Sigox APIs beginners

API How To

External Use, version 1.3

sigfox

Summary

Sigfox data interfaces

Sigfox provides 3 different interfaces to access data:

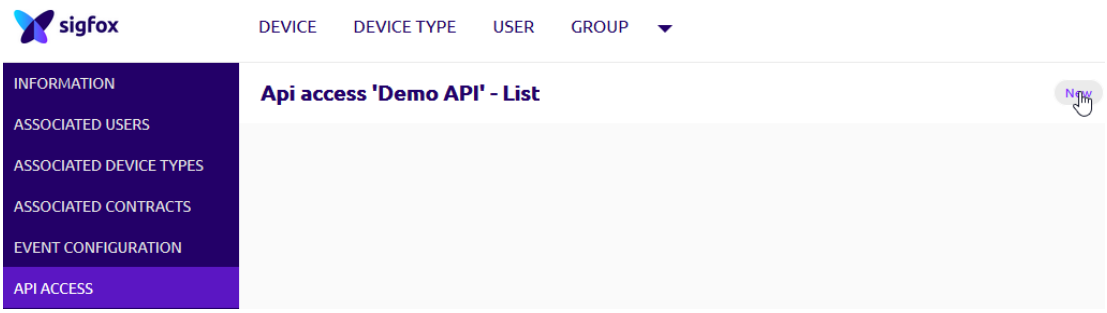
- Graphical User Interface (GUI) to access and alter data in a visual way, for human operators. All available data are accessible through GUI
- Callbacks to receive automatic notification upon specific event, as for example new messages from devices, newly activated device… The target of callbacks are computers running an HTTP server. Only event based data are eligible to callbacks
- API, to request, create or alter a specific data through an HTTPs request. The originator of the request is a computer running a computer program, to do specific tasks. Most of the data operations are available through API, such as group creation, device registration, callback creation…

How to use API

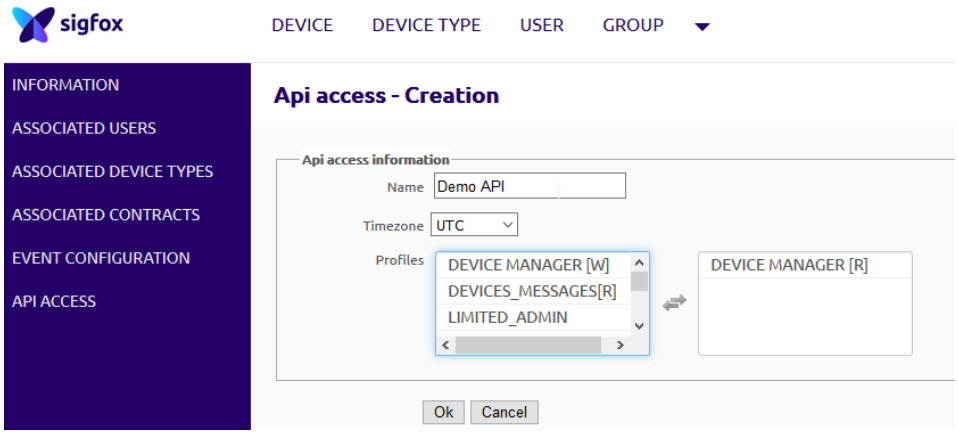
As APIs access is restricted to authenticated API user, the first step to access API is to generate the API credentials.

Credentials generation

To generate API credentials, you must go to the group that will support the API, then chose API Access in the submenu and hit the ‘New’ button to generate new credentials.

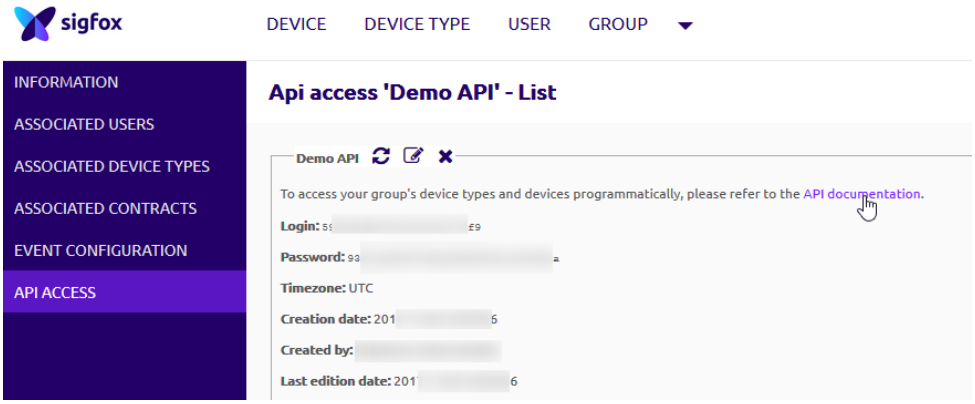


The interface will then prompt for the API name (descriptive info) and applicable time zone and, the most important part, the accessible roles for this new API. The latest info will determine what would be available with this API credentials. For example, with a read only role, the API won’t allow any creation or update, but only access data reading.



API documentation

Once the API is created, the documentation is accessible through the GUI in the same submenu, beside the API name, with the ‘Documentation’ link. This documentation is restricted to the operation allowed with this API credentials (update primitives aren’t accessible with API with read only roles for example)

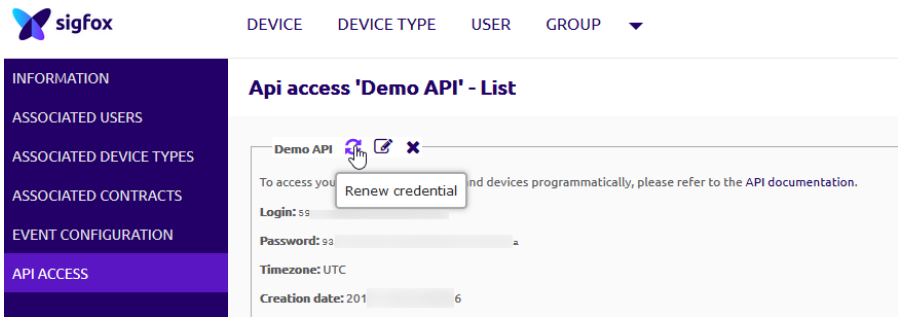


API usage

Following the API documentation, the REST principle is use for sigfox API: any access to APIs is done through an authenticated HTTPS request, with the URL shown in the documentation, and the potential parameters as explained in the documentation. The result code of the HTTP request will be used to check the correct behavior (20x result code indicate success), and the return data will give the optional data (such as job ID, requested data…)

Credentials renewal

To renew API credentials, either to change the login password in preventive mode or after an attack, you must go to the group that will support the API, then chose API Access in the submenu and click on the ‘Renew’ button to generate new credentials with the same scope.



Note that after renewing credentials, all systems that used the previous credentials have to be updated with the newest login and password values.

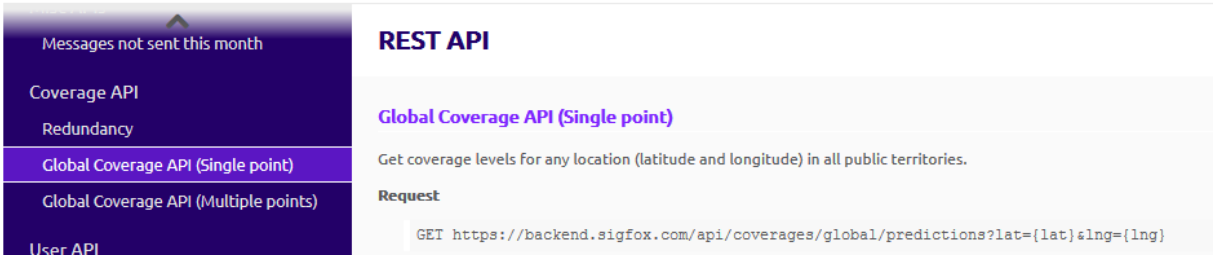
Step by step example: get coverage information

API scope

To access coverage information, an API must be created with read roles: Device Manager Read or Customer Read for example is fully enough to access this API primitive. For API creation, especially for production APIs, it is highly recommended to use strict, less permissive roles: in case of security breach, having low scope APIs role exposed will contain the breach.
For the same reason, if multiple systems use APIs, it is highly recommended to have at least one credentials pair per system. If one of them is compromised, it will be easier to renew the credentials (using the renew button) for this system and change its credentials than changing all the credentials of other systems.

Understanding documentation, preparing API request

Once the API credentials are created, the documentation in section ‘Coverage API’ shows the different primitives, including ‘Global Coverage API (Single point), which is perfectly suitable for this example: the description of this API is to get coverage levels for any location



The API description shows that two mandatory parameters must be provided, and one is being optional:

Request

```
GET https://backend.sigfox.com/api/coverages/global/predictions?lat={lat}&lng={lng}
```

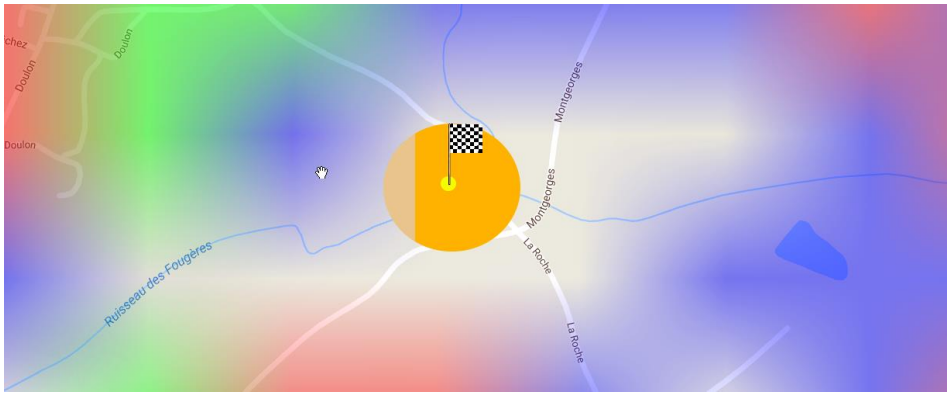
Parameters:

- `lat`: the latitude. Must be between -90° and 90°.
- `lng`: the longitude. Must be between -180° and 180°.

Optionally, the request can also have the following parameter:

- `radius`: (optional) the distance around the lat/lng point in meters, default is 300

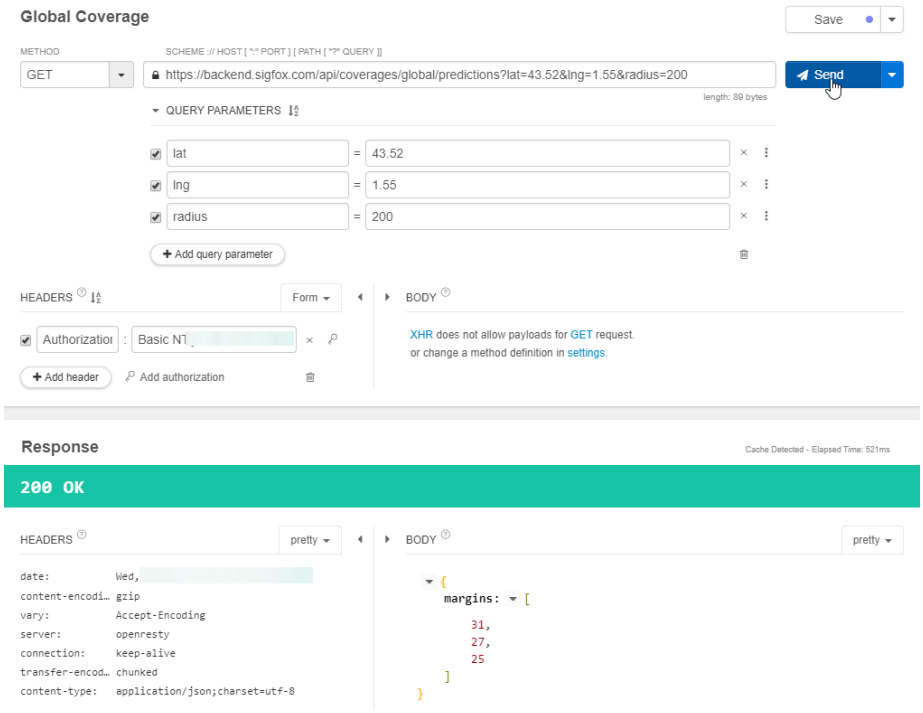
The latitude and the longitude are mandatory. The radius is optional. Note that if the input data is a postal address, corresponding latitude and longitude must be retrieved using distinct API, either using google, mapbox, opencage, openaddresses or any others geocoding APIs.
The radius, the optional parameter, will define the surface around the test point to consider. As described in the documentation, 300 meters is the default value, but we can set a value down to 200 meters.
Note that the coverage simulation accuracy doesn't go down to 1 meter precision (two nearby coordinates will have same the coverage value, coverage "definition" is above several dozen of meters), thus the global coverage doesn't allow values below 200 meters.
Considering 200 meters radius will avoid most mixed results from two adjacent simulated surfaces that might lead to average value between a non-covered and a covered zone. For example, with the image bellow, when using the orange radius, there will be a mix between a covered zone (at the left of the flag represented by a pastel-orange surface) and a non-covered zone (full orange surface). On the opposite, with the smaller yellow radius, there will be no coverage for the same coordinates. Thus, with the same position, with an orange -bigger- radius, the coverage will exist (with low coverage values), whereas with the yellow -smaller- radius there will be no coverage.



Once longitude, latitude and optionally radius have been set, the request can be sent to the API endpoint. For example, to test the API call, you can use a browser plugin such as RESTlet client for Chrome, a command line command such as curl with linux, or a programing language such as python, java or others.

Implementation

With RESTlet plugin:



Tip: Enter the URL and click on “Query parameters” to add the desired parameters, and on “Add authorization” for the credentials before hitting the “Send” button

With curl command line:

```
curl  --get --data lat=43.52 --data lng=1.55 --data radius=200
      --user 5***6:2***3
      https://backend.sigfox.com/api/coverages/global/predictions
…
* Connection #0 to host backend.sigfox.com left intact
{"margins":[31,27,25]}
```

With python:

```
import requests
parameters = {"lat": 43.52, "lng": 1.55, "radius": 200}
login = "5***6"
password = "2***3"
authentication = (login, password)
response = requests.get("https://backend.sigfox.com/api/coverages/global/predictions",
                        auth=authentication,
                        params=parameters)
# The variable response contains the response from the server
```

With java:

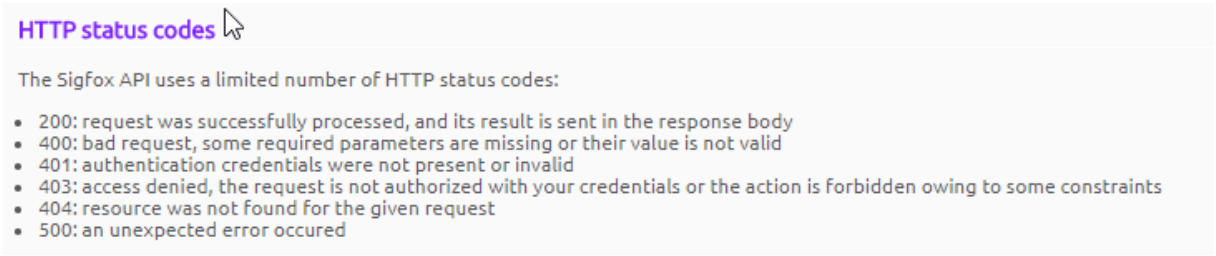
```
URIBuilder builder = new URIBuilder();
builder.setScheme("http").setHost("https://backend.sigfox.com")
                         .setPath("/api/coverages/global/predictions")
                         .setParameter("lat", 43.52)
                         .setParameter("lng", 1.55)
```

```
                        .setParameter("radius", 200);
URI uri = builder.build();
String user = "5***6";
String pwd = "2***3";
HttpGet httpget = new HttpGet(uri);
httpGet.addHeader("Authorization",
                  "Basic " + Base64.encodeToString(
                              (user + ":" + pwd).getBytes(),
                              Base64.NO_WRAP));
response = httpget.getURI();
// The variable response contains the response from the server
```

API Result handling and interpretation

The first check to be done once the request has been sent is the return code.
If the return code is not 20x, then there was an issue with the request. For example, if the radius of our example has been set to 10, then the return code would be 400. In that case, the text in the HTTP reply is helpful to understand the issue:

If another request targets for example a device that is out of the scope of the API scope, then an HTTP 403 return code will be sent. The API documentation will help to understand the result code, and the text associated to interpret the code.



If the result is a 20x, the request has been successful. Once the content has been retrieved, the result can then be interpreted. In our example, with 3 positives, non-zero values, we have 3 or more base stations covering our targeted surface of 200 meters radius.
For a different position return only 2 strictly positive values, then the targeted surface is only covered by two stations and, for another position that do not return positives values, then the targeted surface isn't covered.

Depending on the objective, the result can be:

- A binary, either covered or not position
- A result of redundancy at this specific position
- A useful information on the margin for this specific position
- A simulation of coverage knowing the attenuation due to environment (indoor: 20dB margin is usually considered) or device used (1U device has a up to 7dB penalty compared to 0U devices)
- A note built from all those considerations (reliability and resilience comes from margin and redundancy, moving objects will greater benefit from redundancy than static objects that will more benefit from margin…)

All those results interpretations can (and should) be done to present an information adapted to the user.

When API should **not** be used

APIs shouldn't be used to target event based data: they aren't efficient with pooling. The right way to deal with event based data is using callbacks. They can be used on conjunction with APIs (and should), and APIs might be used then to go further on processing this single event (for example, moving a device to verified device-type after receiving a specific message…).
APIs shouldn't be used to retrieve messages or status data, nor refresh user database periodically, especially if the pooling frequency is above once per day.

When API should be used

APIs should be used in any recurring task that have rare occurrence, for example:

- Perform a device move from one device type to another upon end customer subscription
- Disengage sequence number of a device after receiving event callback warning
- Retrieve missed callback after event callback send error or user's database maintenance
- Replacement of a failed device by a working one

Other data retrieval task might benefit from API calls:

- End user exposition of coverage on a specific address
- Monthly check of token duration associated with devices
- PAC retrieval prior to moving a device

Sample use case, mixing all interfaces

For example, if a user wants to build and keep its own copy of sigfox backend data, such as the groups and device type organization, callbacks definition, registered devices list and all the messages received, then synchronization between sigfox and user servers' database should use the following principles:

- Messages synchronization should be done through callbacks (push mechanism),

- status of devices should be synchronized through events (push mechanism),
- Group organization, device types, callbacks and settings (such as email, url…) through APIs

Note that the latest data, retrieved by API, are usually manipulated by a user. In such cases, the user server should provide its own GUI to manipulate groups, device type, callbacks and user. Thus, all interactions will be done on user's server instead of sigfox GUI, with perfect integration and without unnecessary pooling.
In this example, the only task that enroll sigfox GUI is the original API creation itself.

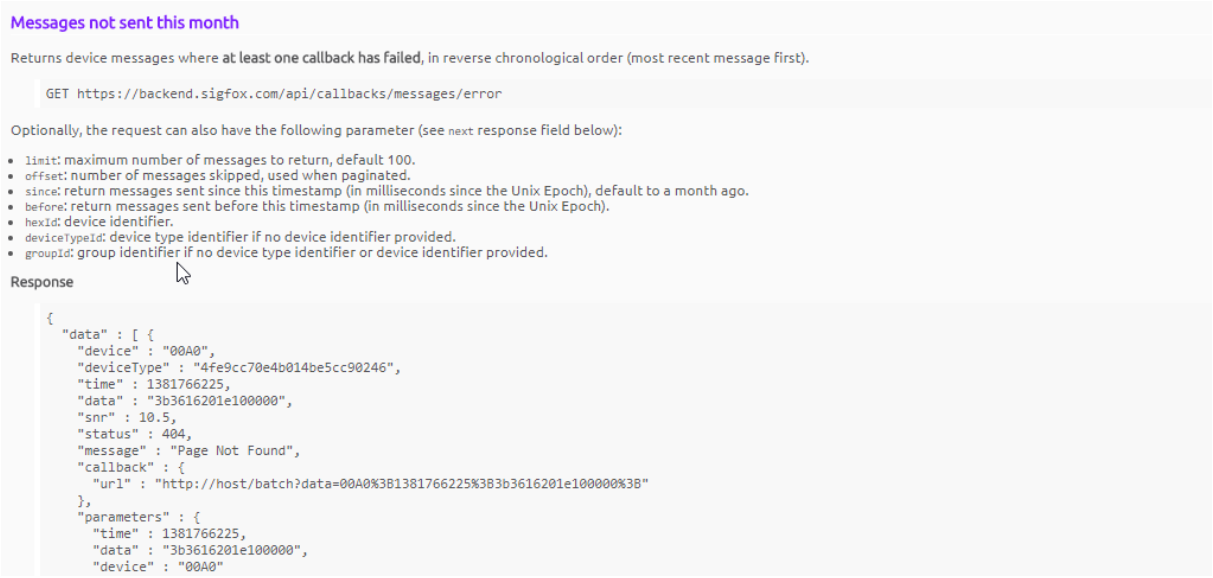
Sample use case involving APIs pooling

If at one point user wants to use the sigfox GUI for specific -rare- tasks but keep a synchronization with its own servers, then periodically or manually, a status refresh must be done. It will mostly use APIs queries to pull and report the changes made on the sigfox GUI to the user's servers. In such case, pooling through APIs is used, although it cannot be as efficient as directly using APIs to handle those rare tasks. When APIs pooling is required, the pooling frequency must match the probable usage. For example, if the only part relying on sigfox GUI is the group organization, and if the occurrence of group creation / update / deletion is one per month, pooling automatically changes through API should not be done on a minute base: less than 1/40000 requests will likely show a change. Either a month based timing or a manual triggering should be preferred.

APIs that are worth a look

Missed messages

One of the main usage of sigfox is to get data from devices using callbacks. But what's happens when a callback is missed? How to get information back to the server handling callbacks?



Messages not sent this month

Returns device messages where **at least one callback has failed**, in reverse chronological order (most recent message first).

```
GET https://backend.sigfox.com/api/callbacks/messages/error
```

Optionally, the request can also have the following parameter (see `next` response field below):

- `limit`: maximum number of messages to return, default 100.
- `offset`: number of messages skipped, used when paginated.
- `since`: return messages sent since this timestamp (in milliseconds since the Unix Epoch), default to a month ago.
- `before`: return messages sent before this timestamp (in milliseconds since the Unix Epoch).
- `hexId`: device identifier.
- `deviceTypeId`: device type identifier if no device identifier provided.
- `groupId`: group identifier if no device type identifier or device identifier provided.

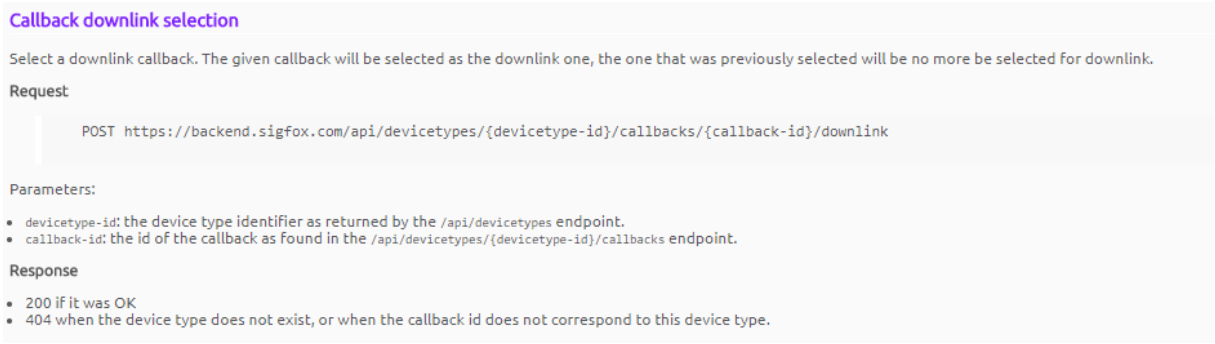
Response


```
{
  "data" : [ {
    "device" : "00A0",
    "deviceType" : "4fe9cc70e4b014be5cc90246",
    "time" : 1381766225,
    "data" : "3b3616201e100000",
    "snr" : 10.5,
    "status" : 404,
    "message" : "Page Not Found",
    "callback" : {
      "url" : "http://host/batch?data=00A0%3B1381766225%3B3b3616201e100000%3B"
    },
    "parameters" : {
      "time" : 1381766225,
      "data" : "3b3616201e100000",
      "device" : "00A0"
```

After a server downtime or after receiving an event callback for missed callback, you should use the API addressing missed messages:

Downlink selection

When bidirectional mode is used with messages, you might want to change the downlink URL to redirect the requests to another server while working on a planned maintenance on the main server. The downlink selection can help to define the right URL to get downlink answer from:



Callback downlink selection

Select a downlink callback. The given callback will be selected as the downlink one, the one that was previously selected will be no more be selected for downlink.

Request

```
POST https://backend.sigfox.com/api/devicetypes/{devicetype-id}/callbacks/{callback-id}/downlink
```

Parameters:

- `devicetype-id`: the device type identifier as returned by the `/api/devicetypes` endpoint.
- `callback-id`: the id of the callback as found in the `/api/devicetypes/{devicetype-id}/callbacks` endpoint.

Response

- 200 if it was OK
- 404 when the device type does not exist, or when the callback id does not correspond to this device type.