**AllSense IOT mqtt 통신을 위한 SerialData pub**

소스코드 구조설명서

**2024.** **09**

**㈜쿠노소프트**

- 목차 –

1. [main.js 4](#_bookmark0)
   1. [파일 설명 4](#_bookmark1)
   2. [createWindow 4](#_bookmark2)
   3. [connectSerialPort 4](#_bookmark3)
   4. [app 5](#_bookmark4)
   5. [ipcmain 5](#_bookmark5)
2. [preload.1.js 16](#_bookmark31)
   1. [파일 설명 16](#_bookmark32)
   2. [preload전문 16](#_bookmark33)
3. [renderer.js 19](#_bookmark35)
   1. [파일 설명 19](#_bookmark36)
   2. [onSerial 19](#_bookmark37)
   3. [getElementById 19](#_bookmark38)
4. [index.html 21](#_bookmark42)
   1. [파일 설명 21](#_bookmark43)
5. [package.json 22](#_bookmark44)
   1. [파일 설명 22](#_bookmark45)

# main.js

## 파일 설명

Electron의 서버를 담당하는 main 코드 mqtt서버, Serial Port와 연결하고 포트로 전달되는 값을 파싱하여 data Object에 저장하고 연결된 mqtt 서버에 publish한다. 특정 event마다 preload 된 콜백함수에 따라 Renderer로 값을 전송한다.

## createWindow

: 실행시 화면을 표시할 윈도우창 생성

function createWindow() {

    win = new BrowserWindow({

        width: 800,

        height: 600,

        webPreferences: {

            preload: path.join(\_\_dirname, 'preload.js'),

            contextIsolation: true,

            enableRemoteModule: false,

        }

    });

    win.loadFile('index.html');

}

## connectSerialPort

: 시리얼포트와 연결 및 mqtt 서버 publish 구현

async function connectSerialPort() {

*try* {

        const ports = *await* SerialPort.list();

        const targetVendorId1 = "1A86"; *// 찾고자 하는 vendorId*

        const targetVendorId2 = "1a86"; *// 찾고자 하는 vendorId*

        const targetProductId = "7523"; *// 찾고자 하는 productId*

        const connectedPort = ports.find(*port* => (*port*.vendorId === targetVendorId1 || *port*.vendorId === targetVendorId2) && *port*.productId === targetProductId);

*if* (connectedPort) {

            const port = new SerialPort({

                path: connectedPort.path,

                baudRate: 115200,

                autoOpen: false

            });

*await* new Promise((*resolve*, *reject*) => {

                port.open((*error*) => {

*if*(open == 1){

                        console.log("포트가 이미 연결되어 있습니다.");

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                        win.webContents.send('serial-data', "포트가 이미 연결되어 있습니다.");

*return* resolve(port); *// 함수를 종료하고 현재 포트를 반환*

                    }

*if* (*error*) {

                        sendData = "연결된 포트 없음"; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                        win.webContents.send('serial-data', sendData);

                        sendData = "연결된 포트 없음"; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                        win.webContents.send('serial-data', *error*);

                        console.log("non connectied port");

                        console.log(*error*)

                        reject(*error*);

                    } *else* {

                        sendData = "시리얼 포트 연결됨"; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                        win.webContents.send('serial-data', sendData);

                        console.log("port sink on");

                        open = 1;

                        resolve(port);

                    }

                });

            });

            const parser = port.pipe(new ReadlineParser());

*// MQTT 브로커에 연결합니다.*

            const client = mqtt.connect('mqtt://203.251.137.136');

            client.on('connect', function () {

                sendData = "MQTT 클라이언트 연결됨 데이터 송신 중..."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                win.webContents.send('serial-data', sendData);

                console.log('MQTT client on');

                parser.on('data', function (*data*) {

                    let value = *data*.trim().split('-');

                    value[0] = value[0].trimEnd();

*if* (value[0] == "BH1750" || value[0] == "PM2008") {

*return*; *// 무시*

                    } *else* *if* (value[0].trim().split(':')[0] == "## SEND\_BT\_SENSORF2") {

*return*; *// 무시*

                    }

                    let newKey = value[0];

                    let newValue = value[1];

*if* (newKey == "CH2O") {

                        newValue = newValue.trim().split(" ");

                        dataBuffer[newKey] = Number(newValue[0]); *// 업데이트*

                    } *else* *if* (newKey == "VOC" || newKey == "MQ131") {

                        dataBuffer[newKey] = Number(newValue); *// 업데이트*

                    } *else* *if* (newKey == "SHT41") {

                        let newValue1 = newValue.trim().split('\t');

                        let newValue2 = newValue1[0].trim().split(':');

                        let temperature = Number(newValue2[1]);

                        newValue2 = newValue1[1].trim().split(':');

                        let humidity = Number(newValue2[1]);

                        dataBuffer.Temperature = temperature; *// 업데이트*

                        dataBuffer.Humidity = humidity; *// 업데이트*

                    } *else* *if* (newKey == "SCD40") {

                        let newValue1 = newValue.trim().split('\t');

                        let co2 = Number(newValue1[0].trim().split(':')[1]);

                        dataBuffer.CO2 = co2; *// 업데이트*

                    } *else* *if* (newKey == "MICS") {

                        let newValue1 = newValue.trim().split(':');

                        newValue1[0] = newValue1[0].trim();

*if*(newValue1[1] < 30000){

*if* (newValue1[0] == "C3H8"||newValue1[0] == "C4H10"||newValue1[0] == "CH4"){

*if*(newValue1[1] > 3000){

                                    dataBuffer[newValue1[0]] = Number(newValue1[1]); *// 업데이트*

                                }

                            }*else* *if*(newValue1[1] !=0){

                                dataBuffer[newValue1[0]] = Number(newValue1[1]); *// 업데이트*

                            }

                        }

                    }

                });

*// 0.5초마다 수집된 데이터 publish*

                setInterval(() => {

*if* (Object.keys(dataBuffer).length > 0) { *// 데이터가 있을 경우에만*

                        client.publish(topic, JSON.stringify(dataBuffer));

                        console.log(`Sent data to MQTT: ${JSON.stringify(dataBuffer)}`);

                    }

                }, 500);

            });

*return* port;

        } *else* {

            sendData = "연결된 포트가 없습니다 포트를 연결하고 재실행 해주세요."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

            win.webContents.send('serial-data', sendData);

            console.log('No matching serial ports found.');

*return* null;

        }

    } *catch* (error) {

        sendData = "시리얼 포트와 연결 실패."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

        win.webContents.send('serial-data', sendData);

        console.error('Error connecting to serial port:', error);

*return* null;

    }

}

            const parser = port.pipe(new ReadlineParser());

*// MQTT 브로커에 연결합니다.*

            const client = mqtt.connect('mqtt://203.251.137.136');

            client.on('connect', function () {

                sendData = "MQTT 클라이언트 연결됨 데이터 송신 중..."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

                win.webContents.send('serial-data', sendData);

                console.log('MQTT client on');

                parser.on('data', function (*data*) {

                    let value = *data*.trim().split('-');

                    value[0] = value[0].trimEnd();

*if* (value[0] == "BH1750" || value[0] == "PM2008") {

*return*; *// 무시*

                    } *else* *if* (value[0].trim().split(':')[0] == "## SEND\_BT\_SENSORF2") {

*return*; *// 무시*

                    }

                    let newKey = value[0];

                    let newValue = value[1];

*if* (newKey == "CH2O") {

                        newValue = newValue.trim().split(" ");

                        dataBuffer[newKey] = Number(newValue[0]); *// 업데이트*

                    } *else* *if* (newKey == "VOC" || newKey == "MQ131") {

                        dataBuffer[newKey] = Number(newValue); *// 업데이트*

                    } *else* *if* (newKey == "SHT41") {

                        let newValue1 = newValue.trim().split('\t');

                        let newValue2 = newValue1[0].trim().split(':');

                        let temperature = Number(newValue2[1]);

                        newValue2 = newValue1[1].trim().split(':');

                        let humidity = Number(newValue2[1]);

                        dataBuffer.Temperature = temperature; *// 업데이트*

                        dataBuffer.Humidity = humidity; *// 업데이트*

                    } *else* *if* (newKey == "SCD40") {

                        let newValue1 = newValue.trim().split('\t');

                        let co2 = Number(newValue1[0].trim().split(':')[1]);

                        dataBuffer.CO2 = co2; *// 업데이트*

                    } *else* *if* (newKey == "MICS") {

                        let newValue1 = newValue.trim().split(':');

                        newValue1[0] = newValue1[0].trim();

*if*(newValue1[1] < 30000){

*if* (newValue1[0] == "C3H8"||newValue1[0] == "C4H10"||newValue1[0] == "CH4"){

*if*(newValue1[1] > 3000){

                                    dataBuffer[newValue1[0]] = Number(newValue1[1]); *// 업데이트*

                                }

                            }*else* *if*(newValue1[1] !=0){

                                dataBuffer[newValue1[0]] = Number(newValue1[1]); *// 업데이트*

                            }

                        }

                    }

                });

*// 0.5초마다 수집된 데이터 publish*

                setInterval(() => {

*if* (Object.keys(dataBuffer).length > 0) { *// 데이터가 있을 경우에만*

                        client.publish(topic, JSON.stringify(dataBuffer));

                        console.log(`Sent data to MQTT: ${JSON.stringify(dataBuffer)}`);

                    }

                }, 500);

            });

*return* port;

        } *else* {

            sendData = "연결된 포트가 없습니다 포트를 연결하고 재실행 해주세요."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

            win.webContents.send('serial-data', sendData);

            console.log('No matching serial ports found.');

*return* null;

        }

    } *catch* (error) {

        sendData = "시리얼 포트와 연결 실패."; *// 예시 데이터*

*// 렌더러 프로세스에 'serial-data' 이벤트 송신*

        win.webContents.send('serial-data', sendData);

        console.error('Error connecting to serial port:', error);

*return* null;

    }

}

## app

: electron 실행

app.whenReady().then(() => {

    createWindow();

    app.on('activate', () => {

*if* (BrowserWindow.getAllWindows().length === 0) createWindow();

    });

});

app.on('window-all-closed', () => {

*if* (process.platform !== 'darwin') app.quit();

});

## ipcMain

: renderer와 데이터 통신 구현

ipcMain.on('checkOff', async (*event*) => {

*try* {

*if*(sendData != "MQTT 클라이언트 연결됨 데이터 송신 중..."){

*event*.sender.send('serial-status', '연결을 재점검 해주세요');

        }*else* *if*(dataBuffer.status == 1){

            dataBuffer.status = 0;

            console.log("check end");

*event*.sender.send('serial-status', '측정 종료');

        }*else* *if*(dataBuffer.status == 0){

            console.log("check already end");

*event*.sender.send('serial-status', '측정하지 않는 중...');

        }

    } *catch* (error) {

*event*.sender.send('serial-status', '메인과 통신 실패');

        console.error('Error connecting to serial port:', error);

    }

});

ipcMain.on('submit', async (*event*,*text*) => {

*try* {

        topic = *text*

*event*.sender.send('serial-status', 'topic 설정 완료 mqtt 연결 시도중...');

        console.log("connect error with main : topic")

        setTimeout(connectSerialPort, 500);

    }*catch* (error) {

*event*.sender.send('serial-status', 'topic 설정 실패');

        console.error('Error connecting to main:', error);

    }

});

# preload.js

## 파일 설명

Renderer와 main이 통신할 수 있도록 실행 이전에 선언하는 콜백함수

* 1. Preload 전문

: check,submit render에서 Main으로 onSerial mian에서 renderer로

const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electron', {

    checkOn: () => ipcRenderer.send('checkOn'),

    checkOff: () => ipcRenderer.send('checkOff'),

    submit: (*text*) => ipcRenderer.send('submit',*text*),

    onSerialData: (*callback*) => ipcRenderer.on('serial-data', (*event*, *data*) => callback(*data*)),

    onSerialStatus: (*callback*) => ipcRenderer.on('serial-status', (*event*, *status*) => callback(*status*))

});

# renderer.js

## 파일 설명

Main에서 preload를 통해 전달받은 값을 index에 index에서 받은 값을 main으로 보낸다

## onSerial

: main -> index

window.electron.onSerialData((*data*) => {

    console.log('수신된 데이터:', *data*);

*// 새로운 데이터 항목을 생성*

    const newDataElement = document.createElement('p');

    newDataElement.textContent = `${*data*}`;

*// 스크롤 박스에 추가*

    scrollBox.appendChild(newDataElement);

*// 스크롤 박스의 스크롤을 가장 아래로 이동*

    scrollBox.scrollTop = scrollBox.scrollHeight;

});

*// 시리얼 포트 상태 수신 이벤트*

window.electron.onSerialStatus((*status*) => {

    console.log('메인과 연결 상태:', *status*);

*// 상태 메시지를 스크롤 박스에 추가*

    const statusElement = document.createElement('p');

    statusElement.textContent = `${*status*}`;

*// 스크롤 박스에 추가*

    scrollBox.appendChild(statusElement);

*// 스크롤 박스의 스크롤을 가장 아래로 이동*

    scrollBox.scrollTop = scrollBox.scrollHeight;

});

## getElementById

: index -> main

document.getElementById('startButton').addEventListener('click', () => {

    window.electron.checkOn();

});

document.getElementById('endButton').addEventListener('click', () => {

    window.electron.checkOff();

});

document.getElementById("submitButton").addEventListener("click", function() {

    const textValue = document.getElementById("textInput").value;

    window.electron.submit(textValue);

});

# index.html

## 파일 설명

Electron 실행시 표시되는 페이지를 구성할 html파일

<!DOCTYPE html>

<html lang="ko">

<head>

    <meta charset="UTF-8">

    <title>호흡기 단말장치</title>

    <style>

        body {

            display: flex;

            flex-direction: column;

            align-items: center;

            justify-content: center;

            height: 100vh; */\* 전체 화면 높이 \*/*

            margin: 0; */\* 기본 마진 제거 \*/*

        }

        #scrollBox {

            width: 550px; */\* 스크롤 박스 너비 \*/*

            height: 300px; */\* 스크롤 박스 높이 \*/*

            overflow-y: auto; */\* 세로 스크롤 가능 \*/*

            border: 1px solid #ccc; */\* 테두리 \*/*

            margin-bottom: 20px; */\* 버튼과의 간격 \*/*

        }

        #buttonContainer {

            display: flex;

            flex-direction: index;

            align-items: center; */\* 버튼 가운데 정렬 \*/*

        }

        button {

            margin: 5px; */\* 버튼 간격 \*/*

        }

    </style>

</head>

<body>

    <h1>호흡기 단말장치</h1>

    <div id="scrollBox">

        <p>전송할 topic을 입력해 주세요</p>

*<!-- 스크롤 박스 내용 (예시) -->*

    </div>

    <div id="buttonContainer">

        <input type="text" id="textInput" placeholder="topic을 입력하세요">

        <button id="submitButton">전달하기</button>

        <button id="startButton">측정 시작</button>

        <button id="endButton">측정 종료</button>

    </div>

    <script src="renderer.js"></script>

</body>

</html>

# Package.json

## 파일 설명

npm 라이브러리 및 실행, 빌드 스크립츠

{

  "name": "electronactivate",

  "version": "1.0.0",

  "main": "main.js",

  "scripts": {

    "start": "electron .",

    "build" : "electron-builder --win --x64",

    "deploy": "electron-builder --windows nsis:ia32"

  },

  "build": {

    "productName": "test",

    "appId": "com.electron.hello",

    "asar": false,

    "protocols": {

      "name": "test",

      "schemes": [

        "test"

      ]

    },

    "win": {

      "target": [

        "zip",

        "nsis"

      ],

      "icon": "./resources/installer/Icon.ico"

    },

    "nsis": {

      "oneClick": false,

      "allowToChangeInstallationDirectory": true

    },

    "directories": {

      "buildResources": "./resources/installer/",

      "output": "./exeResult/",

      "app": "."

    },

    "extraResources": [

      "node\_modules/\*\*\*/\*\*/\*"

    ]

  },

  "author": "",

  "license": "ISC",

  "description": "",

  "devDependencies": {

    "electron": "^31.3.0",

    "electron-builder": "^24.13.3",

    "express": "^4.19.2",

    "mqtt": "^5.9.0",

    "mysql": "^2.18.1",

    "pkg": "^5.8.1",

    "serialport": "^12.0.0",

    "sqlite3": "^5.1.7",

    "webpack": "^5.93.0",

    "webpack-cli": "^5.1.4",

    "mqtt-packet": "^9.0.0"

  }

}