Tedee API documentation

Oct 21, 2021

Overview

CHAPTER 1

Getting started

Tedee's API exposes resources that enable you to work with your devices. By calling relevant endpoint user is able, among others, to manipulate lock, get battery level or read it's activities. This guide aims to help you to get started with Tedee's API.

1.1 What you need?

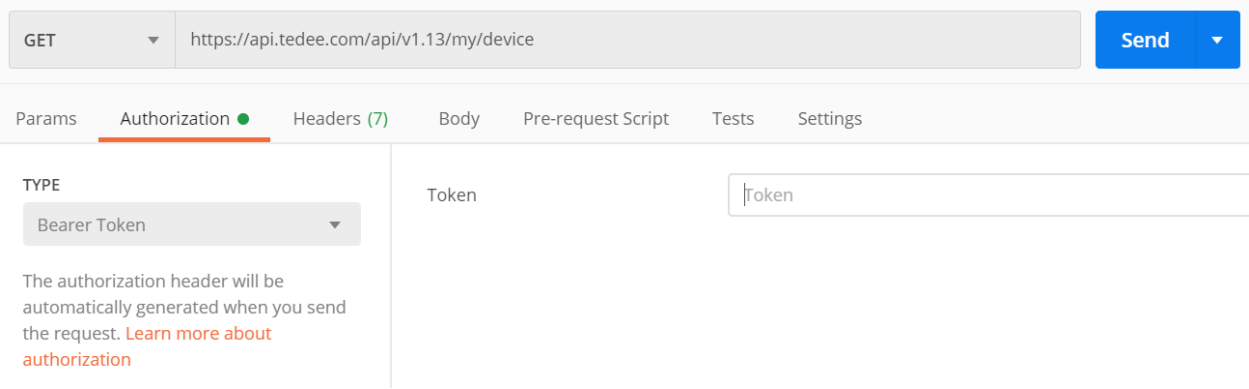
Starting working with the API doesn't require much prerequisites. You'll need:

- REST API client - Postman is a great example here

1.2 Registration and authentication

Before you can use the API you must create an account first. This can be done using the mobile tedee app or using the registration page. Every request requires authentication token. The process of authentication is described in dedicated *section*, however for now we'll just need to get JWT (described in mentioned section). Once we've got our access token we can use Postman to make authenticated request.

To do this open Postman and go to Authorization tab.



In `TYPE` dropdown select **Bearer Token** and in `Token` input field put your access token. From now on, our requests should be authenticated.

1.3 REST API request

To interact with the Tedee REST API, you send HTTP requests that use a supported method: GET, POST, PATCH, or DELETE. POST and PATCH request bodies and server responses are sent in JSON payloads.

The path URL resource names and query parameters are case insensitive. However, values you assign, entity IDs, and other base64 encoded values are case sensitive.

1.3.1 Request message URI

All Tedee REST API requests use the following URL format:

`https://api.tedee.com/{version}/{resource}`

- **version** - we use *API versioning* to deliver new functionalities more easily, keeping backwards compatibility
- **resource** - path to the resource you want to manipulate

1.3.2 Request message headers

Requests require additional meta data sent in headers, which help to process them correctly:

- **Authorization** - contains JWT to authorize the request (*read more*)
- **Content-Type** - `application/json` value is required in this header for POST, PUT and PATCH requests

1.3.3 User context

Tedee API is based on REST architecture. This implies that the application does not store any state. Hence, the client session can not be handled on the server side and every request should provide the information about the user. This is handled by JWT included in *authorization* header. The access token contains an information that allows to identify the user. All the requests that starts with `/my/` referes to resources directly assigned to the current user for example asking for `/my/devices` will return only devices to which authenticated user has access.

1.3.4 Example request

Let's get some information about our devices now. Put this address `https://api.tedee.com/api/v1.22/my/device` in the *url* input like in the *screen above* and click **Send**.

```
GET https://api.tedee.com/api/v1.22/my/device HTTP/1.1
Accept: application/json
Authorization: Bearer <<your-jwt>>
```

You should receive response with all your devices.

1.4 REST API Response

1.4.1 Response HTTP code

Each response contains an HTTP code that informs about the status of the request processing. Tedee API uses standard HTTP status codes.

1.4.2 Response message body

Each endpoint returns data in the same format:

```
{
  "result": object,
  "success": boolean,
  "errorMessages": array,
  "statusCode": number
}
```

- **result** - represents actual data that user requests for,
- **success** - describes weather the request has been processed successfully or not,
- **errorMessages** - is an array of eventual errors that occured while processing the request,
- **statusCode** - represents Http status code of the response.

1.4.3 Response message headers

Here's a list of most important headers returned in Tedee API responses:

- **Content-Length** - size of the response body
- **Content-Type** - indicates the media type of the resource, `application/json` in most cases
- **X-Correlation-ID** - corralates subsequent requests
- **Date** - includes date and time when the messages was sent
- **API-Supported-Versions** - lists all available API versions for that endpoint

1.4.4 Example response

Below is an example response for the battery level request:

- HTTP status code - `200`
- Response body:

```
{
    "result": {
        "level": 75,
        "date": "2020-04-01T11:31:54.969"
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

- Response headers:

```
api-supported-versions: 1.9, 1.10, 1.11, 1.12, 1.13
content-encoding: gzip
content-length: 220
content-type: application/json; charset=utf-8
status: 200
x-correlation-id: 800003f6-0400-1600-d63f-84710c7967bb
Date: Wed, 01 Apr 2020 14:17:21 GMT
```

1.5 Code samples

Tedee API documentation also provides code samples which present practical usage examples. Currently you can find there samples written in C#.

Note: Before you run any sample, provide user name and password in the **appsettings.json** file to authenticate.

1.6 What's next?

Here's a list of example actions that you can do using the API:

- Get device details
- Update device settings
- Calibrate lock
- Read lock state
- Read device activities
- Lock, unlock or pull spring

CHAPTER 2

API versioning

Tedee API is versioned, which allows us to develop and provide new functionalities while keeping backwards compatibility. This ensures that even when a breaking change happens, other services will still be able to work with the API the same way they did.

2.1 Usage

Each request requires the version number to be provided in the url using this format.
`https://api.tedee.com/{version}/{resource}`
For example:
`https://api.tedee.com/v1.22/my/device`

2.2 Version supported features

To verify which versions provide the feature that you want to utilize, we recommend to visit the API Swagger documentation. You should be able to select a version in top right corner and check if the desired endpoint is available and how the request body should look like.

Note: We recommend to upgrade to the latest available version of the API as soon as possible. We do not guarante how long older versions will be available.

CHAPTER 3

Logo guideliness

If you want to use the tedee logo in your applicaton / plugin / extension, it must meet some requirements.

3.1 Logo variations

You can use tedee logo in two different variations: full logo - contain sign with full tedee symbol, second option is only a ‘t’ symbol on blue background.

Download tedee full logo
Download tedee sign

Logo

Tedee sign

3.2 Safe areas

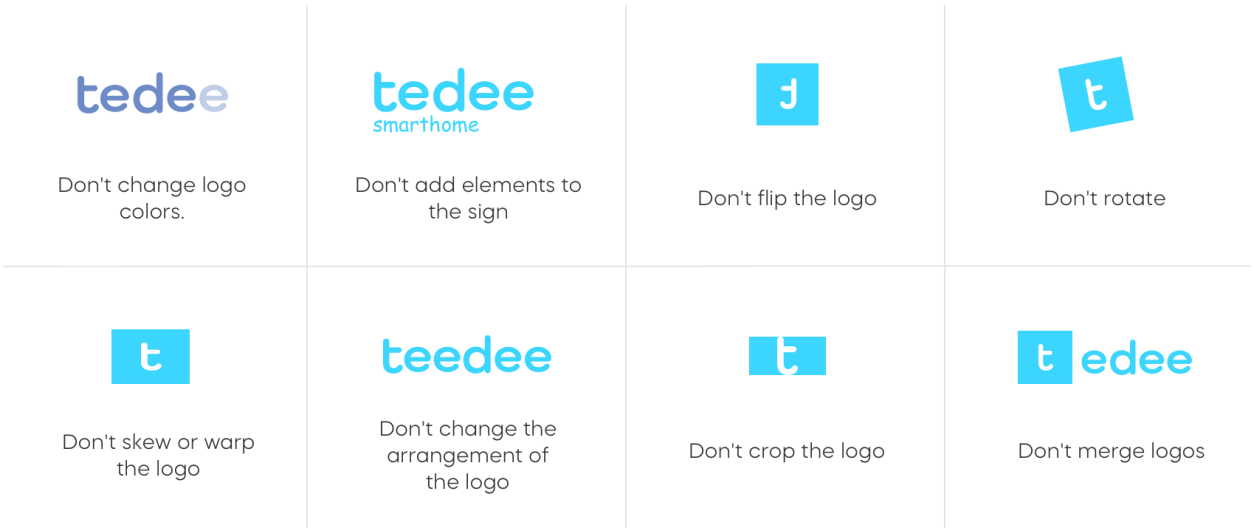
The protective field defines the minimal area around the logo, where no other text or graphic may appear. The protective field is based on the height of the sign.

3.3 Main colour

The main brand colour is shade of blue.

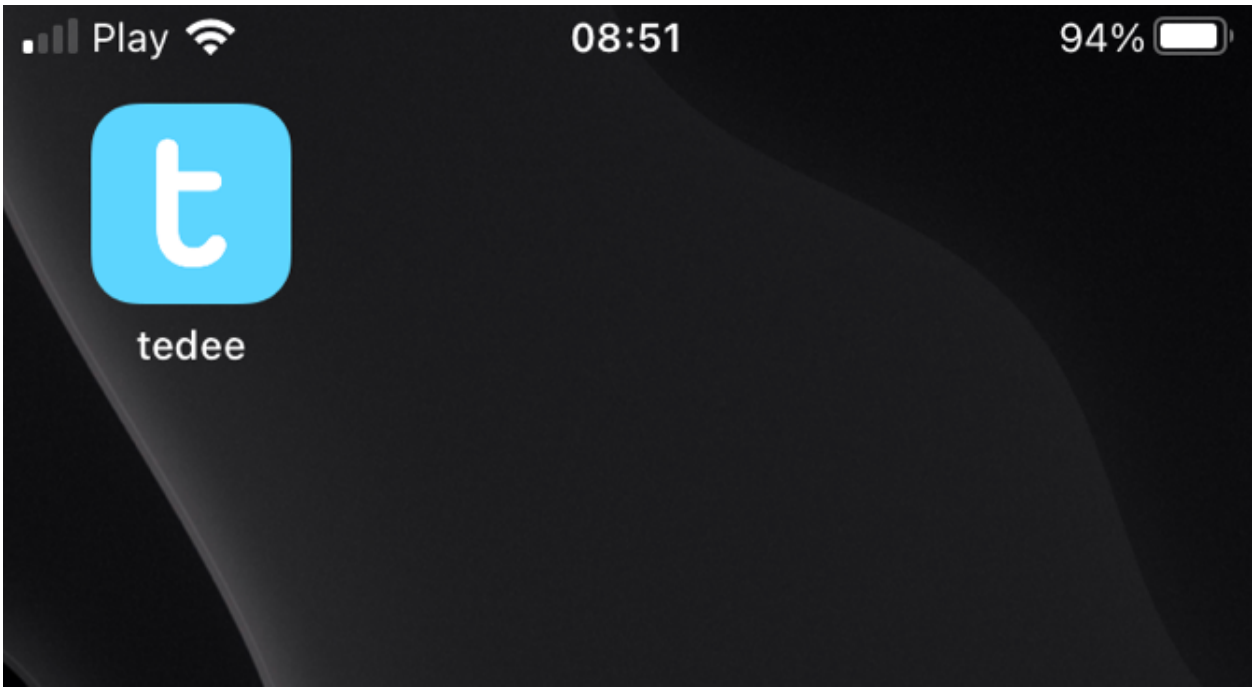
3.4 List of don’ts

Do not modify the logos in any way, other than resizing. If you need to resize, preserve the ratio, and ensure the badge is legible and fully visible.



3.5 Naming

You can only use 'tedee' word, as a name. Do not attempt to add some words to it, like 'tedee smarthome', 'tedee-lock'. For example, find out screen with tedee application for iOS.



CHAPTER 4

Release notes

2021-05-24:

- added a new API version 1.19
- added timezone for lock and bridge devices
- removed “Put” endpoint and added “Patch” endpoint to bridge update

2021-05-10:

- added a new API version 1.18
- removed “unregister” endpoint from StandardNotificationHub

2021-04-26:

- added a new API version 1.17
- added endpoint to check close/open/pull operation status
- changed response for close/open/pull operation request (added “lastStateChangedDate” and changed result code to 202 Accepted)
- added “lastStateChangedDate” and “stateChangeResult” to all endpoints returning Lock state

2021-01-18:

- added a new API version 1.16
- all user settings moved to a new “userSettings” property in the Lock type
- removed “bridgeId” property from the Lock type
- removed “connectedToId”, “iotDeviceName” and “voipNumber” properties from Bridge type

2020-12-07:

- removed API versions 1.10, 1.11 and 1.12
- added information about an ongoing bridge software update
- lock state returned as an enum

2020-11-23:

- scopes applied to all API endpoints
- added new endpoints for lock synchronization
- API versions in swagger sorted in descending order

CHAPTER 5

How to begin integration

All the endpoints exposed on this API require authentication. You can find a guide to achieve that on *How to authenticate* page.

5.1 Client id for you application

If you want to use Code Flow or Implicit Flow in the authorization process, you will need a **client id** issued for your application by Tedee. For that please send the request for obtaining **client id** using this form.

Based on above information, we will register your application in our system and send you **client id** and **client secret**.

5.2 Tutorials

To learn step by step how to integrate with tedee, please follow these tutorials:

5.2.1 Integration via API

These tutorials are intended for integrators who will use only Tedee API for integrations:

- *How to authenticate*
- *Get and sync locks*
- *Operate locks*
- *How to manage device shares*
- *Update locks settings*

5.2.2 Integration via BLE

These tutorials are intended for integrators who wants to integrate with Tedee devices via Bluetooth connection:

- *How to connect to Tedee device via Bluetooth*

5.3 More information

If you need more information about integration please check following links:

- You can user our logo but first read the *logo guidance*
- Check existing *integration examples*
- Use community forum if you need help

CHAPTER 6

How to authenticate

Each request of this API requires authentication. We utilizes OAuth 2.0 or Personal Access Key to identify the user.

Authentication type	Description
Personal Access Key	User can create his own keys with different scopes and expiration dates and then use them to authenticate his requests
Oauth 2.0	Standard OAuth 2.0 type of authentication based on tokens. We only support code flow.

6.1 Personal Access Key

To authenticate via personal access key (PAK) first you need to generate it on your account. To do this you need to send request to *Create Personal Access Key* endpoint.

Sample response

```
{
    "result": {
        "id": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
        "key": "smnxaz.IWA6u00VLQmA8tlfioDXcH+bSiI6u8LgTG9cv3Evh/E"
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

The PAK is in result.key.

> **Warning:** Keep your key secure!

Note: You can see the full personal access key just once in the response.

After creating a PAK you can use it to authenticate to endpoints that you gave permissions (by defining proper scopes). To use this type of authentication use schema PersonalKey in the Authorization header.

Sample request to get lock using PersonalKey schema

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1" -H "accept: application/json"␣
↪-H "Authorization: PersonalKey <<personal key>>"
```

6.2 OAuth 2.0

We support OAuth 2.0 code flow authorization to get the access token:

Flow name	When to use
Code Flow	When you can store refresh tokens and periodically exchange them for access tokens. One time interaction with the user is needed to obtain the refresh token. Examples: mobile apps, service apps

Warning: Please remember to protect the access token and store it in a secure place. If someone else can capture your JWT, they can pretend to be you and invoke some actions in your behalf.

6.2.1 Code Flow

Warning: Code flow should not be used in public facing application (for example service apps) only service to service. If you intend to use code flow with public facing application please consider using proof key for code exchange.

This flow should be used for applications that can store refresh tokens and periodically exchange them for access tokens after they expire. One time interaction with the user is needed to obtain the refresh token. Next, the refresh token can be used to automatically obtain the next refresh tokens and access tokens. Access token is valid for 4 hours. Refresh token is valid for 14 days.

Note: To receive the JWT using Code Flow you will need a **client id** and **client secret** issued for your application by Tedee. You can find a guide to achieve that on How to begin integration page.

There are three steps to get the JWT using Code Flow:

1) Get an authorization code

The authorization process begins with the GET request to the authorization endpoint. This is the interactive part of the flow, where the user takes action.

```
GET https://tedee.b2clogin.com/tedee.onmicrosoft.com/B2C_1A_Signup_Signin_With_Kmsi/
↪oauth2/v2.0/authorize
?response_type=code
&client_id={client_id}
&redirect_uri={redirect_uri}
&response_mode={response_mode}
&scope={scope}
&state={state}
```

- **client_id** - The client id assigned to your application.
- **redirect_uri** - The redirect URI of your application, where authentication responses are sent and received by your application.
- **response_mode** - The method that you use to send the resulting authorization code back to your application. It can be **query**, **form_post**, or **fragment**. You need to choose the one that is compatible with your application.
- **scope** - A space-separated list of scopes. A single scope value indicates the permissions that are being requested. The **offline_access** scope indicates that your app needs a refresh token for long-lived access to resources. The "https://tedee.onmicrosoft.com/api/user_impersonation" scope is required (*list of available scopes*).
- **state** - A value included in the request that can be a string of any content that you want to use. Usually, a randomly generated unique value is used, to prevent cross-site request forgery attacks.

Example

```
GET https://tedee.b2clogin.com/tedee.onmicrosoft.com/B2C_1A_Signup_Signin_With_Kmsi/
↪oauth2/v2.0/authorize
?response_type=code
&client_id=bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996
&redirect_uri=https://yoursite.com/auth
&response_mode=query
&scope=https://tedee.onmicrosoft.com/api/user_impersonation%20https://tedee.
↪onmicrosoft.com/api/Lock.Operate
&state=d917d40e-0b1a-4495-8e23-e449c916a532
```

After the user sign-in, the authorization code will be sent to your application to the address specified in the **redirect_uri** parameter (using the method specified in the **response_mode** parameter).

A successful response that uses response_mode=query looks like this:

```
GET {redirect_uri}
?code={code}
&state={state}
```

- **redirect_uri** - The redirect URI of your application.
- **code** - The authorization code that the application requested.
- **state** - If a state parameter is included in the request, the same value should appear in the response. The application should verify that the state values in the request and response are identical.

2) Get a token

After successfully receiving the authorization code, you can use it to request an access token by sending a POST request to the token endpoint.

```
POST https://tedee.b2clogin.com/tedee.onmicrosoft.com/B2C_1A_Signup_Signin_With_Kmsi/
↪oauth2/v2.0/token
Content-Type: application/x-www-form-urlencoded
```

(continues on next page)

(continued from previous page)

```
grant_type=authorization_code
&client_id={client_id}
&client_secret={client_secret}
&scope={scope}
&code={code}
&redirect_uri={redirect_uri}
```

- **client_id** - The client id assigned to your application.
- **client_secret** - The application client secret.
- **scope** - A space-separated list of scopes. A single scope value indicates the permissions that are being requested. The **offline_access** scope indicates that your app needs a refresh token for long-lived access to resources. The "https://tedee.onmicrosoft.com/api/user_impersonation" scope is required (*list of available scopes*).
- **code** - The authorization code that you acquired in the first step of the flow.
- **redirect_uri** - The redirect URI of the application where you received the authorization code.

Example

```
POST https://tedee.b2clogin.com/tedee.onmicrosoft.com/B2C_1A_Signup_Signin_With_Kmsi/
↪oauth2/v2.0/token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&client_id=bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996
&client_secret=81A2Bde1ZsZeEPDJLASKq1sBsuKaNa11W+3biasTkLAC=
&scope=https://tedee.onmicrosoft.com/api/user_impersonation%20https://tedee.
↪onmicrosoft.com/api/Lock.Operate
&code=AwABAAAAvPM1KaPlrEqdFSBzjqfTGBCmLdgfSTLEMPGYuNHSUYBrq
&redirect_uri=https://yoursite.com/auth
```

A successful token response looks like this:

```
{
    "not_before": "1442340812",
    "token_type": "Bearer",
    "access_token":
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
    "expires_in": "3600",
    "refresh_token": "AwABAAAAvPM1KaPlrEqdFSBzjqfTGAMxZGUTdM0t4B4...",
    "refresh_token_expires_in": 1209600
}
```

- **not_before** - The time at which the token is considered valid, in epoch time.
- **token_type** - The token type value (Bearer).
- **access_token** - The signed JSON Web Token (JWT) that you requested.
- **expires_in** - The length of time that the access token is valid (in seconds).
- **refresh_token** - An OAuth 2.0 refresh token. The app can use this token to acquire additional tokens after the current token expires.
- **refresh_token_expires_in** - The length of time that the refresh token is valid (in seconds).

The value of the `access_token` property is your **JWT** that should be used to *authenticate your calls* to the API.

3) Refresh the token

Access tokens are short-lived. After they expire, you must refresh them to continue to access resources. To do this, submit another POST request to the token endpoint. This time, set **grant_type=refresh_token** and provide the refresh token instead of the authorization code.

```
POST https://tedee.b2clogin.com/tedee.onmicrosoft.com/B2C_1A_Signup_Signin_With_Kmsi/
↪oauth2/v2.0/token
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&client_id={client_id}
&client_secret={client_secret}
&scope={scope}
&refresh_token={refresh_token}
&redirect_uri={redirect_uri}
```

6.3 Attach JWT to the request

Now, since we have our JWT, we can use it to authenticate our calls. To achieve that, we just have to add an `Authorization` header containing our access token. This header value should look like `Bearer <<access_token>>`, where **<<access_token>>** is our JWT.

Let's see it on the below examples where we want to get information about all our devices:

```
curl -H "Authorization: Bearer <<access_token>>" https://api.tedee.com/api/v1.22/my/
↪device
```

6.4 JWT token details

JSON Web Token (JWT) is open standard of securely transmitting information between parties. Anyone who has access to the token is able to decode it and read the information.

6.4.1 Claims

The JWT contains useful information which you can use and the table below describe the most important one:

Claim name	Description
exp	Presents the expiration time on and after which the JWT will not be processed.
email	Contains user's email address provided during registration process.
name	Contains user's name provided during registration process.
oid	User's unique identifier assigned during registration process.

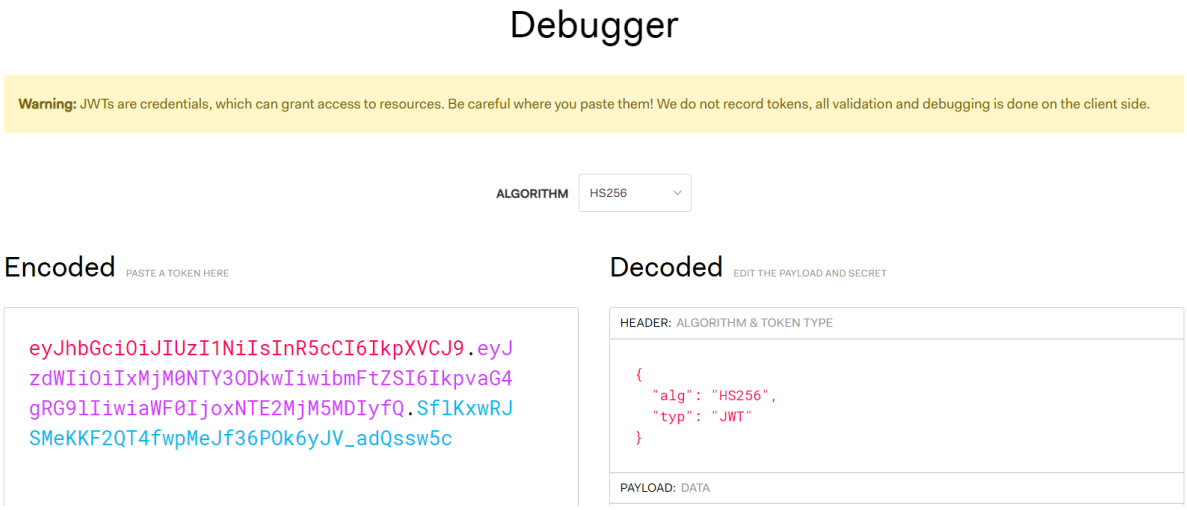
You can read more about claims here.

6.4.2 Expiration date

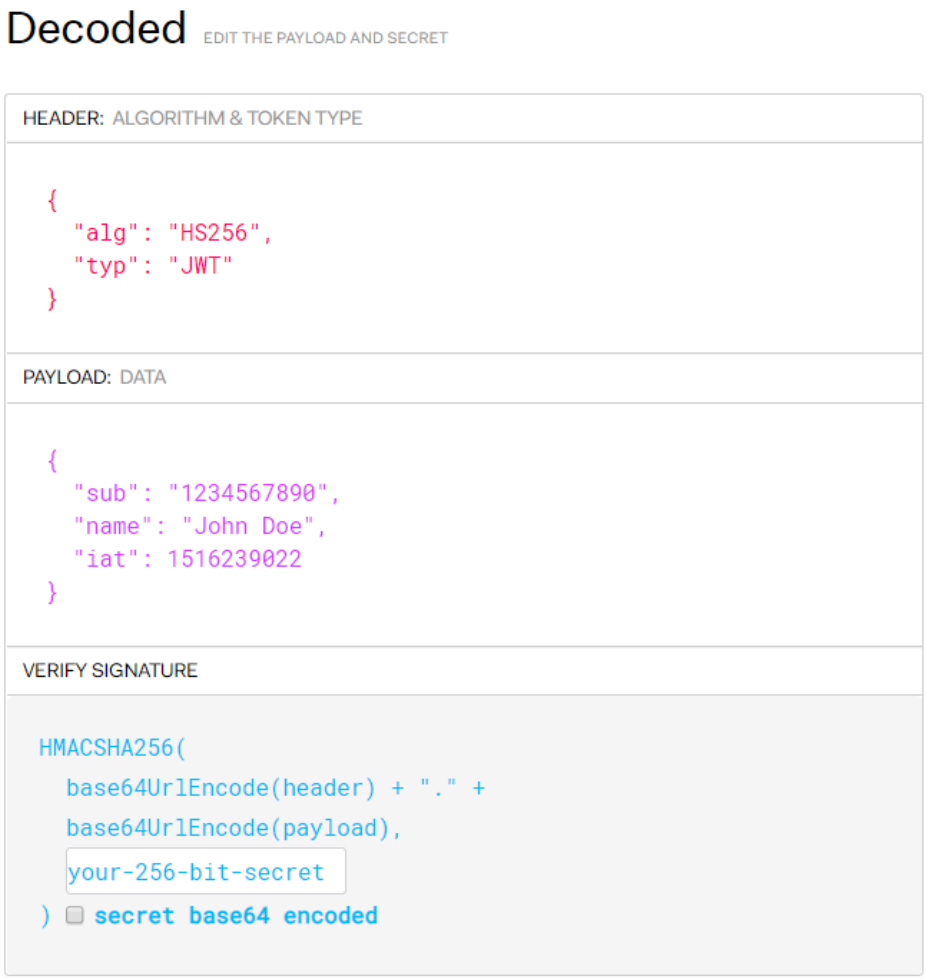
Tedee API tokens are valid for 4 hours since the creation time.

6.4.3 Debugger

https://jwt.io provides a very usefull online tool to work with JWT tokens. You can use it to decode and read data included in JWT. To do that go to JWT debugger and fill in the **Encoded** input field with your token.



You should see the decoded data right away on the right side of the screen



6.5 Scopes

Scopes define the set of permissions that the application requests. Below is a list of available scopes that can be requested during the authorization process (a single scope value indicates the permissions that are being requested).

Scope	Operation	Description
https://tedee.onmicrosoft.com/api/user_impersonation	Impersonate user	Access this app on behalf of the signed-in user.
https://tedee.onmicrosoft.com/api/Account.Read	View user account	Grants the ability to view user information.
https://tedee.onmicrosoft.com/api/Account.ReadWrite	View and edit user account	Grants the ability to view and edit user information. Also grant the ability to delete user account.
https://tedee.onmicrosoft.com/api/Device.Read	View devices	Grants the ability to view all devices and query information for specific device.
https://tedee.onmicrosoft.com/api/Device.ReadWrite	View and edit devices	
https://tedee.onmicrosoft.com/api/DeviceShare.Read	View device shares	Grants the ability to view shares for all devices or for specific device.
https://tedee.onmicrosoft.com/api/DeviceShare.ReadWrite	View and edit device shares	Grants the ability to view shares for all devices or for specific device. Also grants the ability to update or delete existing share or create new one.
https://tedee.onmicrosoft.com/api/DeviceActivity.Read	View activity logs	Grants the ability to query activity logs.
https://tedee.onmicrosoft.com/api/Bridge.Operate	Operate bridges	Grants the ability to pair and unpair locks with bridges.
https://tedee.onmicrosoft.com/api/Lock.Operate	Operate locks	Grants the ability to lock, unlock and perform pull spring. Also grants the ability to perform lock calibration.
https://tedee.onmicrosoft.com/api/Mobile.Read	View mobile devices	Grants the ability to view user registered mobiles.
https://tedee.onmicrosoft.com/api/Mobile.ReadWrite	View and edit mobile devices	Grants the ability to manage user mobile or other devices.
https://tedee.onmicrosoft.com/api/DeviceCertificate.Operate	View devices certificates	Grants user possibility to access devices certificates.

CHAPTER 7

How to get and sync locks

For this tutorial let's consider that you have one or more locks and you need to get their data and then sync their states periodically.

7.1 Get user locks

First thing that you need to do is to use endpoint *Get all locks*. This endpoint will return full data of all currently logged user locks.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json" -
↪H "Authorization: Bearer <<access token>>"
```

7.2 Sync user locks

Then after you have succesfully downloaded locks data you can use endpoint *Sync locks* to periodically refresh current state of user locks like battery level, connection state, and lock position.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/sync" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response

```
{
    "result": [
        {
            "id": 1,
            "isConnected": true,
```

(continues on next page)

(continued from previous page)


```
            "lockProperties": {
                "state": 3,
                "isCharging": false,
                "batteryLevel": 54,
                "stateChangeResult": 0,
                "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
            }
        },
        {
            "id": 2,
            "isConnected": true,
            "lockProperties": {
                "state": 2,
                "isCharging": true,
                "batteryLevel": 80,
                "stateChangeResult": 0,
                "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
            }
        }
    ],
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Warning: You shouldn't run sync endpoint more than once every 10 seconds.

Note: Do not hardcode lock Id. It will change everytime user add the lock to account. If you must hardcode then use device Serial Number which will not change.

CHAPTER 8

How to operate your locks

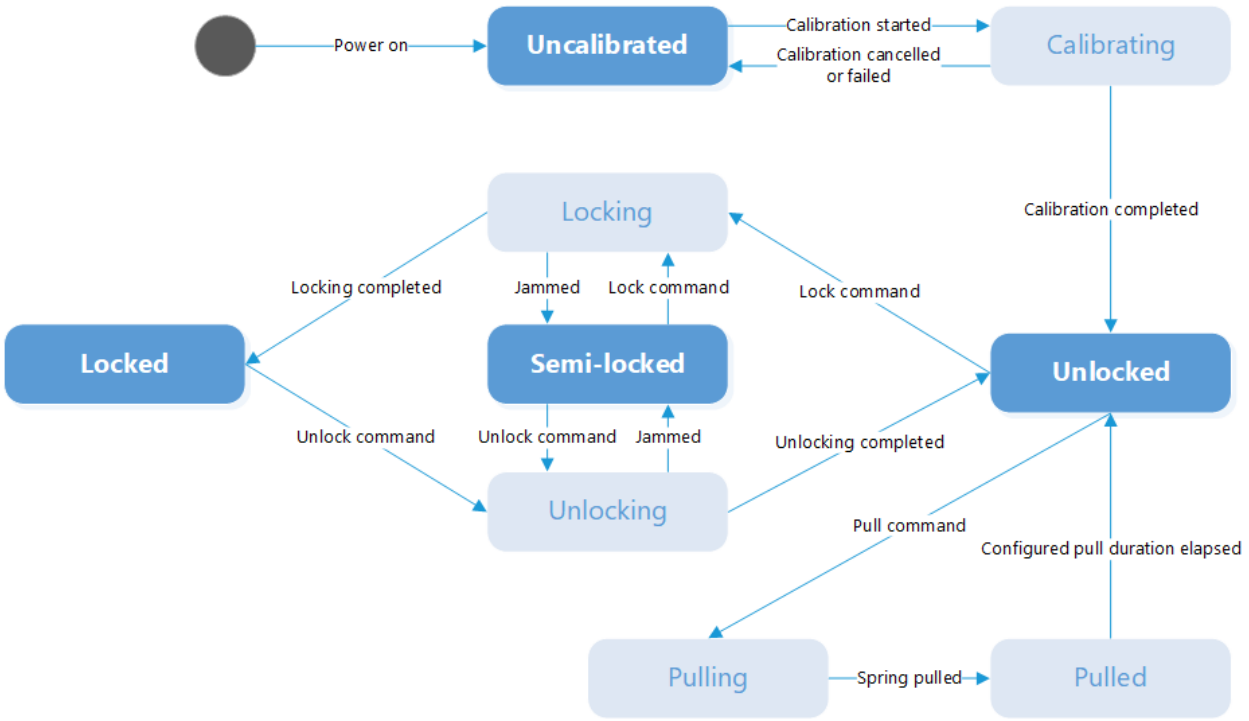
In this section let's focus on how to operate tedee locks. To operate lock via API you need to be owner of the device or have active share with remoteAccessDisabled flag set on false (see *Share Details*). What you need is id of your lock.

Note: You should calibrate your lock before using these endpoints. If you didn't calibrate your lock these endpoints will return successful response but nothing will happen.

You can perform following actions on lock:

- Lock
- Unlock
- Pull

Each action can be performed only in specific lock states. Here is Lock state diagram:



Make sure lock and bridge are connected.

8.1 Lock tedee lock

To lock the device first make sure it is in unlocked or semi-locked state then send *lock command*.

8.1.1 Lock request

We will send lock command for device with id = 1.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/lock" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

In response you will receive operationId and lastStateChangedDate. The locking operation usually takes up to 3 seconds.

8.2 Unlock tedee lock

To unlock the device first make sure it is in locked or semi-locked state then send *unlock command*.

8.2.1 Unlock request

We will send unlock command for device with id = 1.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/unlock" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

In response you will receive operationId and lastStateChangedDate. The unlocking operation usually takes up to 3 seconds. When lock has auto pull spring enabled then unlocking lock will perform pull operation.

Note: When lock has auto pull spring enabled it will also perform pull spring within unlock command. Optional parameter in the request allows to unlock the lock without pulling the spring.

Note: Optional parameter in the request allows to perform pull spring when lock is in unlocked state.

8.3 Pull spring in tedee lock

To perform pull spring first make sure lock is in unlocked state then use *pull spring command*.

Note: When lock has auto pull spring enabled it will also perform pull spring within unlock command. You shouldn't send additional pull spring command then.

8.3.1 Pull request

Example request will perform pull spring on the lock with id = 1.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/pull" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

In response you will receive operationId and lastStateChangedDate. The duration of pull spring is configured by user.

Note: To perform pull spring you can also use Unlock request with optional parameter.

Note: Additionally you should calibrate pull spring in your lock before using this endpoint. If you didn't calibrate pull spring this endpoint will return successful response but nothing will happen.

8.4 Checking operation progress

The lock/unlock/pull actions will take few seconds so you must somehow check the progress. To do that first call the *Get device operation endpoint* with the operationId you received when you called lock/unlock/pull endpoint. To ensure that operation is completed check fields "status" and "result", if the operation was succesfull the first field should have value "COMPLETED" and the second one "0". After that you can simply call the *Sync single endpoint* to get new lock status.

8.4.1 Get device operation

Example of getting operation status

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/device/operation/1619078520230" -H
↪"accept: application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "deviceId" : 1,
        "operationId" : "1619078520230",
        "result" : 0,
        "status" : "COMPLETED"
        "type" : 1
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

8.4.2 Sample sync single request

Example of syncing single lock with id = 1.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/sync" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "isConnected": true,
        "lockProperties": {
            "state": 3,
            "isCharging": false,
            "batteryLevel": 54,
            "stateChangeResult": 0,
            "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
        }
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

CHAPTER 9

How to update lock settings

In this section let's focus on how to update tedee lock settings. If you want to update lock settings, the first thing you need to know is that there are two kinds of lock settings:

- **device settings** - settings on the device side. That means changing settings will affect all device users that have access to that lock
- **user settings** - each user has his/her own settings and they are responsible for auto unlock feature

To update device settings firstly use endpoint *Get single lock*:

Sample request

This request will get data for lock with Id = 1.

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1" -H "accept: application/json"␣
↪-H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "deviceSettings": {
            "autoLockEnabled": true,
            "autoLockDelay": 10,
            "autoLockImplicitEnabled": false,
            "autoLockImplicitDelay": 10,
            "pullSpringEnabled": true,
            "pullSpringDuration": 10,
            "autoPullSpringEnabled": false,
            "postponedLockEnabled": false,
            "postponedLockDelay": 10,
            "buttonLockEnabled": false,
            "buttonUnlockEnabled": false
        },
```

(continues on next page)

(continued from previous page)


```
        "userSettings": {
            "autoUnlockEnabled": true,
            "autoUnlockConfirmEnabled": true,
            "autoUnlockRangeIn": 300,
            "autoUnlockRangeOut": 400,
            "autoUnlockTimeout": 20,
            "location": {
                "latitude": 52.24070739746092,
                "longitude": 21.086990356445305
        },
        "lockProperties": {
            "state": 3,
            "isCharging": false,
            "batteryLevel": 18,
            "stateChangeResult": 0,
            "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
        },
        "id": 1,
        "connectedToId": 2,
        "serialNumber": "111111-11111",
        "macAddress": "00:0A:95:9D:68:16",
        "name": "Room 6",
        "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
        "type": 2,
        "created": "2020-01-01T00:00:00",
        "revision": 2,
        "deviceRevision": 2,
        "targetDeviceRevision": 2,
        "isConnected": true,
        "accessLevel": 2,
        "shareDetails": null,
        "softwareVersions": [
            {
                "softwareType": 0,
                "version": "1.0.0",
                "updateAvailable": true
            }
        ]
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Above endpoint returns data for the selected lock. The data includes the **revision** attribute. This is the version of the current lock settings and you must provide this value in the next update request.

After successfully retrieving the revision of the current lock settings, you can use endpoint *Update lock* to update the lock.

9.1 Update device settings

Firstly let's focus on updating device settings. To do that you need to specify which settings from *Device settings* you want to update.

Sample request

Example shows how to update settings that enable auto lock feature and set delay to 10 seconds for the device with id = 1. Also we update name of the lock.

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 2,
    "name": "Front door lock",
    "deviceSettings": {
        "autoLockEnabled": true,
        "autoLockDelay": 10
    }
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 3,
        "targetDeviceRevision": 3
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

The revision value in the update request must be the same as the current value in the system, otherwise the request will be refused with 409 (Conflict) error. If update will success you will receive the new revison value.

The targetDeviceRevision value is responsible for checking if device settings are up to date. Process of the updating device settings is described below:

1. Device receives settings with targetDeviceRevision.
2. If targetDeviceRevision is greater than revision on the device, device will update its settings.
3. Device sends confirmation about successful settings update.

Note: It is possible that revision and targetDeviceRevision values are not equal. It means that there have been more updates not related to device settings than to the device settings themselves.

All parameters in this endpoint (except id and revision) are optional. This means that specifying a given parameter will update its value. If a given parameter is not specified, its value will not change.

Only the owner or admin can update device settings and name. Guest can only modify user settings and location.

9.2 Update user settings

Let's focus now how to update user settings for the lock. Each user can have different set of settings. You need to specify which settings from *User settings* you want to update.

Sample request

Sample request will update auto unlock settings with location for the device with id = 1.

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 2,
    "userSettings": {
            "autoUnlockEnabled": true,
            "autoUnlockConfirmEnabled": true,
            "autoUnlockRangeIn": 300,
            "autoUnlockRangeOut": 400,
            "autoUnlockTimeout": 20,
            "location": {
                "latitude": 52.24070739746092,
                "longitude": 21.086990356445305
    }
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 3,
        "targetDeviceRevision": 3
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

CHAPTER 10

How to manage device shares

Once user have his device set, he will probably want to share it with other users (e.g. family or friends). To do this he need to create a share. In this tutorial we will walk through the process of creating, updating, deleting and listing share for user device.

10.1 Get current share list

If you want to get current shares for device, you will need deviceId and use *Get all shares*. This endpoint will return all shares for device.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/deviceShare?deviceId=1" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

10.1.1 Permanent access type

If you want user to have permanent access to the device you need send empty *repeatEvent* object.

Sample repeat event object

```
"repeatEvent": {
    "weekDays": null,
    "dayStartTime": null,
    "dayEndTime": null,
    "startDate": null,
    "endDate": null
},
```

10.1.2 Time restricted access type

If you want to restrict user access to the device you can send fields “startDate” or “endDate”, it will mark the period when share for user will be active. You can also restrict access to specific hours of the day by sending “dayStartTime” and “dayEndTime”. User can further customize this by selecting days. To send it proper way you need to use *Week days* enum.

Sample repeat event objects

In this case the share will be created from 14 december to 31 december.

```
"repeatEvent": {
    "weekDays": null,
    "dayStartTime": null,
    "dayEndTime": null,
    "startDate": "2020-12-14T08:09:57.781Z",
    "endDate": "2020-12-31T08:10:57.781Z"
},
```

In this case the share will be created from 1 december to 31 december, and user will have access only on friday and saturday between 15:00 and 18:00

```
"repeatEvent": {
    "weekDays": 48,
    "dayStartTime": "2020-12-01T15:00:00.000Z",
    "dayEndTime": "2020-12-31T18:00:00.000Z",
    "startDate": "2020-12-01T08:00:00.000Z",
    "endDate": "2020-12-31T20:00:00.000Z"
},
```

In this case user will have access only from monday to friday between 8:00 and 16:00

```
"repeatEvent": {
    "weekDays": 31,
    "dayStartTime": "2020-12-01T08:00:00.000Z",
    "dayEndTime": "2020-12-31T16:00:00.000Z",
    "startDate": null,
    "endDate": null
},
```

10.2 Add access to the device

Let’s consider that situation. You are responsible for managing access for users in your organization. If new employee is recruited you don’t want to give him keys to the office (or you don’t use keys in your organization). Instead you want to share door lock to him/her. To do that you need to use *Create share*. Simply call this endpoint with new organization email address to create new device share.

If user that you want to share device with already have Tedee account he will be notified that device was shared with him. If not the email with invitaion will be sent.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "deviceId": 1,
    "accessLevel": 1,
    "userEmail": "john.doe@email.com"
    "repeatEvent": {
        "weekDays": 10,
        "dayStartTime": "2020-12-14T08:09:57.781Z",
        "dayEndTime": "2020-12-31T08:10:57.781Z",
        "startDate": null,
        "endDate": null
    },
    "remoteAccessDisabled" : false
}
```

10.3 Update access to the device

If you want to change access to the door lock for your employees for example you want give some of them admin permissions, you can update user access to the device. For that you need to have shareId, which you get when creating share with success or you can simply use endpoint to get all share for the device *Get all shares*. When you have complete information you can send request *Update share* to update share.

Sample request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "accessLevel": 1,
    "repeatEvent": {
        "id": 1,
        "weekDays": 10,
        "dayStartTime": "2020-12-14T08:09:57.781Z",
        "dayEndTime": "2020-12-31T08:10:57.781Z",
        "startDate": null,
        "endDate": null
    },
    "remoteAccessDisabled" : false
}
```

10.4 Delete share

Let's consider different situation. Unfortunately, you need to fire one of your employee. After deleting access to organization resources you can also remove employee's access to the devices within organization with the call to the *Delete share* endpoint. For that you need to have share id you want to delete. You can get shares for each device from *Get all shares* endpoint.

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/deviceshare/15" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>"
```

CHAPTER 11

How to manage PIN list of the lock

If you have a lock paired with a keypad and your lock is also paired with a bridge, you can manage pins using API. In this tutorial we will walk through the process of listing, creating, updating, and deleting pins for the lock device.

11.1 Get current pin list

If you want to get the current pin list for the lock, you will need deviceId and use *Get all pins* endpoint.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

This endpoint will return all pins for the given lock.

Sample response

HTTP status code: `200`

```
{
    "result": {
        "listVersion": 2,
        "pins": [
            {
                "id": 1,
                "alias": "test pin 1"
            },
            {
                "id": 2,
                "alias": "test pin 2"
            }
        ]
    },
    "success": true,
```

(continues on next page)

(continued from previous page)

```
    "errorMessages": [],
    "statusCode": 200
}
```

Along with the pins, the version of the list is also returned. After each pin operation, such as creating, updating or deleting, the version of the pin list is incremented.

When getting the pins, you can optionally provide the last received version of the pin list. If there is a newer version of the list on the device, all pins will be returned, otherwise the response code 304 Not Modified will be returned.

Sample request with a list version parameter

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin?listVersion=2" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response when the provided version of the list is up-to-date

HTTP status code: `304`

```
{
    "success": false,
    "errorMessages": [
        "List version is up to date."
    ],
    "statusCode": 304
}
```

Sample response when a newer version of the list is available

HTTP status code: `200`

```
{
    "result": {
        "listVersion": 3,
        "pins": [
            {
                "id": 1,
                "alias": "test pin 1"
            },
            {
                "id": 2,
                "alias": "test pin 2"
            }
        ]
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

11.2 Get pin details

If you want to get details of a selected pin, you will need deviceId along with pinId and use *Get single pin* endpoint.

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

This endpoint will return pin details.

Sample response

HTTP status code: `200`


```
{
    "result": {
        "id": 2,
        "alias": "test pin 2",
        "pin": "192837",
        "startDate": "2021-10-01T00:00:00.000Z",
        "endDate": "2021-12-31T00:00:00.000Z",
        "dayStartTime": "2021-10-01T10:00:00.000Z",
        "dayEndTime": "2021-10-01T18:00:00.000Z",
        "weekDays": 1
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


11.3 Create a new pin

If you want to add a new pin for the given lock, you will need deviceId and use *Create pin* endpoint.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/pin" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

This endpoint will return Id of the created pin.

11.3.1 Create a pin with permanent access to the lock

If you want to have a pin with permanent access to the lock, you only need to send pin alias and pin value:

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273"
}
```

11.3.2 Create a pin with time restricted access to the lock

If you want to restrict pin access to the lock you can send fields “startDate” or “endDate”, it will mark the period when pin will be active. You can also restrict access to specific hours of the day by sending “dayStartTime” and “dayEndTime”. You can further customize this by selecting days. To send it proper way you need to use *Week days* enum.

In this case the created pin will be active from 1 October to 31 December:

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273",
    "startDate": "2021-10-01T00:00:00.000Z",
    "endDate": "2021-12-31T00:00:00.000Z",
    "dayStartTime": null,
    "dayEndTime": null,
    "weekDays": null
}
```

In this case the created pin will be active from 1 October to 31 December, and also will have access only on friday and saturday between 10:00 and 18:00:

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273",
    "startDate": "2021-10-01T00:00:00.000Z",
    "endDate": "2021-12-31T00:00:00.000Z",
    "dayStartTime": "2021-10-01T10:00:00.000Z",
    "dayEndTime": "2021-12-31T18:00:00.000Z",
    "weekDays": 48
}
```

In this case the created pin will be active only from monday to friday between 8:00 and 16:00:

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273",
    "startDate": null,
    "endDate": null,
    "dayStartTime": "2021-10-01T08:00:00.000Z",
    "dayEndTime": "2021-10-01T16:00:00.000Z",
    "weekDays": 31
}
```

11.4 Update selected pin

If you want to update the selected pin for the given lock, you will need deviceId and pinId. Before making any changes you should read the pin details to have complete information about the pin. To do that use *Get single pin* endpoint. Once you have all, you can send the updated information to the endpoint *Update pin* to update the pin.

Sample request

```
curl -X PUT "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "alias": "new test pin 2",
    "pin": "827364",
    "startDate": "2021-10-01T00:00:00.000Z",
    "endDate": "2021-12-31T00:00:00.000Z",
    "dayStartTime": "2021-10-01T10:00:00.000Z",
    "dayEndTime": "2021-10-01T18:00:00.000Z",
    "weekDays": 1
}
```

11.5 Delete pin

If you want to delete selected pin for the given lock, you will need deviceId along with pinId and use *Delete pin* endpoint.

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

CHAPTER 12

How to connect to Tedee device via Bluetooth

As an integrator, you may want to establish a secure connection between the Tedee device and your device over BLE. To do that you need to register your device and generate a certificate for it. In this tutorial, we will cover how to register a new integration device in Tedee Cloud (e.g. mobile device) and manage access certificates for it.

Note: Originally integration with Tedee devices was reserved only for mobile phones with Tedee App. Currently, we are extending the Mobile area to handle registration to our system by other external devices.

In this tutorial, we will name a device that wants to establish a secure connection over BLE with the Tedee device as **mobile**.

12.1 Prequisities

- Tedee device should be added to user account.

12.2 Step 1: Generate ECSDA auth pair key on the mobile

The Tedee devices are communicating only with other trusted devices. As a first step, on the mobile device, an ECDSA auth key pair should be generated.

Tedee devices are using the ECDSA cryptography algorithm with uses elliptic curve cryptography. The elliptic curve base point 256 is used.

Note:

We are recommending using libraries to generate the auth key pair:

- C Language: **MbedTLS**,
- iOS: **CryptoSwift** and native Apple framework **Security** for keys management,

- Java: **javax.crypto** and **java.security**

Warning: The generated ECDSA private key should be kept only on the mobile device and stored securely.

12.3 Step 2: Register mobile device

Only mobiles registered in Tedee Cloud can establish secure Bluetooth connections with Tedee devices. For those mobiles, Tedee Cloud is generating access certificates.

When the auth key pair is generated on the mobile device, then it can be registered in Tedee Cloud. If you are integrating an external device operating system flag should be set to 3.

To register new mobile you should use Register mobile endpoint.

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/mobile" -H "accept: application/json
↪" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Sample body

```
{
    "name": "<<integration device name>>",
    "operatingSystem": 3,
    "publicKey": "<<public as base64 key generated in step 1>>"
}
```

Sample response

```
{
    "result": {
        "id": 123,
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

If the registration succeed it means, your device is properly registered in Tedee Cloud. Mobile identifier should be saved.

12.4 Step 3: Get certificate for mobile device

To establish a secure BLE connection with the Tedee device, an access certificate is required. This certificate is read and validated by the Tedee device.

The access certificate is issued by Tedee Cloud with a 10-days validity period. The mobile device is responsible for requesting and refreshing the access certificates. We recommend refreshing the certificate when validity is about 40-50% remain validity period.

To generate the certificate you should use Get for mobile endpoint.

Sample request

```
GET https://api.tedee.com/api/v1.22/my/devicecertificate/getformobile?mobileId=123&
↪deviceId=1
```

In the response the certificate data will be returned.

Sample response

HTTP status code: `200`

```
{
    "result": {
        "certificate": "AQECAgECAwRhZ+ZwBAF/BQQAAAAABgQAAVF/
↪BRhZNAACARjRgOACQQAACLdCgQAAGHMCwgAAAF7ybAKin5BBKbnztHKIog
        8hD3/OqFWBI5/oNECVRyQm5EfxZyGz/Pv7oKvlXNkF2503/RCRgocotF6rVQaYsH9c5Xq4btSF/
↪RjBEAiDLNX00yWXmpIi0AigSb3veeFyEQRN
        sCRYbEwCZxkFe1gIgJEGKT6EoSHwPfYmPJsHCdcgtBQPiDXM/M2qJRbu6Pb4=",
        "expirationDate": "2021-12-12T00:00:00.000000Z",
        "devicePublicKey": "BL41FWWQ0SCxYr5aLWaCUA/88XsWkVJdxihYIN0kL9VKhE9jAx8+INXVG/
↪vsen/VEj9YltNMtb1I+qDTUdVqo8c=",
        "mobilePublicKey": "BkbnztKHIog8hD3/OqFWBI5/oNECVRyQm5EfXZyGz/Pv7oKvlXNkF2503/
↪RCRgocotF6rVQaYsH9c5Xq4btSYKE="
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

The 'result.certificate' field is containing the certificate issued by Tedee Cloud. The 'result.expirationDate' is containing the date when the access certificate will expire. The 'result.certificate' field value should be passed to the Tedee device as the certificate.

Note: The access certificate is returned in bytes in Base64 format.

For more details find **Tedee BLE API documentation**.

12.5 Step 4: Get time for Tedee device

To establish a secure BLE connection Tedee device require current time for proper work. When the Tedee device is not having a current time set, the mobile device is responsible for providing it.

To get current time from Tedee API you should use Get signed time endpoint.

```
curl -X GET "https://api.tedee.com/api/v1.22/datetime/getsignedtime" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "datetime" : "AAABfHgtDbU=",
        "signature" :
↪"MEQCIIa7NtKXDHDzynw0y5Gmi98HrA9UYnBimzXbixzaqoJiAiBDekhYsN7Eo0+d4so79zJFVni25kJJKlDklX04u7gEMA==
↪",
```

(continues on next page)

(continued from previous page)


```
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


More information

- Tedee BLE API documentation

CHAPTER 13

Example integrations

If you need you can take a look at integrations with our API prepared by third party developers.

Please note that these third party libraries are not developed or maintained by Tedee. You may find these examples useful, but note that these examples are not affiliated with or endorsed by Tedee.

TypeScript

- Homebridge

Python

- Home Assistant

CHAPTER 14

DateTime

Endpoints to get proper time for system.

14.1 Get signed time

Get signed time, required by Tedee device to secure Bluetooth connection.

```
GET https://api.tedee.com/api/v1.22/datetime/getsignedtime
```

14.1.1 Responses

Name	Type	Description
200 OK	*Signed time*	successful operation

14.1.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

14.1.3 Examples

Get signed time

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/datetime/getsignedtime" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`


```
{
    "result": {
        "datetime" : "AAABfHgtDbU=",
        "signature" :
↪"MEQCIIa7NtKXDHDzynw0y5Gmi98HrA9UYnBimzXbixzaqoJiAiBDekhYsN7Eo0+d4so79zJFVni25kJJKlDklX04u7gEMA==
↪",
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Operations

- Get signed time

CHAPTER 15

Device

Endpoints used for listing devices.

15.1 Get all with details

Get a list of all currently logged user devices with details such as devices specific data as well as their settings and share details.

```
GET https://api.tedee.com/api/v1.22/my/device/details
```

15.1.1 Responses

Name	Type	Description
200 OK	*Lock* []	successful operation
	Bridge []	

15.1.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

15.1.3 Examples

Get all devices with details of currently logged user

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/device/details" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: 200


```
{
    "result": {
        "bridges": [
        {
            "beaconMajor": 10,
            "beaconMinor": 10,
            "wasConfigured": true,
            "isUpdating": false,
            "timeZone": "Europe/Warsaw",
            "id": 2,
            "serialNumber": "111111-111112",
            "macAddress": "00:0A:95:9D:68:17",
            "name": "Bridge 1",
            "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
            "type": 1,
            "created": "2020-01-01T00:00:00",
            "revision": 1,
            "deviceRevision": 1,
            "targetDeviceRevision": 1,
            "isConnected": true,
            "accessLevel": 2,
            "shareDetails": null,
            "softwareVersions": [
            {
                "softwareType": 0,
                "version": "1.0.1",
                "updateAvailable": true
            },
            {
                "softwareType": 1,
                "version": "2.0.0",
                "updateAvailable": true
            }
            ]
        }
        ],
        "locks": [
        {
            "deviceSettings": {
                "autoLockEnabled": true,
                "autoLockDelay": 10,
                "autoLockImplicitEnabled": true,
                "autoLockImplicitDelay": 10,
                "pullSpringEnabled": true,
                "pullSpringDuration": 10,
                "autoPullSpringEnabled": true,
                "postponedLockEnabled": true,
                "postponedLockDelay": 10,
                "buttonLockEnabled": true,
                "buttonUnlockEnabled": true
            },
```


(continues on next page)

(continued from previous page)


```
            "userSettings": {
                "autoUnlockEnabled": true,
                "autoUnlockConfirmEnabled": true,
                "autoUnlockRangeIn": 300,
                "autoUnlockRangeOut": 400,
                "autoUnlockTimeout": 20,
                "location": {
                    "latitude": 52.24070739746092,
                    "longitude": 21.086990356445305
            },
            "lockProperties": {
                "state": 3,
                "isCharging": false,
                "batteryLevel": 18,
                "stateChangeResult": 0,
                "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
            },
            "beaconMajor": 10,
            "beaconMinor": 10,
            "timeZone": "Europe/Warsaw",
            "id": 1,
            "connectedToId": 2,
            "serialNumber": "111111-11111",
            "macAddress": "00:0A:95:9D:68:16",
            "name": "Room 6",
            "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
            "type": 2,
            "created": "2020-01-01T00:00:00",
            "revision": 2,
            "deviceRevision": 2,
            "targetDeviceRevision": 2,
            "isConnected": true,
            "accessLevel": 2,
            "shareDetails": null,
            "softwareVersions": [
                {
                    "softwareType": 0,
                    "version": "1.0.0",
                    "updateAvailable": true
                }
            ]
        }
        ]
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


15.2 Get device operation

Get operation status by it's id

```
GET https://api.tedee.com/api/v1.22/my/device/operation/{operationId}
```

15.2.1 Responses

Name	Type	Description
200 OK	*Device Operation*	successful operation

15.2.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

15.2.3 Examples

Get operation status by id

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/device/operation/1619078520230" -H
↪"accept: application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "deviceId" : 1,
        "operationId" : "1619078520230",
        "result" : 0,
        "status" : "COMPLETED"
        "type" : 1
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Operations

- Get all with details
- Get device operation

CHAPTER 16

Device activity

Endpoints used for listing device activities.

16.1 Get all

Get a list of activities for provided device id. Results are paged and sorted by date in descending order.

Users with guest access to the device can only see his/her activities and activities without the author (for example manual operations).

```
GET https://api.tedee.com/api/v1.22/my/deviceactivity?deviceId={id}&elements=
↪{elements}&lastElemDate={lastElemDate}
```

URI Parameters

Name	Type	Description
deviceId	number	id of device
elements	number (optional)	number of elements to load (max 200, default 200)
lastElemDate	datetime (optional)	when set, only items with a date older than the entered date (in UTC) are shown

16.1.1 Responses

Name	Type	Description
200 OK	*Device activity* []	successful operation

Note: This endpoint doesn't return all activities that exist. Maximum number of actvities you can get from this endpoint is 200. To get more actvity logs you need to use paging feature in that endpoint. To do that simply use URI

parameter lastElemDate. You need to set this parameter a value the same as the date of the last element date from previous page. For first page you can leave this parameter as it's optional and default value is current date in UTC.

16.1.2 Scopes

Name	Description
DeviceActivity.Read	Grants user possibility to read device activities

16.1.3 Examples

To better understand the idea of getting activities and paging mechanism we prepared following examples:

- Get first top activities for device
- Get next page of activities for device

Get first top activities for device

Example shows how to get first top activities. We will leave URI parameter lastElemDate and set that we want to receive 3 top activities.

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/deviceactivity?deviceId=2&elements=3"␣
↪-H "accept: application/json" -H "Authorization: Bearer <<access token>>"
```

Sample Response

HTTP status code: `200`


```
{
    "result": [
        {
            "id": 3,
            "deviceId": 2,
            "userId": 3,
            "username": "Test",
            "event": 32,
            "source": 0,
            "date": "2020-11-20T13:59:26.212"
        },
        {
            "id": 2,
            "deviceId": 2,
            "userId": 3,
            "username": "Test",
            "event": 33,
            "source": 0,
            "date": "2020-11-20T13:00:45.554"
        },
        {
            "id": 1,
            "deviceId": 2,
```


(continues on next page)

(continued from previous page)


```
            "userId": 3,
            "username": "Test",
            "event": 32,
            "source": 0,
            "date": "2020-11-20T12:50:45.600"
        }
    ],
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Get next page of activities for device

Example shows how to get next page of activities. Now we need to specify lastElemDate parameter because we want to receive next elements. To do that we take date from the last element from previous example. Also we take 3 activities.

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/deviceactivity?deviceId=2&elements=3&
↪lastElemDate=2020-11-20T12:50:45.600" -H "accept: application/json" -H
↪"Authorization: Bearer <<access token>>"
```

Sample Response

HTTP status code: `200`


```
{
    "result": [
        {
            "id": 6,
            "deviceId": 2,
            "userId": 3,
            "username": "Test",
            "event": 32,
            "source": 0,
            "date": "2020-11-20T12:49:26.212"
        },
        {
            "id": 5,
            "deviceId": 2,
            "userId": 3,
            "username": "Test",
            "event": 33,
            "source": 0,
            "date": "2020-11-20T12:23:45.554"
        },
        {
            "id": 4,
            "deviceId": 2,
            "userId": 3,
            "username": "Test",
            "event": 32,
            "source": 0,
            "date": "2020-11-20T12:10:45.600"
```


(continues on next page)

(continued from previous page)


```
        }
    ],
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Operations

- Get all

CHAPTER 17

Devices certificate

Endpoints for device certificate operations.

17.1 Get for mobile

Get certificate for specific mobile device endpoint. To establish secure PTLS connection (Secure Bluetooth Connection) with Tedee device required is certificate obtained from this endpoint.

Note: Refreshing the certificate is in scope of mobile.

Note: To keep the connection alive, we recommend to refresh the certificate when validity is about 40-50% remain validity period.

```
GET https://api.tedee.com/api/v1.22/my/devicecertificate/getformobile?mobileId=
↪{mobileId}&deviceId={deviceId}
```

URI Parameters

Name	Type	Description
mobileId	number	id of mobile device
deviceId	number	id of Tedee device

17.1.1 Responses

Name	Type	Description
200 OK	*Certificates for mobile*	successful operation

17.1.2 Scopes

Name	Description
DeviceCertificate.Operate	Grants user possibility to access devices certificates

17.1.3 Examples

Get certificate for mobile

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/devicecertificate/getformobile?
↪mobileId=123&deviceId=456" -H "accept: application/json" -H "Authorization: Bearer <
↪<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "certificate": "AQECAgECAwRhZ+ZwBAF/BQQAAAAABgQAAVF/
↪BRhZNAACARjRgOACQQAACLdCgQAAGHMCwgAAAF7ybAKin5BBKbnztHKIog
        8hD3/OqFWBI5/oNECVRyQm5EfxZyGz/Pv7oKvlXNkF2503/RCRgocotF6rVQaYsH9c5Xq4btSF/
↪RjBEAiDLNX00yWXmpIi0AigSb3veeFyEQRN
        sCRYbEwCZxkFe1gIgJEGKT6EoSHwPfYmPJsHCdcgtBQPiDXM/M2qJRbu6Pb4=",
        "expirationDate": "2021-12-12T00:00:00.000000Z",
        "devicePublicKey": "BL41FWWQ0SCxYr5aLWaCUA/88XsWkVJdxihYIN0kL9VKhE9jAx8+INXVG/
↪vsen/VEj9YltNMtb1I+qDTUdVqo8c=",
        "mobilePublicKey": "BkbnztKHIog8hD3/OqFWBI5/oNECVRyQm5EfXZyGz/Pv7oKvlXNkF2503/
↪RCRgocotF6rVQaYsH9c5Xq4btSYKE="
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

17.2 Get revoked for mobile

Get all revoked certificates for mobile device. When the Tedee devices is asking for revoked certificates, this endpoint is able to return the revoked certificates.

Note: For details when the Tedee device can ask for revoked certificates, please refer to: **Tedee BLE API Documentation**.

```
GET https://api.tedee.com/api/v1.22/my/devicecertificate/getrevokedformobile?deviceId=
↪{deviceId}&version={version}
```

URI Parameters

Name	Type	Description
deviceId	number	id of Tedee device
version	number	version of the last received revoked certificate list

17.2.1 Responses

Name	Type	Description
200 OK	*Revoked certificate list*	successful operation
204 No Content		version up to date

17.2.2 Scopes

Name	Description
DeviceCertificate.Operate	Grants user possibility to access devices certificates

17.2.3 Examples

Revoked certificates for mobile

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/devicecertificate/getrevokedformobile?
↪deviceId=123&version=1" -H "accept: application/json" -H "Authorization: Bearer <
↪<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "deviceId": 123,
        "version": 2,
        "lastUpdated": 1633692928,
        "signature": "MEUCICFNUa+s27psC6CFYV0KJ/f=g/6AU/
↪rS7+CWZMPahrstESAiEA5tOCveS4a1MWz4qMQN7b+cJhuFWcJjPXPr0Sl3GfCUQ=",
        "revokedCertificates": [
            {
                "certificateId": 982,
                "expirationDate": "2020-12-12T00:00:00.000000Z"
            }
        ]
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Version up to date

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/devicecertificate/getrevokedformobile?
↪deviceId=123&version=2" -H "accept: application/json" -H "Authorization: Bearer <
↪<access token>>"
```

Sample response

HTTP status code: `204`

Operations

- Get for mobile
- Get revoked for mobile

CHAPTER 18

Device share

Endpoints used for listing, creating, updating and removing access to the devices.

18.1 Create

Grants access to the device for new user. This endpoint can be used by owner or administrator of the device.

Note: User can have only one active share to the device at the time.

```
POST https://api.tedee.com/api/v1.22/my/deviceshare
```

Body Parameters

Name	Type	Description
accessLevel	*Access level*	represents user access level
deviceId	number	device id
remoteAccessDisabled	boolean	represents if remote access is disabled
repeatEvent	*Repeat event*	represents repeat event of the share
userEmail	string	user email that device will be shared with

18.1.1 Responses

Name	Type	Description
201 Created	*Device Share Success*	successful operation

18.1.2 Scopes

Name	Description
DeviceShare.ReadWrite	Grants user possibility to read and write data connected with device shares

18.1.3 Examples

Grant permanent administrator access

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "deviceId": 1,
    "accessLevel": 1,
    "userEmail": "john.doe@email.com"
    "repeatEvent": {
        "weekDays": null,
        "dayStartTime": null,
        "dayEndTime": null,
        "startDate": null,
        "endDate": null
    },
    "remoteAccessDisabled" : false
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 1,
        "sharedUserDisplayName": "John Doe"
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

Grant guest time restricted access

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "deviceId": 1,
    "accessLevel": 0,
    "userEmail": "john.doe@email.com"
    "repeatEvent": {
        "weekDays": null,
        "dayStartTime": null,
        "dayEndTime": null,
        "startDate": "2020-12-14T08:09:57.781Z",
        "endDate": "2020-12-31T08:10:57.781Z"
    },
    "remoteAccessDisabled" : false
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 1,
        "sharedUserDisplayName": "John Doe"
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

Grant guest custom access

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "deviceId": 1,
    "accessLevel": 0,
    "userEmail": "john.doe@email.com"
    "repeatEvent": {
        "weekDays": 7,
        "dayStartTime": "2020-12-01T08:00:00.000Z",
        "dayEndTime": "2020-12-31T20:00:00.000Z",
        "startDate": "2020-12-01T08:09:57.781Z",
        "endDate": "2020-12-31T23:10:57.781Z"
    },
    "remoteAccessDisabled" : false
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 1,
        "sharedUserDisplayName": "John Doe"
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```


18.2 Delete

This endpoint allows to remove access to the device from the user. To remove access from user you need to be owner or administrator of the device.

```
DELETE https://api.tedee.com/api/v1.22/my/deviceshare/{deviceShareId}
```

URI Parameters

Name	Type	Description
deviceShareId	number	id of share to delete

18.2.1 Responses

Name	Description
204 No Content	successful operation

18.2.2 Scopes

Name	Description
DeviceShare.ReadWrite	Grants user possibility to read and write data connected with device shares

18.2.3 Examples

Remove access to the device

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/deviceshare/15" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>"
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

18.3 Get all

Get all access details for specific device. This endpoint can be used by all users that have share to the device but users with access level "Guest" and those users that share are not active will get in response only information about owner of device and own access details. Owner and administrators with active share will get all users access details.

```
GET https://api.tedee.com/api/v1.22/my/deviceshare?deviceId={id}
```

URI Parameters

Name	Type	Description
deviceId	number	id of device

18.3.1 Responses

Name	Type	Description
200 OK	*Share details* []	successful operation

18.3.2 Scopes

Name	Description
DeviceShare.Read	Grants user possibility to read data connected with device shares
DeviceShare.ReadWrite	Grants user possibility to read and write data connected with device shares

18.3.3 Examples

Get shares

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/deviceShare?deviceId=1" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": [
        {
            "id": 15,
```

(continues on next page)

(continued from previous page)


```
            "userId": 11,
            "deviceId": 1,
            "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
            "accessLevel": 1,
            "accessType": 0,
            "userEmail": "john.doe@email.com",
            "isPending": false,
            "userDisplayName": "John Doe",
            "repeatEvent": {
                "id": 1,
                "weekDays": 10,
                "dayStartTime": "2020-12-14T08:09:57.781Z",
                "dayEndTime": "2020-12-31T08:10:57.781Z",
                "startDate": null,
                "endDate": null
            },
            "remoteAccessDisabled": true
        },
                        {
            "id": 16,
            "userId": 12,
            "deviceId": 1,
            "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba2eaf7333",
            "accessLevel": 0,
            "accessType": 1,
            "userEmail": "john.kowalsky@email.com",
            "isPending": false,
            "userDisplayName": "John Doe",
            "repeatEvent": {
                "id": 1,
                "weekDays": 10,
                "dayStartTime": "2020-12-14T08:09:57.781Z",
                "dayEndTime": "2020-12-31T08:10:57.781Z",
                "startDate": null,
                "endDate": null
            },
            "remoteAccessDisabled": false
        }
    ]
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


18.4 Update

Allows to update access details to the device for specific share. This endpoint can be used by owner or administrator of the device.

```
PATCH https://api.tedee.com/api/v1.22/my/deviceshare
```

Body Parameters

Name	Type	Description
accessLevel	*Access level*	represents user access level
id	number	id of share
remoteAccessDisabled	boolean	represents if remote access is disabled
repeatEvent	*Repeat event*	represents repeat event of the share

18.4.1 Responses

Name	Description
204 No Content	successful operation

18.4.2 Scopes

Name	Description
DeviceShare.ReadWrite	Grants user possibility to read and write data connected with device shares

18.4.3 Examples

Update share

Sample Request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/deviceshare" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "accessLevel": 1,
    "repeatEvent": {
        "id": 1,
        "weekDays": 10,
        "dayStartTime": "2020-12-14T08:09:57.781Z",
        "dayEndTime": "2020-12-31T08:10:57.781Z",
        "startDate": null,
        "endDate": null
    },
    "remoteAccessDisabled" : false
}
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
```

(continues on next page)

(continued from previous page)

```
    "statusCode": 204
}
```

Operations

- Create
- Delete
- Get all
- Update

CHAPTER 19

Lock

Endpoints used for listing, updating and operating locks.

19.1 Get all

Get a list of all currently logged user locks.

```
GET https://api.tedee.com/api/v1.22/my/lock
```

19.1.1 Responses

Name	Type	Description
200 OK	*Lock* []	successful operation

19.1.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

19.1.3 Examples

Get all locks of currently logged user

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json" -
↪H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: 200


```
{
    "result": [
        {
            "deviceSettings": {
                "autoLockEnabled": true,
                "autoLockDelay": 10,
                "autoLockImplicitEnabled": false,
                "autoLockImplicitDelay": 10,
                "pullSpringEnabled": true,
                "pullSpringDuration": 10,
                "autoPullSpringEnabled": false,
                "postponedLockEnabled": false,
                "postponedLockDelay": 10,
                "buttonLockEnabled": false,
                "buttonUnlockEnabled": false
            },
            "userSettings": {
                "autoUnlockEnabled": true,
                "autoUnlockConfirmEnabled": true,
                "autoUnlockRangeIn": 300,
                "autoUnlockRangeOut": 400,
                "autoUnlockTimeout": 20,
                "location": {
                    "latitude": 52.24070739746092,
                    "longitude": 21.086990356445305
            },
            "lockProperties": {
                "state": 3,
                "isCharging": false,
                "batteryLevel": 18,
                "stateChangeResult": 0,
                "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
            },
            "beaconMajor": 56,
            "beaconMinor": 57,
            "timeZone": "Europe/Warsaw",
            "id": 1,
            "connectedToId": 2,
            "serialNumber": "111111-11111",
            "macAddress": "00:0A:95:9D:68:16",
            "name": "Room 6",
            "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
            "type": 2,
            "created": "2020-01-01T00:00:00",
            "revision": 2,
            "deviceRevision": 2,
            "targetDeviceRevision": 2,
            "isConnected": true,
            "accessLevel": 2,
            "shareDetails": null,
            "softwareVersions": [
```


(continues on next page)

(continued from previous page)


```
                {
                    "softwareType": 0,
                    "version": "1.0.0",
                    "updateAvailable": true
                }
            ]
        }
    ]
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


19.2 Get single

Get lock by provided id.

```
GET https://api.tedee.com/api/v1.22/my/lock/{id}
```

URI Parameters

Name	Type	Description
id	number	id of lock

19.2.1 Responses

Name	Type	Description
200 OK	*Lock*	successful operation

19.2.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

19.2.3 Examples

Get single lock

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1" -H "accept: application/json"␣
↪-H "Authorization: Bearer <<access token>>"
```

Sample response for lock paired with bridge

HTTP status code: `200`

```
{
    "result": {
        "deviceSettings": {
            "autoLockEnabled": true,
            "autoLockDelay": 10,
            "autoLockImplicitEnabled": false,
            "autoLockImplicitDelay": 10,
            "pullSpringEnabled": true,
            "pullSpringDuration": 10,
            "autoPullSpringEnabled": false,
            "postponedLockEnabled": false,
            "postponedLockDelay": 10,
            "buttonLockEnabled": false,
            "buttonUnlockEnabled": false
        },
        "userSettings": {
            "autoUnlockEnabled": true,
            "autoUnlockConfirmEnabled": true,
            "autoUnlockRangeIn": 300,
            "autoUnlockRangeOut": 400,
            "autoUnlockTimeout": 20,
            "location": {
                "latitude": 52.24070739746092,
                "longitude": 21.086990356445305
        },
        "lockProperties": {
            "state": 3,
            "isCharging": false,
            "batteryLevel": 18,
            "stateChangeResult": 0,
            "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
        },
        "beaconMajor": 56,
        "beaconMinor": 57,
        "timeZone": "Europe/Warsaw",
        "id": 1,
        "connectedToId": 2,
        "serialNumber": "111111-11111",
        "macAddress": "00:0A:95:9D:68:16",
        "name": "Room 6",
        "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
        "type": 2,
        "created": "2020-01-01T00:00:00",
        "revision": 2,
        "deviceRevision": 2,
        "targetDeviceRevision": 2,
        "isConnected": true,
        "accessLevel": 2,
        "shareDetails": null,
        "softwareVersions": [
            {
                "softwareType": 0,
                "version": "1.0.0",
                "updateAvailable": true
            }
        ]
    }
```


(continues on next page)

(continued from previous page)


```
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Sample response for lock not paired with bridge

HTTP status code: `200`


```
{
    "result": {
        "deviceSettings": {
            "autoLockEnabled": true,
            "autoLockDelay": 10,
            "autoLockImplicitEnabled": false,
            "autoLockImplicitDelay": 10,
            "pullSpringEnabled": true,
            "pullSpringDuration": 10,
            "autoPullSpringEnabled": false,
            "postponedLockEnabled": false,
            "postponedLockDelay": 10,
            "buttonLockEnabled": false,
            "buttonUnlockEnabled": false
        },
        "userSettings": {
            "autoUnlockEnabled": true,
            "autoUnlockConfirmEnabled": true,
            "autoUnlockRangeIn": 300,
            "autoUnlockRangeOut": 400,
            "autoUnlockTimeout": 20,
            "location": {
                "latitude": 52.24070739746092,
                "longitude": 21.086990356445305
        },
        "lockProperties": null,
        "beaconMajor": null,
        "beaconMinor": null,
        "id": 1,
        "connectedToId": null,
        "serialNumber": "111111-11111",
        "macAddress": "00:0A:95:9D:68:16",
        "name": "Room 6",
        "userIdentity": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
        "type": 2,
        "created": "2020-01-01T00:00:00",
        "revision": 2,
        "deviceRevision": 2,
        "targetDeviceRevision": 2,
        "isConnected": null,
        "accessLevel": 2,
        "shareDetails": null,
        "softwareVersions": [
            {
                "softwareType": 0,
                "version": "1.0.0",
                "updateAvailable": true
            }
```


(continues on next page)

(continued from previous page)

```
        ]
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

19.3 Lock

Send command to lock the device by provided id.

```
POST https://api.tedee.com/api/v1.22/my/lock/{id}/operation/lock
```

URI Parameters

Name	Type	Description
id	number	id of lock

19.3.1 Responses

Name	Type	Description
202 Accepted	*Execute command response*	successful operation

19.3.2 Scopes

Name	Description
Lock.Operate	Grants user possibility to operate locks

19.3.3 Examples

Lock

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/lock" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
```

(continues on next page)

(continued from previous page)

```
    "success": true,
    "errorMessages": [],
    "statusCode": 202
}
```

19.4 Pull spring

Send command to pull spring the device by provided id.

```
POST https://api.tedee.com/api/v1.22/my/lock/{id}/operation/pull
```

URI Parameters

Name	Type	Description
id	number	id of lock

19.4.1 Responses

Name	Type	Description
202 Accepted	*Execute command response*	successful operation

19.4.2 Scopes

Name	Description
Lock.Operate	Grants user possibility to operate locks

19.4.3 Examples

Pull spring

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/pull" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
    "success": true,
    "errorMessages": [],
```

(continues on next page)

(continued from previous page)

```
    "statusCode": 202
}
```

19.5 Sync

Sync all users locks states.

Warning: You shouldn't run this endpoint more than once every 10 seconds.

```
GET https://api.tedee.com/api/v1.22/my/lock/sync
```

19.5.1 Responses

Name	Type	Description
200 OK	*Lock sync* []	successful operation

19.5.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

19.5.3 Examples

Sync all users locks

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/sync" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response for lock connected to bridge

HTTP status code: `200`

```
{
    "result": [
        {
            "id": 1,
            "isConnected": true,
            "lockProperties": {
                "state": 3,
                "isCharging": false,
                "batteryLevel": 54,
                "stateChangeResult": 0,
```

(continues on next page)

(continued from previous page)


```
                "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
            }
        }
    ],
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Sample response for lock disconnected from bridge

HTTP status code: `200`


```
{
    "result": [
        {
            "id": 1,
            "isConnected": false,
            "lockProperties": null
        }
    ],
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


19.6 Sync single

Sync single lock state by id.

```
GET https://api.tedee.com/api/v1.22/my/lock/{id}/sync
```

URI Parameters

Name	Type	Description
id	number	id of lock to sync

19.6.1 Responses

Name	Type	Description
200 OK	*Lock sync*	successful operation

19.6.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

19.6.3 Examples

Sync single lock

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/sync" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response for lock connected to bridge

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "isConnected": true,
        "lockProperties": {
            "state": 3,
            "isCharging": false,
            "batteryLevel": 54,
            "stateChangeResult": 0,
            "lastStateChangedDate": "2021-04-26T06:02:04.197Z"
        }
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Sample response for lock disconnected from bridge

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "isConnected": false,
        "lockProperties": null
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

19.7 Unlock

Send command to unlock the device by provided id.

```
POST https://api.tedee.com/api/v1.22/my/lock/{id}/operation/unlock?mode={mode}
```

URI Parameters

Name	Type	Description
id	number	id of lock
mode	*Unlock mode* (optional)	behaviour of unlocking door

19.7.1 Responses

Name	Type	Description
202 Accepted	*Execute command response*	successful operation

19.7.2 Scopes

Name	Description
Lock.Operate	Grants user possibility to operate locks

19.7.3 Examples

Unlock without optional parameter

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/unlock" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 202
}
```

Unlock with Force mode

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/unlock?mode=2" -H
↪"accept: application/json" -H "Content-Type: application/json-patch+json" -H
↪"Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
```

(continues on next page)

(continued from previous page)

```
    "success": true,
    "errorMessages": [],
    "statusCode": 202
}
```

Unlock without auto pull spring

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/unlock?mode=3" -H
↪"accept: application/json" -H "Content-Type: application/json-patch+json" -H
↪"Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 202
}
```

Unlock or pull spring

Sample Request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/operation/unlock?mode=4" -H
↪"accept: application/json" -H "Content-Type: application/json-patch+json" -H
↪"Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `202`

```
{
    "result": {
        "operationId": "1577833200000",
        "lastStateChangedDate": "2021-04-26T05:53:57.423Z"
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 202
}
```

19.8 Update

Update lock.

```
PATCH https://api.tedee.com/api/v1.22/my/lock
```

Body Parameters

Name	Type	Description
id	number	id of lock
revision	number	current lock information and settings in database
deviceSettings	*Device settings* (optional)	device settings to be updated
name	string (optional)	lock name
userSettings	*User settings* (optional)	settings of current user for that device

All parameters in this endpoint (except id and revision) are optional. This means that specifying a given parameter will update its value. If a given parameter is not specified, its value will not change.

Only the owner or admin can update device settings and name. Guest can only modify user settings.

19.8.1 Responses

Name	Type	Description
200 OK	*Lock updated*	successful operation
409 Conflict		revision in request is different than in database

19.8.2 Scopes

Name	Description
Device.ReadWrite	Grants user possibility to read and write data connected with devices

19.8.3 Examples

To better understand the idea of updating lock we prepared following examples:

- Update name of the lock
- Update single device setting
- Update single user setting
- Update location for auto unlock feature

Update name of the lock

Example shows how to update name of the lock with id = 1. Only owner or admin can update name of the device.

Sample Request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 1,
    "name": "New Name"
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 2,
        "targetDeviceRevision": 1
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Update single device setting

Example shows how to update single device setting (as presented below it is auto lock delay) of the lock with id = 1. Only owner or admin can update name of the device.

Sample Request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 1,
    "deviceSettings": {
        "autoLockDelay": 10
    }
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 2,
        "targetDeviceRevision": 2
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Note: Take a look at response of that request. TargetDeviceRevision changed as well as revision. It is because changing any device setting will change actual settings on the device.

Update single user setting

Example shows how to update single user setting (as presented below it is auto unlock) of the lock with id = 1. This action can be performed by any user with active share to that device.

Sample Request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 1,
    "userSettings":{
        "autoUnlockEnabled": true
    }
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 2,
        "targetDeviceRevision": 1
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Update location for auto unlock feature

Example shows how to change location of the lock with id = 1. This action can be performed by any user with active share to that device.

Note: Changing location has sense only if user has enabled auto unlock feature.

Sample Request

```
curl -X PATCH "https://api.tedee.com/api/v1.22/my/lock" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "id": 1,
    "revision": 1,
    "userSettings": {
        "location": {
            "latitude": 52.24070739746092,
            "longitude": 21.086990356445305
        }
    }
}
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 1,
        "revision": 2,
        "targetDeviceRevision": 1
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Operations

- Get all
- Get single
- Lock
- Pull spring
- Sync
- Sync single
- Unlock
- Update

CHAPTER 20

Lock PIN

Endpoints used for listing, creating, updating and deleting lock pins.

20.1 Get all

Get a list of all pins for the given lock.

```
GET https://api.tedee.com/api/v1.22/my/lock/{id}/pin?listVersion={listVersion}
```

URI Parameters

Name	Type	Description
id	number	id of the lock
listVersion	number (optional)	last received version of the pin list

20.1.1 Responses

Name	Type	Description
200 OK	*Lock PIN list* []	successful operation
304 Not Modified		provided list version is up to date
403 Forbidden		user doesn't have permission to the lock
408 Request Timeout		timeout while fetching data from the device
409 Conflict		other request is currently processing

20.1.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

20.1.3 Examples

Get a list of all pins for the lock with id 1

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "listVersion": 2,
        "pins": [
            {
                "id": 1,
                "alias": "test pin 1"
            },
            {
                "id": 2,
                "alias": "test pin 2"
            }
        ]
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

Attempting to get a list of pins after providing the current version of the list

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin?listVersion=2" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `304`

```
{
    "success": false,
    "errorMessages": [
        "List version is up to date."
    ],
```

(continues on next page)

(continued from previous page)

```
    "statusCode": 304
}
```

20.2 Get single

Get pin details by provided pin id.

```
GET https://api.tedee.com/api/v1.22/my/lock/{id}/pin/{pinId}
```

URI Parameters

Name	Type	Description
id	number	id of the lock
pinId	number	id of the pin

20.2.1 Responses

Name	Type	Description
200 OK	*Lock PIN details*	successful operation
403 Forbidden		user doesn't have permission to the lock
404 Not Found		pin not found
408 Request Timeout		timeout while fetching data from the device
409 Conflict		other request is currently processing

20.2.2 Scopes

Name	Description
Device.Read	Grants user possibility to read data connected with devices
Device.ReadWrite	Grants user possibility to read and write data connected with devices

20.2.3 Examples

Get pin details of pin id 2 for the lock with id 1

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept: application/
↪json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": {
        "id": 2,
        "alias": "test pin 2",
        "pin": "192837",
        "startDate": "2021-10-01T00:00:00.000Z",
        "endDate": "2021-12-31T00:00:00.000Z",
        "dayStartTime": "2021-10-01T10:00:00.000Z",
        "dayEndTime": "2021-10-01T18:00:00.000Z",
        "weekDays": 1
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


Sample response

HTTP status code: `200`


```
{
    "result": {
        "id": 2,
        "alias": "test pin 2",
        "pin": "192837",
        "startDate": null,
        "endDate": null,
        "dayStartTime": null,
        "dayEndTime": null,
        "weekDays": null
    },
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


20.3 Create

Creates a new pin for the given lock.

```
POST https://api.tedee.com/api/v1.22/my/lock/{id}/pin
```

URI Parameters

Name	Type	Description
id	number	id of the lock

Body Parameters

Name	Type	Description
alias	string	name of the pin
dayEndTime	datetime (optional)	end time when pin has access to the device
dayStartTime	datetime (optional)	start time when pin has access to the device
endDate	datetime (optional)	end date when pin has access to the device
pin	string	value of the pin
startDate	datetime (optional)	start date when pin has access to the device
weekDays	number (optional)	week days when pin has access to the device

20.3.1 Responses

Name	Type	Description
201 Created	*Pin created*	successful operation
403 Forbidden		user doesn't have permission to the lock
406 Not Acceptable		pin already exists
408 Request Timeout		timeout while sending data to the device
409 Conflict		other request is currently processing
422 Unprocessable Entity		adding more pins is not available

20.3.2 Scopes

Name	Description
Device.ReadWrite	Grants user possibility to read and write data connected with devices

---

Note: Pin value must meet the following requirements:

- pin cannot be null, empty, or whitespace
- pin length must be in range 5-8
- pin can contain only numeric values (0-9)
- pin must contain at least 3 different digits
- pin cannot be built as ascending or descending sequence

20.3.3 Examples

Create a pin with permanent access to the lock with id 1

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/pin" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273"
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 3
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

Create a pin with restricted access to the lock with id 1

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/lock/1/pin" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "alias": "test pin 3",
    "pin": "918273",
    "startDate": "2021-10-01T00:00:00.000Z",
    "endDate": "2021-12-31T00:00:00.000Z",
    "dayStartTime": "2021-10-01T10:00:00.000Z",
    "dayEndTime": "2021-10-01T18:00:00.000Z",
    "weekDays": 1
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 3
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

20.4 Update

Updates selected pin for the given lock.

```
PUT https://api.tedee.com/api/v1.22/my/lock/{id}/pin/{pinId}
```

URI Parameters

Name	Type	Description
id	number	id of the lock
pinId	number	id of the pin

Body Parameters

Name	Type	Description
alias	string	name of the pin
dayEndTime	datetime (optional)	end time when pin has access to the device
dayStartTime	datetime (optional)	start time when pin has access to the device
endDate	datetime (optional)	end date when pin has access to the device
pin	string	value of the pin
startDate	datetime (optional)	start date when pin has access to the device
weekDays	number (optional)	week days when pin has access to the device

20.4.1 Responses

Name	Description
204 No Content	successful operation
403 Forbidden	user doesn't have permission to the lock
404 Not Found	pin not found
406 Not Acceptable	pin already exists
408 Request Timeout	timeout while sending data to the device
409 Conflict	other request is currently processing

20.4.2 Scopes

Name	Description
Device.ReadWrite	Grants user possibility to read and write data connected with devices

---

Note: Pin value must meet the following requirements:

- pin cannot be null, empty, or whitespace
- pin length must be in range 5-8
- pin can contain only numeric values (0-9)
- pin must contain at least 3 different digits
- pin cannot be built as ascending or descending sequence

20.4.3 Examples

Update pin with id 2 for the lock with id 1 (permanent access)

Sample request

```
curl -X PUT "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "alias": "new test pin 2",
    "pin": "827364"
}
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

Update pin with id 2 for the lock with id 1 (restricted access)

Sample request

```
curl -X PUT "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Body:

```
{
    "alias": "new test pin 2",
    "pin": "827364",
    "startDate": "2021-10-01T00:00:00.000Z",
    "endDate": "2021-12-31T00:00:00.000Z",
    "dayStartTime": "2021-10-01T10:00:00.000Z",
    "dayEndTime": "2021-10-01T18:00:00.000Z",
    "weekDays": 1
}
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

20.5 Delete

Deletes selected pin for the given lock.

```
DELETE https://api.tedee.com/api/v1.22/my/lock/{id}/pin/{pinId}
```

URI Parameters

Name	Type	Description
id	number	id of the lock
pinId	number	id of the pin

20.5.1 Responses

Name	Description
204 No Content	successful operation
403 Forbidden	user doesn't have permission to the lock
404 Not Found	pin not found
408 Request Timeout	timeout while sending data to the device
409 Conflict	other request is currently processing

20.5.2 Scopes

Name	Description
Device.ReadWrite	Grants user possibility to read and write data connected with devices

20.5.3 Examples

Delete pin with id 2 for the lock with id 1

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/lock/1/pin/2" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

Operations

- Get all
- Get single

- Create
- Update
- Delete

CHAPTER 21

Mobile

Endpoints used for creating, listing, and deleting mobile devices.

21.1 Register

Register mobile device to user account endpoint. User is required to register all devices which wants to establish connection with Tedee devices. When registering a key pair is generated on the user's access device. The auth pair key is necessary to establish secure PTLS session (Secure Bluetooth Communication) with Tedee device.

Note: Auth pair key generation is user responsibility.

Warning: Remember to keep private auth pair ECDSA private key only in mobile device.

```
POST https://api.tedee.com/api/v1.22/my/mobile
```

Body Parameters

Name	Type	Description
name	string	name of the resource
operatingSystem	*Operating system*	represents operating system
publicKey	string	generated auth pair ECDSA public key

21.1.1 Responses

Name	Type	Description
201 Created	*Mobile identifier*	successful operation

21.1.2 Scopes

Name	Description
Mobile.ReadWrite	Grants the ability to manage user mobile or other devices.

21.1.3 Examples

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/mobile" -H "accept: application/json
↪" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Body:

```
{
    "name": "integrationDevice",
    "operatingSystem": 3,
    "publicKey": "BL4lFWWQ0SCxYr5aLWaCUA/88XsWkVJdxihYIN0kL9VKhE9jAx8+INXVG/vsen/
↪VEj9YltNMtb1I+qDTUdVqo8c="
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": 123,
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

21.2 Show mobiles

Show mobile devices registered on user account.

```
GET https://api.tedee.com/api/v1.22/my/mobile
```

21.2.1 Responses

Name	Type	Description
200 OK	*Registered mobiles* []	successful operation

21.2.2 Scopes

Name	Description
Mobile.Read	Grants the ability to view user registered mobiles.
Mobile.ReadWrite	Grants the ability to manage user mobiles.

21.2.3 Examples

Sample request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/mobile" -H "accept: application/json"␣
↪-H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <<access␣
↪token>>" -d "<<body>>"
```

Sample response

HTTP status code: `200`


```
{
    "result": [
        {
        "id": 123,
        "userIdentity": "00000000-0000-0000-0000-000000000001",
        "name": "iPhone Device",
        "operatingSystem": 0
        },
        {
        "id": 456,
        "userIdentity": "00000000-0000-0000-0000-000000000002",
        "name": "Android Device",
        "operatingSystem": 1
        },
        {
        "id": 789,
        "userIdentity": "00000000-0000-0000-0000-000000000003",
        "name": "Integration Device",
        "operatingSystem": 3
        }
    ]
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```


21.3 Delete

Remove mobile device from user account.

```
DELETE https://api.tedee.com/api/v1.22/my/mobile/{id}
```

URI Parameters

Name	Type	Description
id	number	mobile identidfier

21.3.1 Responses

Name	Description
204 No Content	successful operation

21.3.2 Scopes

Name	Description
Mobile.ReadWrite	Grants the ability to manage user mobile or other devices.

21.3.3 Examples

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/mobile/1" -H "accept: application/
↪json" -H "Content-Type: application/json-patch+json" -H "Authorization: Bearer <
↪<access token>>" -d "<<body>>"
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

Operations

- Register
- Show mobiles

CHAPTER 22

Personal access key

Endpoints used for listing, creating, updating and removing personal access keys.

22.1 Create

Creates new personal access key.

```
POST https://api.tedee.com/api/v1.22/my/personalaccesskey
```

Body Parameters

Name	Type	Description
name	string	name of the personal access key
validTo	datetime	date when key expires (max 5 years)
scopes	string[]	table of *scopes* that is assigned to key

22.1.1 Responses

Name	Type	Description
201 Created	*Personal access key created*	successful operation

22.1.2 Scopes

Name	Description
Account.ReadWrite	Grants user possibility to read and write data connected with account

22.1.3 Examples

Sample request

```
curl -X POST "https://api.tedee.com/api/v1.22/my/personalaccesskey" -H "accept:␣
↪application/json" -H "Content-Type: application/json-patch+json" -H "Authorization:␣
↪Bearer <<access token>>" -d "<<body>>"
```

Body:

```
{
    "name": "SomeExampleKeyName",
    "validTo": "2021-04-26T06:02:04.197Z",
    "scopes": [
        "Device.Read",
        "Organization.ReadWrite"
    ]
}
```

Sample response

HTTP status code: `201`

```
{
    "result": {
        "id": "bcc1fdc9-13ee-43b3-a13e-eaba8eaf7996",
        "key": "smnxaz.IWA6u00VLQmA8tlfioDXcH+bSiI6u8LgTG9cv3Evh/E"
    }
    "success": true,
    "errorMessages": [],
    "statusCode": 201
}
```

Your Personal Access Key is "smnxaz.IWA6u00VLQmA8tlfioDXcH+bSiI6u8LgTG9cv3Evh/E". You can use it to *authenticate*.

Warning: Keep your Personal Access Key safe!

Note: Personal Access Key cannot be read in any other way.

22.2 Get all

Get a list of all currently logged user personal access keys.

```
GET https://api.tedee.com/api/v1.22/my/personalaccesskey
```

22.2.1 Responses

Name	Type	Description
200 OK	*Personal access key* []	successful operation

22.2.2 Scopes

Name	Description
Account.Read	Grants user possibility to read data connected with account

22.2.3 Examples

Get all personal access keys of currently logged user

Sample Request

```
curl -X GET "https://api.tedee.com/api/v1.22/my/personalaccesskey" -H "accept:␣
↪application/json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `200`

```
{
    "result": [
        {
            "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
            "name": "SomeExampleName",
            "prefix": "gdQAs5",
            "scopes": [
                "Device.Read",
                "Devie.Write",
            ],
            "validTo": "2021-09-10T12:45:48.871Z"
        }
    ]
    "success": true,
    "errorMessages": [],
    "statusCode": 200
}
```

22.3 Update

Update personal access key.

```
PUT https://api.tedee.com/api/v1.22/my/personalaccesskey/{id}
```

URI Parameters

Name	Type	Description
id	UUID	id of the personal access key

Body Parameters

Name	Type	Description
name	string	name of the personal access key
validTo	datetime	date when key expires (max 5 years)
scopes	string[]	table of *scopes* that is assigned to key

22.3.1 Responses

Name	Description
204 No Content	successful operation

22.3.2 Scopes

Name	Description
Account.ReadWrite	Grants user possibility to read and write data connected with account

22.3.3 Examples

Sample request

```
curl -X PUT "https://api.tedee.com/api/v1.22/my/personalaccesskey/bcc1fdc9-13ee-43b3-
↪a13e-eaba8eaf7996" -H "accept: application/json" -H "Content-Type: application/json-
↪patch+json" -H "Authorization: Bearer <<access token>>" -d "<<body>>"
```

Body:

```
{
    "name": "SomeExampleKeyName",
    "validTo": "2021-04-26T06:02:04.197Z",
    "scopes": [
        "Device.Read",
        "Organization.ReadWrite"
    ]
}
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

22.4 Delete

Delete personal access key.

```
DELETE https://api.tedee.com/api/v1.22/my/personalaccesskey/{id}
```

URI Parameters

Name	Type	Description
id	UUID	id of the personal access key

22.4.1 Responses

Name	Description
204 No Content	successful operation

22.4.2 Scopes

Name	Description
Account.ReadWrite	Grants user possibility to read and write data connected with account

22.4.3 Examples

Sample request

```
curl -X DELETE "https://api.tedee.com/api/v1.22/my/personalaccesskey/bcc1fdc9-13ee-
↪43b3-a13e-eaba8eaf7996" -H "accept: application/json" -H "Content-Type: application/
↪json-patch+json" -H "Authorization: Bearer <<access token>>"
```

Sample response

HTTP status code: `204`

```
{
    "success": true,
    "errorMessages": [],
    "statusCode": 204
}
```

Operations

- Create
- Get all
- Update
- Delete

CHAPTER 23

Bridge

Name	Type	Description
accessLevel	*Access level*	current user access level
beaconMajor	number	identifies and distinguishes group of beacons
beaconMinor	number	identifies individual beacon within a group of beacons assigned a major value
bridgeId	number	id of bridge
created	datetime	date when bridge was added to user account
deviceRevision	number	current bridge revision on the device
id	number	bridge id
isConnected	boolean	is bridge connected
isUpdating	boolean	is bridge currently updating
macAddress	string	bridge mac address
name	string	bridge name
revision	number	current bridge revision in database
serialNumber	string	bridge serial number
shareDetails	*Share details*	share details of current user for that device
softwareVersions	*Software version*	software versions of the device
targetDeviceRevision	number	revision info that is sent to device from backend
timeZone	string	timezone of bridge
type	*Device type*	device type
userIdentity	string	bridge owner identity
wasConfigured	boolean	is wifi was configured on bridge

CHAPTER 24

Certificate for mobile

Name	Type	Description
certificate	string	certificate string
devicePublicKey	string	Tedee device public key
expirationDate	dateTime	certificate expiration date
mobilePublicKey	string	mobile public key

CHAPTER 25

Device activity

Name	Type	Description
date	datetime	date of activity in UTC
deviceId	number	device id
event	*Event type*	event type
id	number	id of activity
source	*Source*	source of activity
userId	number	id of user who performed activity (if known)
userName	string	name of user who performed activity (if known)

CHAPTER 26

Device operation

Name	Type	Description
deviceId	number	device id
operationId	string	operation id
result	number	result of operation (0 - success, > 0 - error)
status	string	status of operation can be COMPLETED or PENDING
type	*Operation type*	type of operation

CHAPTER 27

Device settings

Name	Type	Description
autoLockDelay	number	the delay for auto lock. If **autoLockEnabled** is set to true lock will automatically lock after specified delay
autoLockEnabled	boolean	if auto lock is enabled
autoLockImplicitDelay	number	the delay of auto lock from semi-locked state. If **autoLockImplicitEnabled** is set to true lock will automatically lock from semi-locked state after specified delay
autoLockImplicitEnabled	boolean	if auto lock from semi-locked state is enabled
autoPullSpringEnabled	boolean	auto pull spring is enabled. If enabled during unlocking process lock will also perform pull spring
buttonLockEnabled	boolean	if locking on button is enabled
buttonUnlockEnabled	boolean	if unlocking on button is enabled
postponedLockDelay	number	postponed delay time of lock. If **postponedLockEnabled** is set to true locking on button will be postponed by specified delay
postponedLockEnabled	boolean	if postponed lock is enabled
pullSpringDuration	number	the duration of pull spring
pullSpringEnabled	boolean	if pull spring is enabled. Setting it to false will reset pull spring calibration

CHAPTER 28

Device share success

Name	Type	Description
id	number	device share id
sharedUserDisplayName	string	share user display name

CHAPTER 29

Execute command response

Response from requests:

- *Lock command*
- *Unlock command*
- *Pull spring command*

Name	Type	Description
operationId	string	epoch time in miliseconds. Used for correlating whole process
lastStateChangedDate	datetime	date and time of last Lock state change

CHAPTER 30

Location

Name	Type	Description
latitude	number	location latitude of your lock
longitude	number	location longitude of your lock

CHAPTER 31

Lock

Name	Type	Description
accessLevel	*Access level*	current user access level
beaconMajor	number	identifies and distinguishes group of beacons
beaconMinor	number	identifies individual beacon within a group of beacons assigned a major value
connectedToId	number	id of bridge that lock is connected to
created	datetime	when lock was added to user account
deviceRevision	number	current lock revision on the device
deviceSettings	*Device settings*	current device settings
id	number	lock id
isConnected	boolean	is lock connected to bridge
lockProperties	*Lock properties*	current lock status and properties
macAddress	string	lock mac address
name	string	lock name
revision	number	current lock revision in database
serialNumber	string	lock serial number
shareDetails	*Share details*	share details of current user for that device
softwareVersions	*Software version*	software versions of the device
targetDeviceRevision	number	revision info that is sent to device from backend
timeZone	string	timezone of lock
type	*Device type*	device type
userIdentity	string	lock owner identity
userSettings	*User settings*	settings of current user for that device

CHAPTER 32

Lock PIN

Name	Type	Description
alias	string	name of the pin
id	number	id of the pin

CHAPTER 33

Lock PIN created

Name	Type	Description
id	number	id of the pin

CHAPTER 34

Lock PIN details

Name	Type	Description
alias	string	name of the pin
dayEndTime	datetime	end time when pin has access to the device
dayStartTime	datetime	start time when pin has access to the device
endDate	datetime	end date when pin has access to the device
id	number	id of the pin
pin	string	value of the pin
startDate	datetime	start date when pin has access to the device
weekDays	number	week days when pin has access to the device

CHAPTER 35

Lock PIN list

Name	Type	Description
listVersion	number	version of the pin list
pins	*Lock PIN* []	list of pins

CHAPTER 36

Lock properties

Name	Type	Description
batteryLevel	number	current battery level
isCharging	boolean	is lock currently charging
state	*Lock state*	current state of the lock
stateChangeResult	number	result of last state change (0 - success, > 0 - error)
lastStateChangedDate	datetime	date and time of last state change

CHAPTER 37

Lock sync

Name	Type	Description
id	number	lock id
isConnected	boolean	is lock connected to bridge
lockProperties	*Lock properties*	current properties of the lock

CHAPTER 38

Lock updated

Name	Type	Description
id	number	lock id
revision	number	current version of lock settings in database
targetDeviceRevision	number	revision info that is sent to device from backend

CHAPTER 39

Mobile identifier

Name	Type	Description
id	number	mobile device identifier

CHAPTER 40

Mobile registered

Name	Type	Description
id	number	mobile device identifier
name	string	integration device Name
operatingSystem	*Operating system*	operating system value
userIdentity	string	user uuid identifier

CHAPTER 41

Personal access key

Name	Type	Description
id	UUID	id of the personal access key
name	string	name of the personal access key
validTo	datetime	date when key expires
scopes	string[]	table of scopes that is assigned to key
prefix	string	part of the key

CHAPTER 42

Personal access key created

Name	Type	Description
id	UUID	key id
key	string	key value

CHAPTER 43

Repeat event

Name	Type	Description
dayEndTime	datetime	end time when user has access to the device
dayStartTime	datetime	start time when user has access to the device
endDate	datetime	end date when user has access to the device
id	number	id of repeat event
startDate	datetime	start date when user has access to the device
weekDays	number	week days when user has access to the device

CHAPTER 44

Revoked certificate

Name	Type	Description
certificateId	number	id of the revoked certificate
expirationDate	dateTime	certificate expiration date

CHAPTER 45

Revoked certificate list

Name	Type	Description
deviceId	number	Tedee device identifier
lastUpdated	number	last updated date
revokedCertificates	*Revoked mobile certificates* []	revoked mobile certificates list
signature	string	signature of the revoked certificate list
version	number	version of the revoked certificate list

Note: Remember, you should validate the data arrived with the signature.

CHAPTER 46

Share details

Name	Type	Description
accessLevel	*Access level*	access level of the share
deviceId	number	id of the shared device
id	number	id of the share
isPending	boolean	if user doesn't have Tedee account the email invitation is sent. Until user create account the share is pending
userDisplay-Name	string	user display name that device is shared for
userEmail	string	user email that device is shared for
userId	number	user id that device is shared for
userIdentity	string	user identity that device is shared for
repeatEvent	*Repeat event*	repeat event of the share
remoteAccess-Disabled	boolean	is remote access disabled

CHAPTER 47

Signed time

Name	Type	Description
datetime	string	signed time as base64 string
signature	string	signature of signed time

Note: Signed time and signature can be verified by Tedee device only.

CHAPTER 48

Software versions

Name	Type	Description
softwareType	*Software type*	type of the current software
updateAvailable	boolean	is software update available
version	string	current version of the software

CHAPTER 49

User settings

Name	Type	Description
autoUnlockConfirmEnabled	boolean	is confirm to auto unlock enabled
autoUnlockEnabled	boolean	is auto unlock enabled
autoUnlockRangeIn	number	value of the in zone
autoUnlockRangeOut	number	value of the out zone
autoUnlockTimeout	number	value of auto unlock timeout
location	*Location*	geographic location used by auto unlock

CHAPTER 50

Access level

This enum describes user access level to the device

Number	Name
0	Guest
1	Admin
2	Owner
3	None

CHAPTER 51

Activity source

This enum describes possible sources of device activity

Number	Name
0	Device
1	Mobile

CHAPTER 52

Device operation type

This enum describes operation type on lock device

Number	Name
0	Lock
1	Unlock
2	Pull

CHAPTER 53

Device type

This enum describes possible device types

Number	Name
0	Nofo
1	Bridge
2	Lock

CHAPTER 54

Event type

This enum describes possible events in device activity logs

Number	Name	Description
32	LockedRemote	locked via mobile app
33	UnlockedRemote	unlocked via mobile app
34	LockedButton	locked by pressing the button on lock device
35	UnlockedButton	unlocked by pressing the button on lock device
36	LockedAuto	successfully performed auto lock feature
37	UnlockedAuto	successfully performed auto unlock feature
38	LockedManual	lock was rotated manually into locked position
39	UnlockedManual	lock was rotated manually into unlocked position
40	Jammed	lock got stuck during locking/unlocking/pulling action
41	PowerOff	lock registered long button push and it will power off
42	PowerOn	lock registered power up
43	Calibration	user successfully calibrate or recalibrate the lock
44	Dismounted	not used
45	Alarm	not used
46	BatteryCharging	device detect start charging process
47	PartiallyOpenManual	user rotated the lock from locked or unlocked position into semi-locked position
48	PartiallyOpenButton	operation of locking or unlocking performed from button failed and lock stopped in semi-lo
49	PartiallyOpenAuto	operation of auto locking failed and lock stopped in semi-locked position
50	BatteryStopCharging	device detect stop charging process
51	PulledRemote	spring was pulled via mobile app
52	PulledAuto	spring was pulled automatically after receiving auto unlock command from mobile app
53	PulledManual	lock was rotated manually to perform pull spring action
54	PartiallyOpenRemote	mobile app sent open or close request and received lock status changed to partially open
55	PulledAutoBy	mobile app sent auto unlock request when lock was already in unlocked position and only p
56	PostponedLock	locked by pressing and holding the button on lock device
57	UnlockedHomeKit	unlocked via HomeKit app
58	PartiallyOpenHomeKit	HomeKit app sent open or close request and received lock status changed to partially open

C

Table 1 – continued from previous page

Number	Name	Description
59	LockedHomeKit	locked via HomeKit app
60	PulledHomeKit	spring was pulled via HomeKit app
61	UnlockByPin	unlock from keypad by pin
62	IncorrectPin	incorrect pin typen on keypad
63	PullSpringByPin	keypad sent unlock request when pull spring is enabled and lock was open and only pull spr
64	PartiallyOpenByPin	keypad sent unlock request and received lock status changed to partially open
224	FirmwareUpdateByBridge	device was updated by bridge
225	FirmwareUpdateByMobile	device was updated by mobile app

CHAPTER 55

Lock state

This enum describes possible states of lock

Number	Name
0	Uncalibrated
1	Calibrating
2	Unlocked
3	SemiLocked
4	Unlocking
5	Locking
6	Locked
7	Pulled
8	Pulling
9	Unknown
18	Updating

CHAPTER 56

Operating system

This enum describes possible operating systems

Number	Name
0	iOS
1	Android
3	Other

CHAPTER 57

Software type

This enum describes possible software types

Number	Name
0	Device
1	WiFi

CHAPTER 58

Unlock mode

This enum describes possible modes for unlocking lock

Number	Name
0	Default behaviour of unlocking lock
2	Force unlock
3	Without auto pull spring
4	Unlock or pull spring

Note: Force unlock value (number 2) has find usage only when lock is in Unknown state. It is responsible for making lock to go to Unlocked state no matter what. It is force or emergency way to unlock the door. Everybody that has access to lock can use this enum value.

Note: Without auto pull spring value (number 3) allows to unlock the lock without pulling the spring (when lock has auto pull spring enabled).

Note: Unlock or pull spring value (number 4) allows to perform two operations depends on current lock state. When lock is in Locked state, it allows to unlock the lock (with pulling the spring when lock has auto pull spring enabled). When lock is in Unlocked state, it allows to perform pull spring.

CHAPTER 59

Week days

This enum describes week days (This enum is a C# flag type for more info visit Microsoft documentation).

Number	Name
1	Mondey
2	Tuesday
4	Wednesday
8	Thursday
16	Friday
32	Saturday
64	Sunday

This site contains a documentation for Tedee REST API (https://api.tedee.com). It aims to help users to automate their lock's actions or to intergrate custom solutions with Tedee API. You can find here guides and code samples which should help you to work with the API.

CHAPTER 60

About Tedee

Tedee is smart home system which allows you control access to your home directly from phone. Please visit our site for more details https://tedee.com.

CHAPTER 61

Useful links

- Forum
- Tedee API
- Documentation repository

CHAPTER 62

Need a help?

We encourage you to join a discussion or start a new one under Community forum or ask on Twiter (@tedee_smartlock).