**AllSense IOT mqtt 통신을 위한 SerialData sub**

소스코드 구조설명서

**2024.** **09**

**㈜쿠노소프트**

- 목차 –

1. [main.js 4](#_bookmark0)
   1. [파일 설명 4](#_bookmark1)
   2. toKoreanTime
   3. insertData
   4. [subscribeToMqtt 4](#_bookmark3)
   5. Db 연결 및 종료
   6. createwindow
   7. [app 5](#_bookmark4)
   8. [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)
4. [index.html 21](#_bookmark42)
   1. [파일 설명 21](#_bookmark43)
5. [package.json 22](#_bookmark44)
   1. [파일 설명 22](#_bookmark45)
6. [Qtbrowser.py 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.

## toKoreanTime

: 시간을 전달받아 grafana가 읽을 수 있는 format으로 전환

const toKoreanTime = (*date*) => {

    const utc = *date*.getTime() + (*date*.getTimezoneOffset() \* 60000);

    const kst = new Date(utc + (9 \* 3600000)); *// UTC+9*

*return* kst.toLocaleString('ko-KR', { timeZone: 'Asia/Seoul' }).replace('T', ' ').substring(0, 23);

};

## insertData : Serial port 로 전달받은 데이터를 db에 저장

function insertData(*timestamp*, *topic*, *data*) {

    db.run(

        `

        INSERT INTO sensor\_data (timestamp, topic, status, CH2O, MQ131, VOC, NH3, CO, NO2, CO2, Temperature, Humidity, C3H8, C4H10, CH4, H2, C2H5OH)

        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

        `,

        [

*timestamp*,

*topic*,

*data*.status,

*data*.CH2O,

*data*.MQ131,

*data*.VOC,

*data*.NH3,

*data*.CO,

*data*.NO2,

*data*.CO2,

*data*.Temperature,

*data*.Humidity,

*data*.C3H8,

*data*.C4H10,

*data*.CH4,

*data*.H2,

*data*.C2H5OH

        ],

        (*err*) => {

*if* (*err*) {

*if* (*err*.code === 'SQLITE\_BUSY') {

                    console.error('Database busy, retrying...');

                    setTimeout(() => insertData(*timestamp*, *topic*, *data*), 1000); *// 1초 후 재시도*

                } *else* {

                    console.error('Database insertion error:', *err*);

                }

            } *else* {

*// console.log('Data saved to SQLite');*

            }

        }

    );

}

## subscribeToMqtt

: 시리얼포트와 연결 및 mqtt 서버 subscribe 구현

function subscribeToMqtt() {

    mqttClient.subscribe('#', (*err*) => {

*if* (*err*) {

            sendData = `구독 실패`;

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

            console.error('Subscription error:', *err*);

            setTimeout(subscribeToMqtt, 5000); *// 5초 후 다시 시도*

        } *else* {

            sendData = `mqtt 구독 성공`;

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

            console.log(`Subscribed to topic`);

        }

    });

    mqttClient.on('message', (*topic*, *message*) => {

*try* {

*if* (!topics.includes(*topic*)){

                sendData = `구독 topic 명 : ${*topic*}`;

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

                topics.push(*topic*);

            }

            const data = JSON.parse(*message*.toString());

            const now = new Date();

            const time = new Date(now.getTime());

            const timestamp = time.toISOString();

*if* (status[*topic*] == undefined){

                status[*topic*] = 0

            }

*if* (data.status == 1 && status[*topic*]==0){

                status[*topic*] = 1;

                sendData = `${*topic*} 측정 시작 시간: ${toKoreanTime(now)}\n`;

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

                fs.appendFileSync('측정시간.txt', `${*topic*} 측정 시작 시간: ${toKoreanTime(now)}\n`);

                console.log("측정이 시작되었습니다.");

            }*else* *if*(data.status == 0 && status[*topic*]==1){

                status[*topic*] = 0;

                sendData = `${*topic*} 측정 종료 시간: ${toKoreanTime(now)}\n`;

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

                fs.appendFileSync('측정시간.txt', `${*topic*} 측정 종료 시간: ${toKoreanTime(now)}\n`);

                console.log("측정이 종료되었습니다.");

            }

*// 데이터 삽입*

            insertData(timestamp, *topic*, data);

        } *catch* (e) {

            console.error('Error parsing message:', e);

        }

    });

}

## db 연결 및 종료

const db = new sqlite3.Database('sensor\_data.sqlite');

*// 데이터베이스 테이블 생성*

db.run(`

    CREATE TABLE IF NOT EXISTS sensor\_data (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        timestamp TEXT,

        topic TEXT,

        status FLOAT,

        CH2O FLOAT,

        MQ131 FLOAT,

        VOC FLOAT,

        NH3 FLOAT,

        CO FLOAT,

        NO2 FLOAT,

        CO2 FLOAT,

        Temperature FLOAT,

        Humidity FLOAT,

        C3H8 FLOAT,

        C4H10 FLOAT,

        CH4 FLOAT,

        H2 FLOAT,

        C2H5OH FLOAT

    )

`);

process.on('SIGINT', () => {

    db.close((*err*) => {

*if* (*err*) {

            console.error('Error closing database:', *err*);

        }

        console.log('Database closed');

        process.exit(0);

    });

});

## createWindow : 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');

}

## 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('checkOn', async (*event*) => {

*try* {

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

            dataBuffer.status = 1;

            console.log("check start");

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

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

            console.log("check already start");

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

        }

    } *catch* (error) {

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

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

    }

});

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

*try* {

*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');

        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'),*

    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('시리얼 포트 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;

});

# index.html

## 파일 설명

Electron 실행시 표시되는 페이지를 구성할 html파일

<!DOCTYPE html>

<html lang="ko">

<head>

    <meta charset="UTF-8">

    <title>센서 자료 분석</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>센서 자료 분석</h1>

    <div id="scrollBox">

*<!-- 스크롤 박스 내용 (예시) -->*

    </div>

*<!-- <div id="buttonContainer">*

*<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:x64"

  },

  "build": {

    "productName": "test",

    "appId": "com.electron.hello",

    "asar": true,

    "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"

  }

}

# Qtbrowser.py

## 파일 설명

sqlite에 저장된 값을 엑셀파일로 다운로드하는 프로그램 qt designer로 제작한 qt.ui를 불러와 사용한다.

*import* sys

*from* PyQt5.QtWidgets *import* \*

*from* PyQt5 *import* uic

*from* test *import* download

formMain = uic.loadUiType("qt.ui")[0]

class MyWindow(QMainWindow,formMain): *#윈도우,ui모두 상속*

    fileName = ""

    topic = ""

    status = 2

    a = ""

    cb1 = 0

    cb2 = 0

    cb3 = 0

    cb4 = 0

    cb5 = 0

    cb6 = 0

    def \_\_init\_\_(*self*):

        super().\_\_init\_\_()

*self*.setupUi(*self*)

*self*.btn1.clicked.connect(*self*.btn1Clicked)

*self*.led1.textChanged.connect(*self*.led1Changed)

*self*.led2.textChanged.connect(*self*.led2Changed)

*self*.rbn1.toggled.connect(*self*.rbn1Checked)

*self*.rbn2.toggled.connect(*self*.rbn2Checked)

*self*.rbn3.toggled.connect(*self*.rbn3Checked)

*self*.rbn1.setChecked(True)

*self*.cb1.stateChanged.connect(*self*.cb1Checked)

*self*.cb2.stateChanged.connect(*self*.cb2Checked)

*self*.cb3.stateChanged.connect(*self*.cb3Checked)

*self*.cb4.stateChanged.connect(*self*.cb4Checked)

*self*.cb5.stateChanged.connect(*self*.cb5Checked)

*self*.cb6.stateChanged.connect(*self*.cb6Checked)

*self*.cb1 = 0

*self*.cb2 = 0

*self*.cb3 = 0

*self*.cb4 = 0

*self*.cb5 = 0

*self*.cb6 = 0

    def cb1Checked(*self*):

*if* *self*.cb1 == 0:

*self*.cb1 = 1

*else*:

*self*.cb1 = 0

    def cb2Checked(*self*):

*if* *self*.cb2 == 0:

*self*.cb2 = 1

*else*:

*self*.cb2 = 0

    def cb3Checked(*self*):

*if* *self*.cb3 == 0:

*self*.cb3 = 1

*else*:

*self*.cb3 = 0

    def cb4Checked(*self*):

*if* *self*.cb4 == 0:

*self*.cb4 = 1

*else*:

*self*.cb4 = 0

    def cb5Checked(*self*):

*if* *self*.cb5 == 0:

*self*.cb5 = 1

*else*:

*self*.cb5 = 0

    def cb6Checked(*self*):

*if* *self*.cb6 == 0:

*self*.cb6 = 1

*else*:

*self*.cb6 = 0

    def led1Changed(*self*,*text*):

*self*.fileName = *text*

    def led2Changed(*self*,*text*):

*text* = list(*text*.split(","))

*self*.topic = *text*

    def rbn1Checked(*self*):

*self*.status = 2

    def rbn2Checked(*self*):

*self*.status = 1

    def rbn3Checked(*self*):

*self*.status = 0

    def btn1Clicked(*self*):

*self*.a = download(*self*.fileName,*self*.topic,*self*.status,*self*.cb1,*self*.cb2,*self*.cb3,*self*.cb4,*self*.cb5,*self*.cb6)

*if* *self*.a == "success":

            QMessageBox.about(*self*,"다운 완료","다운로드 성공")

*elif* *self*.a == "fail":

            QMessageBox.about(*self*,"다운 실패","다운로드 실패")

*#깃허브에 이런 소스코드들이 즐비하다 가져와 쓰는것을 추천*

*else*:

            QMessageBox.about(*self*,"실패","잘못된 형식")

*# def btn2Clicked(self):*

*#     btnQa = QMessageBox.information(*

*#         self,*

*#         "삭제",*

*#         "자료를 삭제하시겠습니까?",*

*#         QMessageBox.Yes | QMessageBox.Cancel*

*#     )*

*#     if btnQa == QMessageBox.Yes:*

*#         QMessageBox.about(self,"삭제완료","자료삭제완료")*

*#     else:*

*#         QMessageBox.about(self,"삭제미완료","자료삭제미완료")*

app = QApplication(sys.argv)

window = MyWindow()

window.show()

app.exec\_()

*# pyinstaller -w -F 01.py*