EET SolMate API Documentation

Communication Technology: Secure Websockets (WSS)
Encryption: TLSv1.2
Request and Response format: JSON Objects
DateTime RFC-defined format: `%Y-%m-%dT%H:%M:%S`
Endpoint: *wss://sol.eet.energy:9124/* or locally on SolMate: *ws://[ip]:9124/*
Port: 9124
Sub-protocol: Not specifically defined, "chat" and "superchat" as the most common ones will work, confer RFC definition of Sec-WebSocket-Protocol header: https://tools.ietf.org/html/rfc6455#section-4.1. Best practice is leaving the choice to the browser-server combination as security standards evolve.

The protocol intends for requests to go both ways, as the connection should be kept alive at all times (reconnect on connection loss).

Format definition

As soon as the connection is opened, requests can be sent in the following format.
Each request is accompanied by a "route" which corresponds to an API method, and data sent along with it as parameters to the API method.

Request format (raw sent bytes):

```
{
    "route": "e.g. live_values",
    "id": 1234, // Simple counter from 0, used to re-identify the response to the respective request
    "data": {...} // Data passed to the api method
}
```

Response format

```
{
    "id": 1234,
    "data": {...}
}
```

Error response format

```
{
    "id": 1234,
    "error": {
        "type": "SolmateNotConnected",
        "message": "The Solmate identified by 123-456 is not connected to our
server. Live values cannot be fetched."
    }
}
```

Redirect response format

When authenticating on the sol server, you might get redirected by the load balancer.

```
{
    "id": 1234,
    "redirect": "wss://sol.eet.energy:9125/"
}
```

Example Implementations

Example implementation for handling requests/responses using Python 3.5 with the websockets package (`pip install websockets`):

```
#!/usr/bin/env python
import asyncio
import time
import websockets
import base64
import hashlib
import json


class Connection:
    server_uri = "wss://sol.eet.energy:9124/"   # The endpoint
    websocket = None                            # Websocket will be stored here
    message_id = 0                              # Continous message id

    @staticmethod
    async def create_socket():
        """
        Create a websocket and connect it to the endpoint.
        """
        Connection.websocket = await websockets.connect(Connection.server_uri)
        print("Websocket is connected to: " + Connection.server_uri)

    @staticmethod
    async def send_api_request(data):
```

```
        """
        Send an api request with the given data and return response data.
        """
        Connection.message_id += 1
        await Connection.websocket.send(json.dumps(data))
        response = json.loads(await Connection.websocket.recv())

        if "data" in response:
            return response["data"]
        elif "error" in response:
            raise Exception(str(response["error"]))
        else:
            raise Exception("Response did not contain data.")


def authenticate(serial_number, password, device_id):
    """
    Authenticate on the server with the given serial number, password and device
id
    """

    # Create and connect websocket
    asyncio.get_event_loop().run_until_complete(Connection.create_socket())

    # Get the response to the request of the login route with the login data
    response =
asyncio.get_event_loop().run_until_complete(Connection.send_api_request(
        {
            "id": Connection.message_id,
            "route": "login",
            "data": {
                "serial_num": serial_number,
                "user_password_hash":
base64.encodebytes(hashlib.sha256(password.encode()).digest()).decode(),
                "device_id": device_id
            }
        }
    ))

    # If the login was successful
    if "success" in response and response["success"] and "signature" in response:
        # Get the signature for the session
        signature = response["signature"]

        # Check if Server redirects to another instance (on first connect the
connection to a load-balancer is established)
        correct_server = False
        while not correct_server:
            # Get the response to the authentication route with the
authentication data
            response =
asyncio.get_event_loop().run_until_complete(Connection.send_api_request(
                {
                    "id": Connection.message_id,
                    "route": "authenticate",
                    "data": {
                        "serial_num": serial_number,
                        "signature": signature,
```

```
                        "device_id": device_id
                    }
                }
            ))

            # Handle load-balancer redirect, if there is one
            if "redirect" in response and response["redirect"] is not None:
                print("Got redirected to: " + str(response["redirect"]))
                Connection.server_uri = response["redirect"]

 asyncio.get_event_loop().run_until_complete(Connection.create_socket())
            else:
                # When there is no redirect parameter, the socket is connected to
the correct instance
                correct_server = True

        # return authentication response
        return response
    return {"success": False}


def example_request(route, data):
    """
    Send a request and print the response.
    """
    response =
asyncio.get_event_loop().run_until_complete(Connection.send_api_request(
        {"id": Connection.message_id, "route": route, "data": data}
    ))
    print(route + ": " + str(response))


def main():
    # Your SolMate login data
    serial_number = "SOLMATE_SERIAL_NUMBER"     # The SolMate Serial Number
    password = "SOLMATE_PASSWORD"               # The SolMate Password
    device_id = "smart_home"                    # Whatever you want

    # Authentication
    print("Authenticating...")
    response = authenticate(serial_number, password, device_id)
    if "success" in response and response["success"]:
        print("Authentication successful!")

        # If SolMate is online
        if "online" in response and response["online"]:
            print("SolMate is online.")
            time.sleep(1)
            # Make example requests
            example_request("get_api_info", {})
            example_request("get_solmate_info", {})
            example_request("get_user_settings", {})
            example_request("get_grid_mode", {})
            example_request("get_injection_settings", {})
            # Print live values forever every 10 seconds
            while True:
                try:
                    example_request("live_values", {})
```

```
                except websockets.exceptions.ConnectionClosedOK:
                    raise ConnectionError("Connection closed.")
                time.sleep(10)
        else:
            # If SolMate is not online
            print("Your SolMate is offline.")
    else:
        # If authentication was not successful
        print("Authentication failed.")


if __name__ == "__main__":
    main()
```

API info

For a list of available API routes, authenticate on the API and send a request to "api_info". So in the example above it would be:

```
example_request("api_info", {})
```

This will show a lot of information about the API.

Help

If you encounter problems regarding the API feel free to contact:

felix@eet.energy