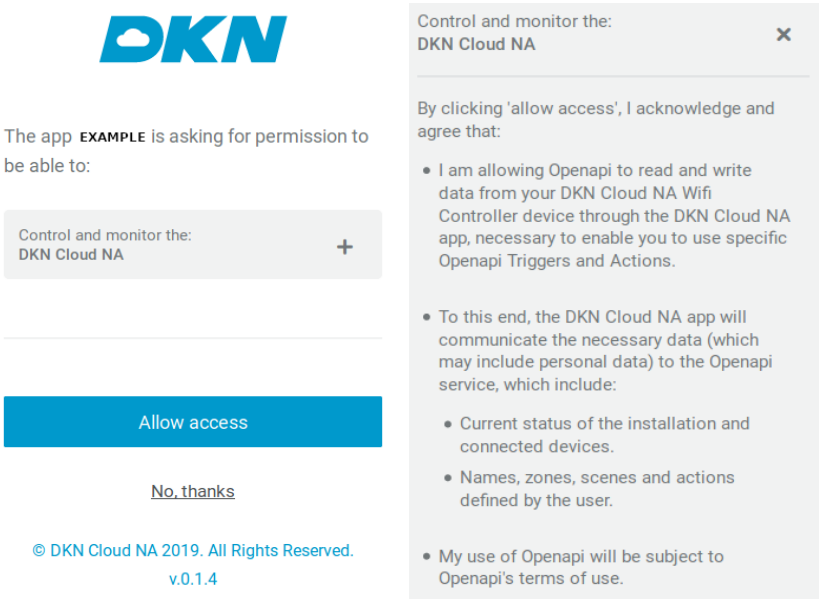
- https://dkncloudna.com/#/oauth the API authorization endpoint.
- client_id=CLIENT_ID: the application's client ID (how the API identifies the application).

- redirect_uri=CALLBACK_URL: where the service redirects the user-agent after an authorization code is granted. This value must match one of the valid redirect URLs specified in the registration process.
- response_type=code: specifies that your application is requesting an authorization code grant.
- scope=installations+devices: specifies the level of access that the application is requesting. In the case of Open API, the scopes must be installations and devices.
- state=STATE_TOKEN: this parameter preserves, as an encoded format, some state object set by the client in the Authorization request and makes it available to the client in the response. Used to mitigate CSRF attacks (explained later in more detail).

When the user clicks the link, they must first log in to the service, to authenticate their identity (unless they are already logged in). In case the user isn't logged in to DKN Cloud NA, this link will redirect the user to the following login page:



Step 2. Once the user is logged in, an authorization site will appear which will request the user to authorize or deny the third party service to access the user's information regarding installations and devices stored in DKN Cloud NA:



In this screenshot, the name of the third party application would be visible where the **EXAMPLE** text is set. This same is the name provided on the registration process.

Step 3. Once the user authorizes the third party service, the web page will redirect the user-agent to the application redirect URI specified in the first request, along with an *authorization code*. This redirect URI must match one of the valid redirect URLs specified during the client registration. The authorization code is a temporary code that the client will exchange for an access token. It will be passed in *querystring* format. The redirect would look something like this (assuming the application redirect URL is "example.com"):

https://example.com/?code=hLRlrpTIRUz96crm9PmH5PafXbE4JOAV

At this point, the application has a valid *authorization code* and can request an Access token. The remaining requests involved will be explained after describing the programmatic interface access, since they are the same for both flows.

2.3.2 Programmatic Interface

This interface basically performs the actions described above (Login and Authorize), but without the need of a browser; only through HTTPS requests. This method is suited for the integration in home automated systems/BMS. We will describe the structure and format of each of the requests.

Step 1 and 2. Login:

- REQUEST:
- POST https://www.dkncloudna.com/api/v1/auth/login/dknUsa
- HEADERS:
    - Accept: application/json
    - Content-Type: application/json
- BODY: JSON body parameters:
    - ***email***: (String) user email
    - ***password***: (String) user password
- EXAMPLE:

```
curl --location --request POST "https://www.dkncloudna.com/api/v1/auth/login/dknUsa" \
--header "Accept: application/json" \
--header "Content-Type: application/json" \
--data "{\"email\": \"user@email.com\", \"password\": \"examplePassword\"}"
```

- RESPONSE: This returns a JSON object, where we only are interested in only one of its properties: *token.*

  Example response:

```
{
                                    …
"token": "R7k6xX3HsHH1LIPYrCV5SbtYNH7rzFlb…",
                                    …
}
```

The value of this field is needed for the next request. It acts as an authentication token for the DKN Cloud NA environment, identifying the request in behalf of the user.

Step 3. Authorization of the third party service:

- REQUEST:
- POST https://www.dkncloudna.com/api/v1/users/oauth2/authorize
- HEADERS:
    - Accept: application/json
    - Content-Type: application/json
    - Authorization: Bearer **TOKEN_VALUE**

      -The value in the Authorization header is the value of the *token* response property of the Login response.
- BODY: JSON body parameters:
    - ***entityName***: (String) name of the third party entity, as registered when its credentials were issued.
    - ***client_id***: (String) value of the *client_id* token issued for the entity
    - ***scopes***: (String[]) scopes to authorize. Currently this property must equal to an array of two String elements => *devices, installations*
- EXAMPLE:

```
curl --location –request POST "https://www.dkncloudna.com/api/v1/users/oauth2/authorize?" \
--header "Content-Type: application/json" \
--data "{\"entityName\": \"ENTITY_NAME\", \"client_id\": \"CLIENT_ID\", \"scopes\": [\"devices\",
\"installations\"]}"
```

- RESPONSE: This returns a JSON object, with only one property: *redirectUri.* The value of this property will be one of the registered redirect URLs of the entity plus the *code* parameter (same format as described in the Web interface process).

Example response:

```
{
 " redirectUri": "https://example.com/?code=9zHBr2TwEkUanHTlLGvs5EjCPOvdBAHF "
}
```

The remaining requests are common for both interfaces.

Step 4: Application Requests Access Token. The value of *code* is used by the third party to ask DKN Cloud NA for the first access token on behalf of the user. This is a one time use, short lived token, so it must be used immediately after being received. This code is used in the following request:

- REQUEST:
- POST https://www.dkncloudna.com/api/v1/open/oauth2/token
- HEADERS:
    - Content-Type: application/x-ẃww-form-urlencoded
    - Accept: application/json
- BODY: application/x-www-form-urlencoded type body parameters:
    - ***grant_type***: (String) its value must be "*authorization_code*".
    - ***client_id***: (String) value of the *client_id* token issued for the entity.
    - ***client_secret***: (String) value of the *client_secret* token issued for the entity.
    - ***code***: (String) value of the *code* parameter acquired in the previous request.
- EXAMPLE:

```
curl --location --request POST "https://www.dkncloudna.com/api/v1/open/oauth2/token" \
--header "Content-Type: application/x-www-form-urlencoded" \
--form "grant_type=authorization_code" \
--form "client_id=CLIENT_ID" \
--form "client_secret=CLIENT_SECRET" \
--form "code=CODE"
```

- RESPONSE: This returns a JSON object, with the following four properties:
  - ***access_token***: token used in every request made to the API to authenticate the request in behalf of the user.
  - ***token_type*****:** Doesn't apply; its value will always be: *Bearer*
  - ***expires_in***: (Number) value in seconds specifying the lifetime of the token. By default, it will be two hours
  - ***refresh_token***: used for requesting a new access token when this one expires.

  Example response:

```
{
 "refresh_token": "R7k6xX3HsHH1LIPYrCV5SbtYNH7rzFlb",
 "token_type": "bearer",
 "access_token": "duUB2GpPoEg5dBRg2giWfFb0EvBFKg3Q",
 "expires_in": 7200
}
```

Reached this point, the third party application is authorized to access the DKN Cloud NA environment through the Open API endpoints on behalf of the user.

When the access token expires, DKN Cloud NA will respond with 401 error codes to every request, with the following JSON body:

```
{
 "_error_description": "The access token is invalid or has expired",
 "error": "invalid_token"
}
```

The third party service will then need to request a new access token for the user. In order to do so, the service must use the value of the *refresh_token* property obtained in the response body of the previous request, and use it as shown in the following request:

- REQUEST:
- POST https://www.dkncloudna.com/api/v1/open/oauth2/token
- HEADERS:
  - Content-Type: application/x-ẃww-form-urlencoded
  - Accept: application/json
- BODY: application/x-www-form-urlencoded type body parameters:
  - ***grant_type***: (String) its value must be "*refresh_token*".
  - ***client_id***: (String) value of the *client_id* token issued for the entity.
  - ***client_secret***: (String) value of the *client_secret* token issued for the entity.
  - ***refresh_token***: (String) value of a valid refresh token for that user.
- EXAMPLE:

```
curl --location --request POST "https://www.dkncloudna.com/api/v1/open/oauth2/token" \
--header "Content-Type: application/x-www-form-urlencoded" \
```

```
--form "grant_type=refresh_token" \
--form "client_id=CLIENT_ID" \
--form "client_secret=CLIENT_SECRET" \
--form "refresh_token=REFRESH TOKEN"
```

- RESPONSE: This returns a JSON object, with the same properties as the previous request.

  Example response:

```
{
    "refresh_token": "vkVz6nVhtwb6cZgylCsAEev3eetcrRCG",
    "token_type": "bearer",
    "access_token": "3VOO9NScVG02ibR2ssrGAjIcnblRbG0I",
    "expires_in": 7200
}
```

  We can observe that the refresh token request is basically the same request as the previous one, with the difference in the request body parameters: for requesting a refresh token, the value of the *grant_type* property is now *refresh_token*, and the *code* property is substituted for the *refresh_token* property.

Both the access token and the refresh token can have it's *TTL (Time To Live)* parameter configured. By default, the Open API issues access tokens of a lifetime of two hours, while the refresh tokens never expire. This configuration is set up to avoid the need of doing the authentication process all over again in case the refresh token expires. Also, when issuing a new pair of access/refresh tokens with the refresh request, the old pair of tokens will be invalidated.

2.4 Oauth2 Best Practices

Before diving in the API itself, we will point out some best practices in order to implement a safe OAuth2 client.

- OAuth Client should avoid forwarding the user's browser to a URI obtained from a query parameter since such a function could be utilized to exfiltrate Authorization Codes and Access Tokens. This refers to the first step of the web interface flow, where the application redirects the user-agent to the DKN Cloud NA authorization URL. It's recommended not to ask another service for this URL, because the request could be intercepted by a malicious entity and return a different authorization URL. This could lead to redirecting the user-agent to a fake website, where the user would perform the authorization flow, leaving authorization codes and access tokens exposed. The recommended way of dealing with this auhtorization URL is storing it in the application's backend database, assuring the integrity of the URL.
- If there is a strong need for this kind of redirects, clients are advised to implement appropriate countermeasures against open redirection, e.g., as described by the OWASP.
- Clients must prevent CSRF and ensure that each Authorization Response is only accepted once. One-time use CSRF tokens carried in the "OAuth state parameter", which are securely bound to the User-agent, should be used for that purpose. This encourages the use of the state parameter in the Authentication flow as a method for the application to identify that the authorization process hasn't been tampered. When you use state for CSRF mitigation on the redirection endpoint, that means that within the state value there is a unique and non-guessable value associated with each authentication request about to be initiated. It's that unique and non-guessable value that allows you to prevent the attack by confirming if the value coming from the response matches the one you expect (the one you generated when initiating the request). The state parameter is a string so you can encode any other information in it. The way this works is that you send a random value when starting an authentication request and validate the received value when processing the response.

- In order to prevent OAuth 2.0 Mix-Up Attacks, OAuth Clients must only process redirect responses of the OAuth Authorization Server they send the respective request to and from the same user agent this Authorization Request was initiated with. Clients must memorize which Authorization Server they sent an Authorization Request to and bind this information to the user agent (browser sent from) and ensure any sub-sequent messages are sent to the same Authorization Server.

3 API

For every API request, the following headers are required:

- Accept: application/json
- Authorization: Bearer **{{ACCESS_TOKEN}}**

Where the value of *ACCESS_TOKEN* must correspond to a valid access_token for a specific user.

As for PUT requests, the following header is also required:

- Content-Type: application/json

The headers will be omitted in the following descriptions.

All responses are in JSON format.

3.1 Status Requests

3.1.1 Devices

- REQUEST:
- GET https://dkncloudna.com/api/v1/open/devices
- BODY: none
- RESPONSE:

  Example response:


```
[{
    "_id": "5cd51f1684c09013f6765d9c",
    "name": "Test group",
    "devices": [{
      "mac": "AA:BB:CC:DD:EE:0D",
      "icon": 1,
      "name": "Test"
    }],
    "timezoneId": "Europe/Madrid",
    "units": 0,
    "schedules": [{
      "devices": ["AA:BB:CC:DD:EE:0D"],
      "hour": 14,
      "minutes": 32,
      "power": "on",
      "mode": 1,
      "temp": 23,
      "vent": 3,
      "days": [ 0,1,2,3,5,6],
      "name": "Program_1",
      "enabled": true,
      "_id": "5be1b5d973269f2fcc88386b"
    }],
    "added_at": "2019-02-10T06:49:58.591Z",
    "type": "advanced"
}]
```

The response is an array of JSON objects, where each object is the description of an Installation (or group). Each installation consists of the following fields:

- ***_id*:** installation identifier. It will be used in every command request issued to a device.
- ***name*:** installation name, as it appears in the DKN Cloud NA app.
- ***timezoneId*:** timezone code for the installation.
- ***units*:** installation working units. Values:
  - 0: Celsius
  - 1: Fahrenheit
- ***added_at*:** date the installation was created.
- ***type*:** user's privileges over the installation. A user can have one of the following roles in an installation:
  - *basic*: user cannot create schedules; user has permission to control a subset of the installation's devices.
  - *advanced*: user has access to every device in the installation, as well as adding/deleting users to it.
- ***schedules*:** Array of JSON objects representing all the schedules an installation has configured:
  - *devices*: array of devices to which the schedule is applied.
  - *hour*: running hour, 24h format (0-23).
  - *minutes*: (0-59)
  - *power*: if the schedule turns on or off the ACs. Values:
    - on: turns on the ACs
    - off: turns off the ACs
  - *mode*: operation mode to be set. Values:
    - 1: auto
    - 2: cooling
    - 3: heating
    - 4: ventilation
    - 5: dry
  - *temp*: setpoint to apply.
  - *vent*: fan speed to apply. The accepted values vary for each unit (depends on the available speeds). The maximum range covers seven speeds and an auto speed:
    - 0: auto speed
    - [1, 7]: fan speeds.
  - *days*: array of numbers corresponding to the days the schedule applies. Values: [0, 6], where 0 corresponds to Sunday and 6 to Saturday.
  - *name*: name of the schedule.
  - *enabled*: (Boolean) whether the schedule is activated (true) or not (false).
  - *_id*: schedule unique identifier.

- **devices:** Array of JSON objects describing the linked devices from installation which the user has access to:
  - *mac*: device mac.
  - *icon*: number which identifies the device's icon as appears in the DKN Cloud NA app.
  - *name*: device name, as appears in the DKN Cloud NA app

3.1.2 Device State

Returns the state parameters of a device.

- RETURN
- GET https://dkncloudna.com/api/v1/open/{{INSTALLTION_ID}}/{{MAC}}
- BODY: none
- EXAMPLE:
- GET https://dkncloudna.com/api/v1/open/5cd51f1684c09013f6765d9c/AA:BB:CC:DD:EE:0D
- RESPONSE:

```
{
   "mode": <number>,
   "power": <boolean>,
   "setpoint": <number>,
   "temperature": <number>,
    "speed": <number>,
    "isConnected": <boolean>,
    "error": <number>,
    "errorStr": <string>,
    "warning": <number>,
    "warningStr": <string>
}
```

- ***mode**:* device's current operation mode.
- ***power***: *true* if on; *false* if off.
- ***setpoint***: setpoint for the current operation mode.
- ***temperature***: room temperature.
- ***speed***: fan speed.
- ***isConnected***: if device is online or not (connected to the cloud).
- ***error***: AC unit's error code. **If there's no error, this field will not be returned.**
- ***errorStr***: ASCII representation of the error. **If there's no error, this field will not be returned.**
- ***warning***: AC unit's warning code. **If there's no warning, this field will not be returned.**
- ***warningStr***: ASCII representation of the warning. **If there's no warning, this field will not be returned.**

3.2 Command Requests

The following requests execute idempotent actions over a device. Every action, in case of being correctly executed, will return the device's state after applying the command. The format of the response is the same as the previous device state request.

3.2.1 Device – State

Turn on/off the AC.

- REQUEST:
- PUT https://dkncloudna.com/api/v1/open/{{INSTALLTION_ID}}/{{MAC}}/state
- BODY: (JSON). Properties:
    - ***value***: (Boolean) true for on; false for off
- EXAMPLE:
- PUT https://dkcloudna.com/api/v1/open/5cd51f1684c09013f6765d9c/AA:BB:CC:DD:EE:0D/state

3.2.2 Device – Setpoint

Specify the AC's setpoint for the current operation mode.

- REQUEST:
- PUT https://dkncloudna.com/api/v1/open/{{INSTALLTION_ID}}/{{MAC}}/setpoint
- BODY: (JSON). Properties:
    - ***value***: (Number) value of the setpoint.
- EXAMPLE:
- PUT https://dkncloudna.com/api/v1/open/5cd51f1684c09013f6765d9c/AA:BB:CC:DD:EE:0D/setpoint

In case the value is out of range, the request will return the following error:

- Response code: 400
- Body:


```
{
   "_id": "outOfRange",
   "msg": "Value out of range",
   "data": {
      "validRange": {
         "minimumValue": {
            "value": 18
         },
         "maximumValue": {
            "value": 30
         }
      }
   }
}
```


Where the *minimumValue* and *maximumValue* objects represent the minimum and maximum values allowed respectively.

3.2.3 Device – Mode

Change the AC's operation mode.

- REQUEST:
- PUT https://dkncloudna.com/api/v1/open/{{INSTALLTION_ID}}/{{MAC}}/mode

- BODY: *(*JSON). Properties:
    - ***value****:* (Number) value representing a valid mode.
- EXAMPLE:
- PUT https://dkncloudna.com/api/v1/open/5cd51f1684c09013f6765d9c/AA:BB:CC:DD:EE:0D/mode

In case the value is not valid, the request will return the following error:

- Response code: 400
- Body:

```
{
   "_id": "modeNotSupported",
   "msg": "The thermostat doesn't support the specified mode."
}
```

3.2.4 Device – Speed

Change the AC fan's speed.

- REQUEST:
- PUT https://dkncloudna.com/api/v1/open/{{INSTALLTION_ID}}/{{MAC}}/speed
- BODY: *(*JSON). Properties:
    - ***value****:* (Number) value representing a valid speed.
- EXAMPLE:
- PUT https://dkncloudna.com/api/v1/open/5cd51f1684c09013f6765d9c/AA:BB:CC:DD:EE:0D/speed

In case the speed value is not valid, the request returns the following error:

- Response code: 400
- Body:

```
{
    "_id": "speedNotValid",
    "msg": "Speed value not valid",
    "data": {
       "validValues": [
          0,
          2,
          4,
          6
       ]
    }
}
```

A successful command for On/Off or Setpoint or Mode change or Fan speed change shall get a response with current condition of the unit:

```
{
   "mode": 2,
   "power": true,
   "setpoint": 65,
   "temperature": 79,
   "speed": 2,
   "isConnected": true
}
```

4 ERRORS

Apart from the previous specific error messages returned by certain requests, there are some common error responses that must be handled.

The error response always has the same format. Its body is a JSON object with at least two fields, and an optional one:

- ***_id*:** (String) Error id. Used for identifying the error type.
- ***msg:*** (String) generic error message describing the error.
- ***data*** (*optional*)*:* (Object) data associated with the error.

Here are the rest of the common errors, listed by their *id*:

- ***badJSON****:* Returned when the request body parameters, if expecting a JSON body, is a malformed JSON object. Error code: 400. Example response:

```
{
   "_id": "badJSON",
   "msg": "Malformed JSON"
}
```

- ***badParams****:* Returned when the request body parameters don't meet the interface (wrong parameters, wrong type, missing values). Error code: 400. Example response:

```
{
   "_id": "badParams",
   "msg": "Bad body parameters"
}
```

- ***commandNotSupported****:* Returned when the device doesn't support the requested command. Error code: 400. Example response:

```
{
   "_id": "commandNotSupported",
   "msg": "Command not supported"
}
```

- ***databaseErr****:* Returned when there has been an internal problem with the database (very unusual). Error code: 500. Example response:

```
{
   "_id": "databaseErr",
   "msg": "Error DB query"
}
```

- ***deviceNotConnected****:* Returned when trying to set a command to a device when this one isn't connected to the Cloud. Error code: 422. Example response:

```
{
   "_id": "deviceNotConnected",
   "msg": "Device currently not connected to the cloud"
}
```

- ***deviceNotFound****:* Returned when the specified MAC doesn't match to any existing device. Error code: 422. Example response:

```
{
   "_id": "deviceNotFound",
```

```
    "msg”: "No device found"
}
```


- ***installationNotFound:*** Returned when the installation id specified in the URL path doesn't exist or is invalid. Error code: 422. Example response:


```
{
    "_id": "installationNotFound",
    "msg": "No installation found"
}
```


machineError: Returned when trying to set a parameter to a device which has an AC error. In this case, the parameter is not set to the AC. The ASCII representation of the error code can be seen in the message's description. Error code: 422. Example response:


```
{
    "_id": "machineError",
    "msg": "Error in the AC unit: {{ERROR_ASCII}}"
}
```


- ***notAuthorized****:* Returned when the user doesn't have permission to act over the specified device. Example response:


```
{
    "_id": "notAuthorized",
    "msg": "Error in the AC unit: {{ERROR_ASCII}}"
}
```


- ***socketTimeout****:* Returned when there has been an internal error when communicating with other services (very unusual). Error code: 500. Example response:


```
{
 "_id": "socketTimeout",
 "msg": "SocketTimeout with DeviceService"
}
```


- ***unknown****:* Returned when there has been an unhandled internal error (very unusual). Error code: 500. Example response:


```
{
    "_id": "unknown",
    "msg": "Unknown Error. Try again later"
}
```


- ***userNotExist****:* Returned when the user making the request cannot be found in the database. Error code: 400. Example response:


```
{
    "_id": "userNotExist",
    "msg": "No user found with specified data"
}
```

WARNING

- Only qualified personnel must complete the installation.
- Consult your Daikin dealer regarding relocation and reinstallation of the remote controller. Improper installation may result in electric shock or fire.
- Electrical work must be performed in accordance with relevant local and national regulations, and with the instructions in this installation manual. Improper installation may cause electric shock or fire.
- Only use specified accessories and parts for installation. Failure to use specified parts may result in electric shock, fire, or controller damage.
- Do not disassemble, reconstruct, or repair. Electric shock or fire may occur.
- Only use specified wiring and verify all wiring is secured. Assure no external forces act on the terminal connections or wires. Improper connections or installation may result in electric shock or fire.
- Confirm power to the unit is OFF before touching electrical components.