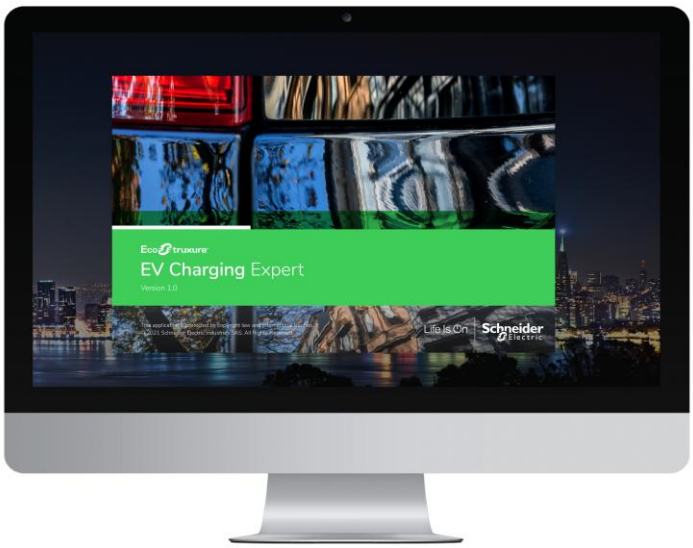
EcoStruxure™ EV Charging Expert

Rest API - Infrastructure integration

04/2026



EVINBEMS-03

Legal information

About this guide

Document scope

The purpose of this guide is to provide Rest API users' technical information necessary to manage EcoStruxure EV Charging Expert through Rest API.

Introduction

- EcoStruxure EV Charging Expert main features:
  - allocate a current setpoint to the charging stations in operation
  - see in real time the status of the charging stations through the dashboard
  - manage user authentication for charging authorization
  - get the charging sessions history and data from the charging stations in the network
- EcoStruxure EV Charging Expert is compatible with remote supervision from a Charge Point Operator in OCPP 1.6 Json.
- EcoStruxure EV Charging Expert allows two access profiles:

Admin: Access to all configuration parameters and features, dashboard operation and RFID cards management.
User: Dashboard operation and RFID cards management.

Related documents

Title of documentation	Reference number
EcoStruxure™ EV Charging Expert 6.0 - User Guide for HMIBSC	DOCA0358EN
EcoStruxure™ EV Charging Expert 6.0 - User Guide for HMIBX1	DOCA0429EN

You can download these technical publications and other technical information from our website at https://www.se/en/download

Safety information

Important information

Read these instructions carefully and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this manual or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of either symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result in** minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation and has received safety training to recognize and avoid the hazards involved.

Table of contents

Chapter 1.
Introduction

1.1Glossary

- **EV :** Electrical Vehicle
- **AC:** Alternate Current
- **DC:** Direct Current
- **VIP:** Very Important Person
- **RFID:** Radio Frequence Identification
- **CDR:** Charging Data Record
- **HMI:** Human Machine Interface
- **OCPP:** Open Charge Point Protocol
- **EBMS:** Energy Building Management System
- **RBAC:** Role Back Access Control
- **HMAC:** Hash-Based Message Authentication Code
- **SHA256:** Secure Hash Algorithm 256 bits
- **JWT:** JSON Web Token

1.2What is a REST API?

A REST API, or Representational State Transfer API, is a software architectural style that defines a set of constraints for creating web services. It's designed to be simple, scalable and efficient, making it a popular choice for building advanced web applications.

Key Characteristics of a REST API:

- **Client-Server Architecture**: A clear separation between the client and server.
- **Statelessness**: Each request from the client must contain all the information necessary to understand and process the request.
- **Cacheable**: Responses can be cached to improve performance.
- **Uniform Interface**: A consistent interface using standard HTTP methods.
- **Layered System**: Multiple layers can be used to improve modularity and scalability.

1.3 Why Use REST APIs?

REST APIs offer many advantages that make them a compelling choice for building web applications:

- **Simplicity**: It is relatively easy to understand and implement, reducing development time and effort.
- **Scalability**: It handles many requests and scales up easily to accommodate growing user bases.
- **Flexibility**: It is used with various programming languages and platforms.
- **Platform Independence**: Such APIs are accessible from any device with an internet connection.
- **Efficiency**: REST APIs are designed to be efficient, minimizing network traffic and processing time.

1.4 Core Concepts

1.4.1 Resources

A resource is any entity that can be identified and manipulated. In a REST API, resources are represented by URLs. For example, a "stations" resource might be represented by the URL / stations /test.

1.4.2 HTTP Methods

REST APIs use standard HTTP methods to perform different operations on resources:

- GET: Retrieve a resource.
- POST: Create a new resource.
- PUT: Update an existing resource.
- DELETE: Delete a resource.

1.4.3 Data Formats

REST APIs use JSON to represent data. JSON is a lightweight, human-readable format that is widely used in modern web applications.

Chapter 2.
Authentication and Authorization

2.1 Certificate management

Access by the Admin tab → Configuration → Certificate

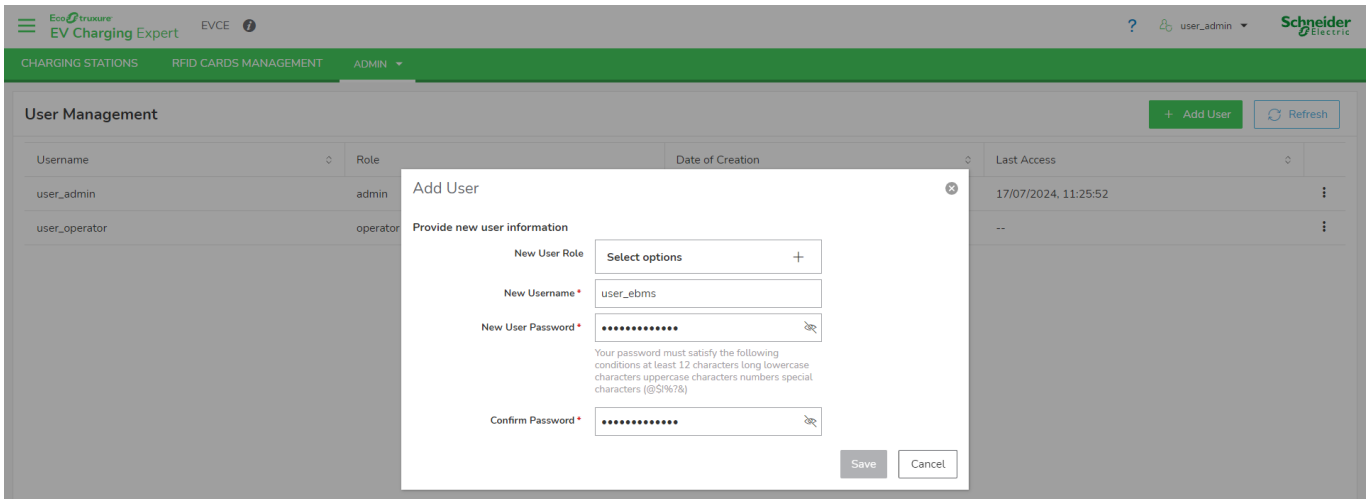
OCPP & Webserver certificates page provides EcoStruxure EV Charging Expert certificates to establish a secure communication between EcoStruxure EV Charging Expert and Rest API user:

- **Webserver certificate**: This certificate needs to be installed to establish a HTTPs connection with EcoStruxure EV Charging Expert.

2.2 User management

Access by the Admin tab → Users management

2.2.1 User registration



To create a user, click on **Add User** and fill below information:

- **New User Role**:
  - EBMS: External Building Management System, access to a subset of the REST API dedicated to energy management
- **New Username**
- **New User Password / Confirm Password**: The password must be at least 12 characters long and must contain lowercase characters, uppercase characters, numbers, special characters (@$!%?&)

2.2.2 Change password

A user can change his password anytime from Rest API by using the request POST /login/passwd.

2.3 Authentication management

The authentication is based on bearer token. The POST login/login request returns the token and using the JSON Web Tokens standard.

At startup, a random 256-bits value is generated and is used as a secret to sign the generated tokens using the HS256 algorithm (HMAC - SHA256). This ensures that tokens that were issued prior to this generation (before a reboot for example) are automatically invalidated.

The JWT header contains the following fields:

```
{
  "alg": "HS256", // Signing algorithm
  "typ": "JWT"    // Indicate that it is a JWT
}
```

The JWT payload contains the following fields:

```
{
    "jti": 123,          // Unique id of the token
    "exp": 123456789,    // Expiration time (UNIX timestamp - UTC)
    "sub": "my_user",    // User for which the token has been issued
    "role": "BEMS"    // Role of the user
}
```

Important to notice: The period of validity encoded in the expiration time field of the JWT is set to **12 hours**. **If no activity is detected (no request received) for 30 minutes** the JWT token will be revoked, and the user will have to log in again.

The JWT must be transferred by the user agent through the Authorization header field of the HTTP request using the Bearer schema. The content of the header should look like the following:

Authorization: Bearer <base64 encoded token>

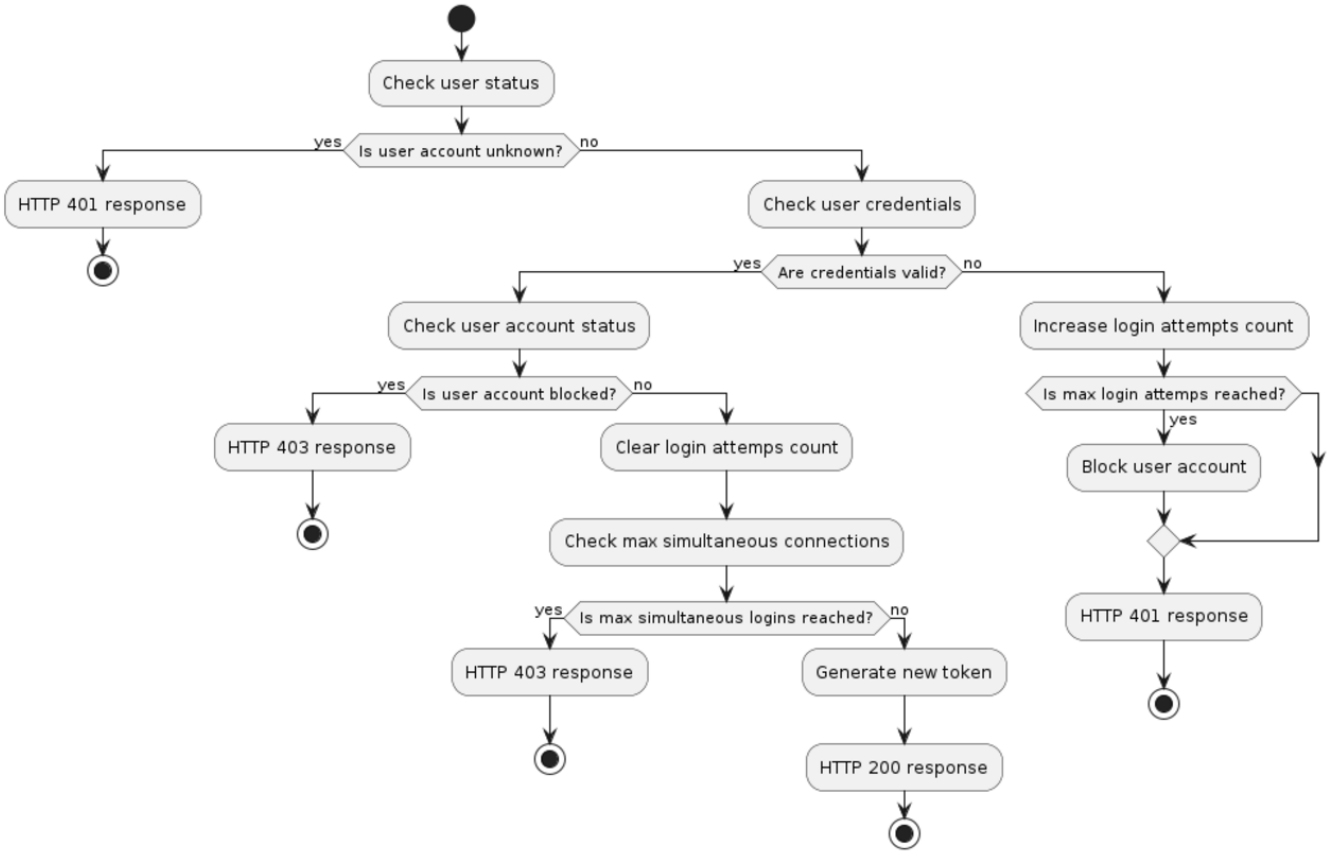
The JWT tokens are also automatically revoked for specific user on following events:

- User password modification
- User deletion

2.4 Authorization

2.4.1 Process

Bearer token is returned only when authentication is valid. The authorization process is represented in below diagram. Each user has a role and corresponding permissions:



A bearer token is available only for one connection and each user can open only one connection at a time.

2.4.2 EBMS permissions

Methods	URI	Description
User access		
POST	/api/v2/login/login	User login operation to be able to use the API
POST	/api/v2/login/logout	User logout operation to release its API token
POST	/api/v2/login/passwd	Update the password of an authenticated user
PUT	/api/v2/login/token-policy	Update the connection token policy
GET	/api/v2/login/token-policy	Get the connection token policy
Station management		
GET	/api/v2/stations/	Get the list of the charging stations
GET	/api/v2/stations/statuses	Get the status of the charging stations
GET	/api/v2/stations/models	Get the list of the supported charging stations models
GET	/api/v2/stations/firmwares	Get the list of the stored charging stations firmwares
POST	/api/v2/stations/firmwares	Request the upload of a new charging station firmware
GET	/api/v2/stations/firmwares/{firmware_id}	Get the properties of a firmware
GET	/api/v2/stations/diagnostics	Get the list of the stored diagnostics reports
POST	/api/v2/stations/diagnostics/export	Export the stored diagnostics reports as a single archive file
GET	/api/v2/stations/{station_id}	Get the properties of a charging station
PUT	/api/v2/stations/{station_id}	Update a charging station configuration
GET	/api/v2/stations/{station_id}/status	Get the status of a charging station
GET	/api/v2/stations/{station_id}/firmware	Get the stored firmware for the charging station
GET	/api/v2/stations/{station_id}/diagnostics	Get the list of the stored diagnostics reports of a charging station
POST	/api/v2/stations/{station_id}/diagnostics	Trigger the generation of a diagnostic report on a charging station

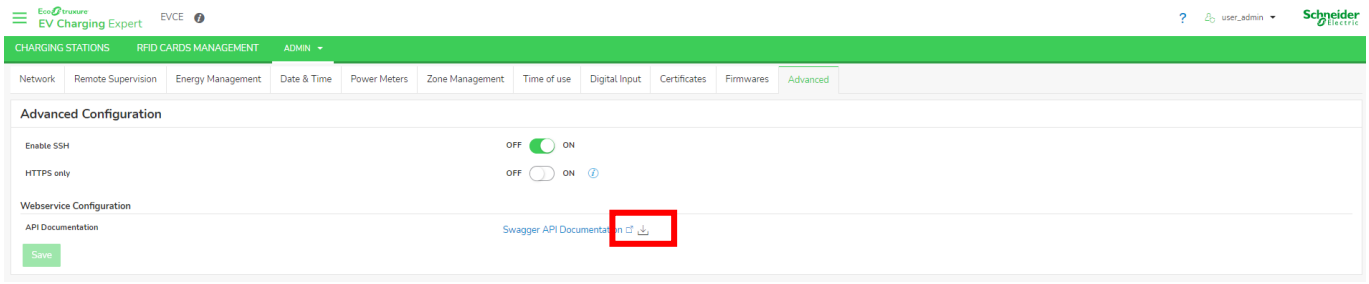
Methods	URI	Description
GET	/api/v2/stations/{station_id}/cmds/status/{cmd_id}	Get the status of a command sent to a charging station
POST	/api/v2/stations/{station_id}/cmds/remote_start	Send a remote start command to a charging station
POST	/api/v2/stations/{station_id}/cmds/remote_stop	Send a remote stop command to a charging station
POST	/api/v2/stations/{station_id}/cmds/reset	Send a reset command to a charging station
POST	/api/v2/stations/{station_id}/cmds/change_availability	Change the availability of a charging station
POST	/api/v2/stations/{station_id}/cmds/get_configuration	Send a get configuration command to a charging station
POST	/api/v2/stations/{station_id}/cmds/get_diagnostics	Send a get diagnostics command to a charging station
POST	/api/v2/stations/{station_id}/cmds/update_firmware	Update the firmware of a charging station
	Energy management	
GET	/api/v2/em/zones	Get the list of the zones
GET	/api/v2/em/zones/{zone_id}	Get the properties of a zone
PUT	/api/v2/em/zones/{zone_id}	Update a zone properties
GET	/api/v2/em/zones/{zone_id}/status	Get the status of a zone
GET	/api/v2/em/zones/{zone_id}/reduction	Get the reduction applied to a zone
PUT	/api/v2/em/zones/{zone_id}/reduction	Set the reduction to apply to a zone
GET	/api/v2/em/zones/{zone_id}/stations	Get the list of stations affected to the zone
	Power meter	
GET	/api/v2/pms	Get the list of the power meters
GET	/api/v2/pms/{pm_id}	Get the properties of a power
PUT	/api/v2/pms/{pm_id}	Update a power meter properties
GET	/api/v2/pms/{pm_id}/values	Get the last values read from a power meter
GET	/api/v2/pms/models	Get the list of the power meter models
GET	/api/v2/pms/models/{pm_model_id}	Get the properties of a power meter model
PUT	/api/v2/pms/models/{pm_model_id}	Update a power meter model properties
	Configuration	
GET	/api/v2/config/em	Get the configuration of the energy management functionalities
	Maintenance	
GET	/api/v2/maintenance/product_caps	Get Product capacities
GET	/api/v2/maintenance/product_status	Get operational status of the product

Chapter 3.
PRATICAL API USAGE

3.1 Practical API Usage

3.1.1 Open API file

All requests are documented into a swagger file. It is possible to download it in Admin/Advanced menu available only for administrator:



The file is Open API format which allows direct usage in API management software. Automatic generation of libraries in the desired programming language is also possible.

3.1.2 URL Base

The URL base to access the EcoStruxure EV Charging Expert API starts with:

https://DEVICE_IP_ADDRESS:443/

Where DEVICE_IP_ADDRESS is the IP address of the product (by default 192.168.0.128).

All endpoints described in this document are accessed by appending the endpoint path to this URL base. For example:

- To retrieve a list of stations: GET https://192.168.0.128:443/api/v2/stations/

3.1.3 Status code

A status code is return for each request:

Status code	Comment
200	The request was successful
400	The request was malformed or invalid
401	Access token is missing or invalid
403	User does not have the permissions to access the resource
404	The requested resource doesn't exist

3.1.4 Unique ID for every resource

In EcoStruxure EV Charging Expert, each resource is identified with a unique tag called “**ID**”. List of resources with unique ID:

- Firmware

- Station
- Zone
- Power meter

This unique ID is a crucial parameter that must be included in all requests while interacting with specific objects. For instance, to retrieve, update, or delete a resource, its unique ID must be mentioned as part of the request. This ensures that the API knows exactly which resource is being referred.

Example:

- GET /api/v2/stations/{station_id}/status: Retrieves the status of the station specified with the ID.

Internally to EV Charging Expert, every time a new resource is added to the system, a unique ID is generated to identify it. This ID is always incremented from the last used ID, ensuring that each ID is unique and never reused. Even if a resource is deleted, its ID is no more available for a future allocation. This approach ensures that IDs remain unique across the entire lifecycle of the product.

Example:

1. Adding a new station: When a new station is added, the system generates a new ID by incrementing the last used ID. For example, if the last ID was 100, the new resource will be assigned the ID 101.
2. Deleting a station: If a resource is deleted, its ID is not reused. This prevents any confusion or overlap in ID assignment.
3. Replacing a station: In order to replace a station, the old charger is first deleted from EV Charging Expert configuration. When the replacement charger is added to EV Charging Expert, a new ID is incrementally assigned.

This method ensures that each ID remains unique and traceable. Once an ID is generated and used, it cannot be reassigned to another resource, ensuring data integrity across the product.

Important to notice: In case of a configuration import, all resources are assigned a new unique ID from the last available increment.

To efficiently manage the correspondence between a specific resource and its unique ID, it is highly recommended to create a dictionary that maps each zone or station name to its unique ID and vice versa. To build such a dictionary, it is necessary to retrieve a list of all resources (stations, zones …) by using the resource primary request. All dictionary information are available in the return dataset (names of the resources and their unique IDs). It is recommended to update dictionary after a reconnexion to detect modifications.

3.2 Practical examples

All examples are using a Rest API client generated by swagger codegen for Python3.

3.2.1 Login

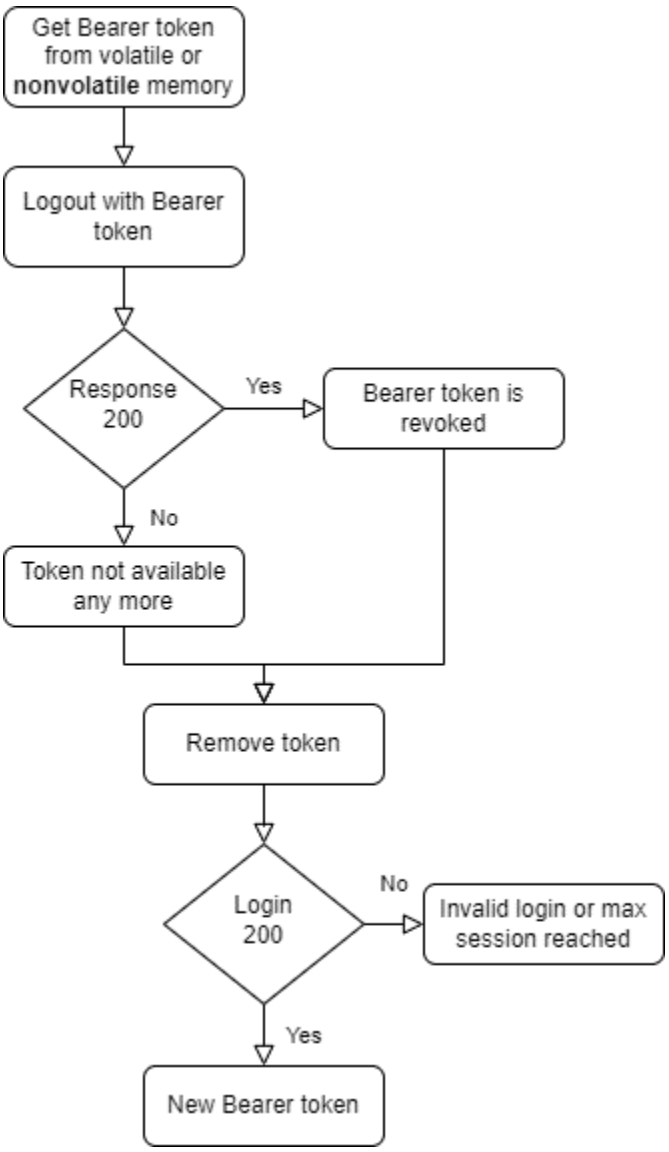
This example shows how to perform a login operation and how to use a Bearer token to manage an EV Charging Expert:

```
from swagger_client.api_client import ApiClient
from swagger_client.configuration import  Configuration
from swagger_client.api.login_api import LoginApi
from swagger_client.models.user_standard_login import UserStandardLogin

class Evce():
    def __init__(self, ip: str = "192.168.0.128", port: str = "443"):
        self.ip = ip
```

```
        self.port = port
        self.bearer_token = ''

        url = f"https://{ip}:{port}/api/v2"

        config = Configuration()
        config.host = url

        self.api_client = ApiClient(config)
        self.login = LoginApi(self.api_client)
        self.stations = StationsApi(self.api_client)


    def connect(self, login: str, password: str):
        try:
            self.login.login_logout_post()
        except Exception as e:
            raise e
        finally:
            self.login.api_client.default_headers.pop('Authorization', None)

        body = UserStandardLogin(login=login, password=password)
        response = self.login.login_login_post_with_http_info(body=body)
        if response[1] == 200:
            token = response[0].token
            self.api_client.set_default_header(
                'Authorization', 'Bearer ' + token)

        return response

    def disconnect(self):
        response = self.login.login_logout_post_with_http_info()

        return response

if __name__ == '__main__':
      evce=Evce()
      try:
            evce.connect(evce_login, evce_password)
      except Exception as e:
            error = json.loads(e.body)
            printf(f'Fail to login: {error["message"]}')
```

Connect method is here to check if a token is already in volatile memory and perform a logout to revoke it safely from the Header. Then a new login request is performed (return code “200” is successful). The new Bearer token is stored and used for all further requests.

To prevent token loss in case of a client application reboot, it is recommended to store Bearer token into nonvolatile memory. Try to perform a logout with this token and revoke it before next login.



Sequence diagram - Login

Example of HTTP header without Bearer token:

```
Content-Type: application/json
Accept: application/json
User-Agent: PostmanRuntime/7.42.0
Cache-Control: no-cache
Host: 10.195.130.5:8287
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 58
```

Example of HTTP header with Bearer token from a previous login:

```
Authorization: Bearer
eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJleHAiOjE3MzM3ODY3OTYsImp0aSI6NCwicm9sZXMiOlsiYWRtaW4iXSwic3
ViIjoidXNlcl9hZG1pbiJ9.kNPFitqHpwQqfKE2rqIAhjBMCGb6gV41SpNRXeA-Cio
User-Agent: PostmanRuntime/7.42.0
Accept: */*
Cache-Control: no-cache
Host: 10.195.130.5:8287
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 0
```

3.2.2 Unique id management

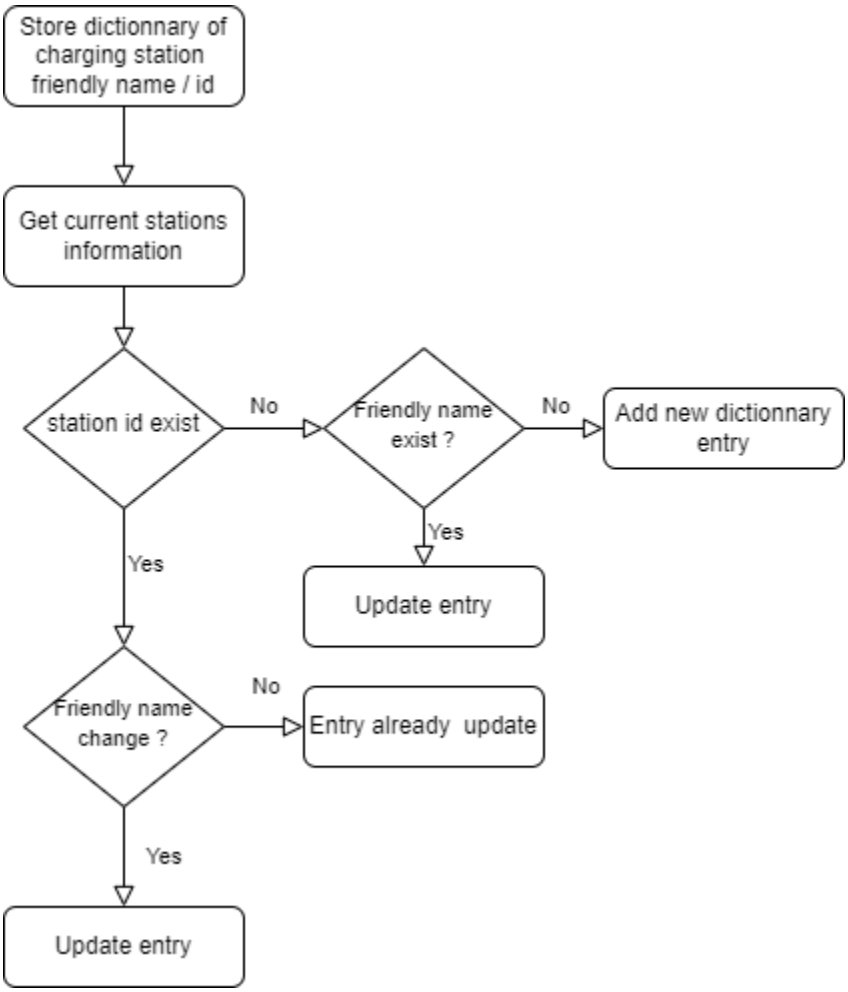
This example shows how to manage a resource specified by a unique ID in EV Charging Expert. Below example for a charging station (also applicable to zones and power meters in the same way):

```
from api.evce import Evce

def update_id(stations: dict, station_dictionnary: dict):
    # Dictionnary generated
    new_station_dictionnary = {}
    # Invert dictionnary
    invert_dictionnary = {value: key for key, value in station_dictionnary.items()}
    # For each charging station check the entry
    for station in stations:
       if station.id in station_dictionnary.values():
            if invert_dictionnary[station.id] != station.config.friendly_name:
                # Update friendly of id
                new_station_dictionnary[station.config.friendly_name] = station.id
                print(f'Update friendly name of charging station id {station.id} by
{station.config.friendly_name}')
            else:
                # Same dictionnary entry
                new_station_dictionnary[station.config.friendly_name] = station.id
                print(f'Charging station {station.config.friendly_name} has same id
{station.id}')
        else:
            if station.config.friendly_name in station_dictionnary:
                # Update id of charging station
                new_station_dictionnary[station.config.friendly_name] = station.id
                print(f'Charging station {station.config.friendly_name} update id
from {station_dictionnary[station.config.friendly_name]} to {station.id}')
            else:
                # Add new dictionnary entry
                new_station_dictionnary[station.config.friendly_name] = station.id
                print(f'New charging station {station.config.friendly_name} entry
with id {station.id}')

    return new_station_dictionnary

if __name__ == '__main__':
    # Generate dictionnary key: friendly name and value: unique id
    station_dictionnary={}

    # EcoStruxure EV Charging Expert object
    evce=Evce()

    # Try to get information from charging station through its friendly name
    try:
        # Connection to EcoStruxure EV Charging Expert
        evce.connect("user_administrator", "P&ssw0rdAdmin")
```

```
        # Get all stations information
        stations=evce.stations.stations_get()
        # Update charging station dictionnary after each connection
        station_dictionnary = update_id(stations, station_dictionnary)

        # Station name present into EcoStruxure EV Charging Expert
        station_name = "test_evce"

        # Get charging station status with friendly name of the station
        station_status =
evce.stations.stations_station_id_status_get(station_id=station_dictionnary[station_n
ame])
        print(f'Charging station with name {station.config.friendly_name} has status
{station_status.status}')

    except Exception as e:
        error = json.loads(e.body)
        summary_of_failed_operation.append((evce_ip_address, f'EVCE:
{error["message"]}'))
    finally:
        evce.disconnect()
```

To manage unique ID for charging station it is recommended to use friendly name and create a dictionary of charging station with key: friendly name and value: unique id. After each reconnection it is recommended to update charging station dictionary to check installation updates.

To find new charging station or update information, follow the diagram below:



Sequence diagram – Update charging station information

3.3 Tools

It is possible to use standard tool to manage EV Charging Expert through swagger:

- EV Charging swagger interface
- Postman: import open API file
- Insomnia: import open API file

There is also a lot of possibilities to manage Rest API through different languages libraries:

- Python (Requests, aiohttp)
- C++ (libcurl …)
- JavaScript (Fetch API, Axios …)

Schneider Electric Industries SAS
35, rue Joseph Monier
CS 30323
92506 Rueil Malmaison Cedex
France

www.se.com