ioBrocker >> OpenEPaperLink (OEPL)

Die Versionen (nur als Info):

JavaScript	v8.8.3
VIS	v1.5.6
js-controller	v7.0.6
Node.js:	v20.18.2
NPM:	v10.8.2

Alle Beispiele wurden für den Tag von Hanshow Nebular 350Y-N (384 x 184 Pixel) gemacht

Falls jemand sich danach überglücklich füllt und mir ein Bier spendieren möchte)))
https://www.paypal.com/paypalme/AWdesign4you

1. Vorbereitung auf dem Host

a. Die folgenden NPM-Module sollen installiert werden: axios, puppeteer, sharp
Dafür die Konsole auf dem Server, wo ioBroker installiert ist, öffnen, oder sich per SSH an dem Server anmelden und die Befehle ausführen:

Locale Installation:

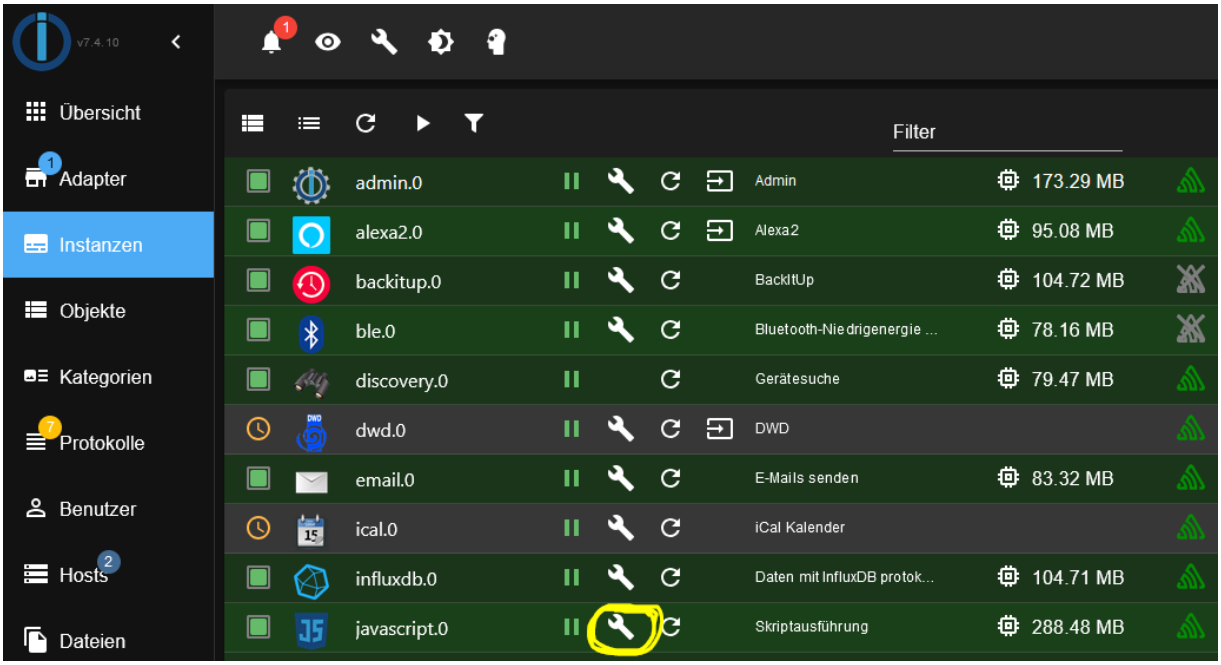
```
npm install axios
npm install puppeteer
npm install sharp
```

Die lokale Installation sollte reichen, aber falls es nicht funktioniert, kann man die Module auch global installieren:

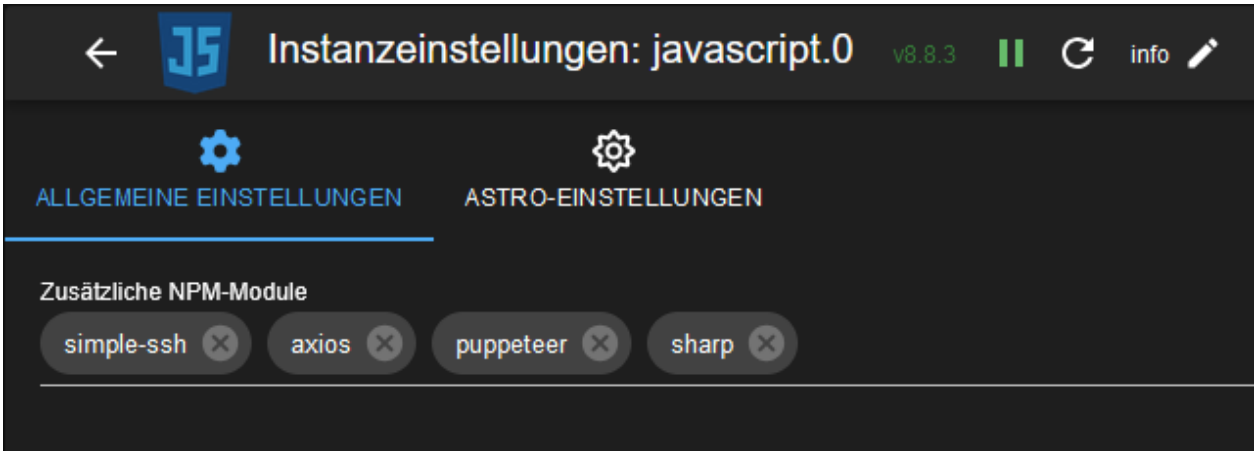
```
sudo npm install axios -g
sudo npm install puppeteer -g
sudo npm install sharp -g
```

b. Nach der Installation sollen die Module im ioBroker aktiviert werden.

Dafür ioBroker starten >> Instanzen >> javascript.0 >> Einstellungen öffnen

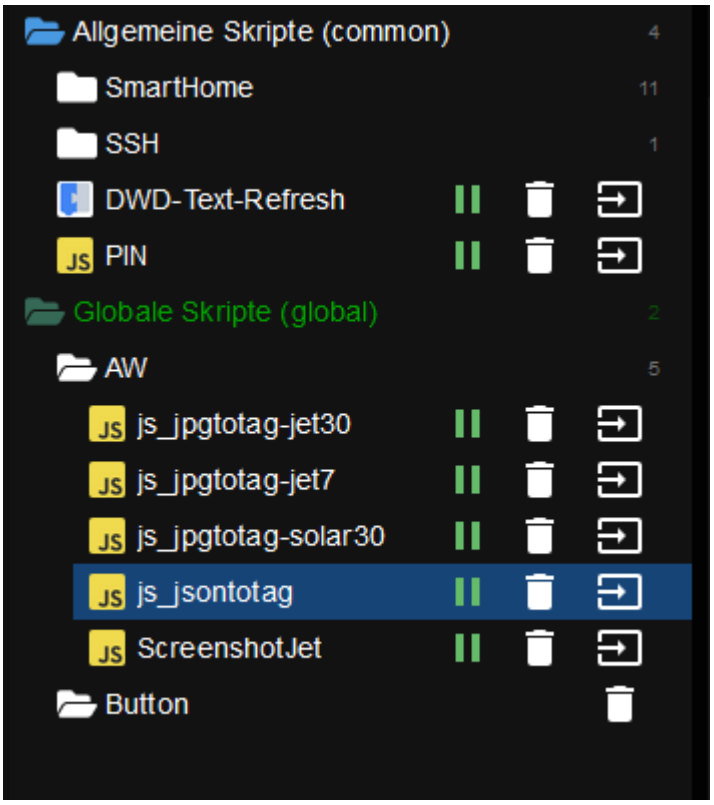


Und unter zusätzliche NPM-Module hinzufügen und anschließend speichern:

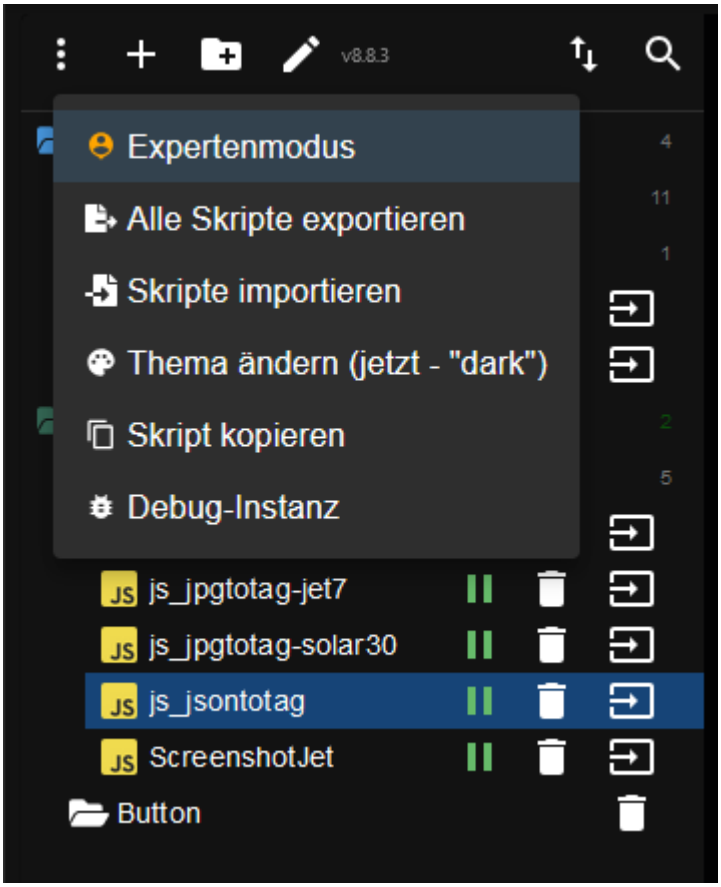


2. Die Infos aus ioBroker mit Hilfe von JSON senden

Ich habe eine Funktion **jsontotag(fmac, fjson)** erstellt. Man kann damit mit der Eingabe von der MAC-Adresse des Displays in OpenEPaperLink und dem Inhalt in JSON-Format die Infos direkt an Display senden. Die Datei mit der Funktion habe ich unter Globale Scripte (global) gespeichert, damit kann man die Funktion überall aufrufen.



Um da zu speichern, soll man die Expertenmodus aktivieren:



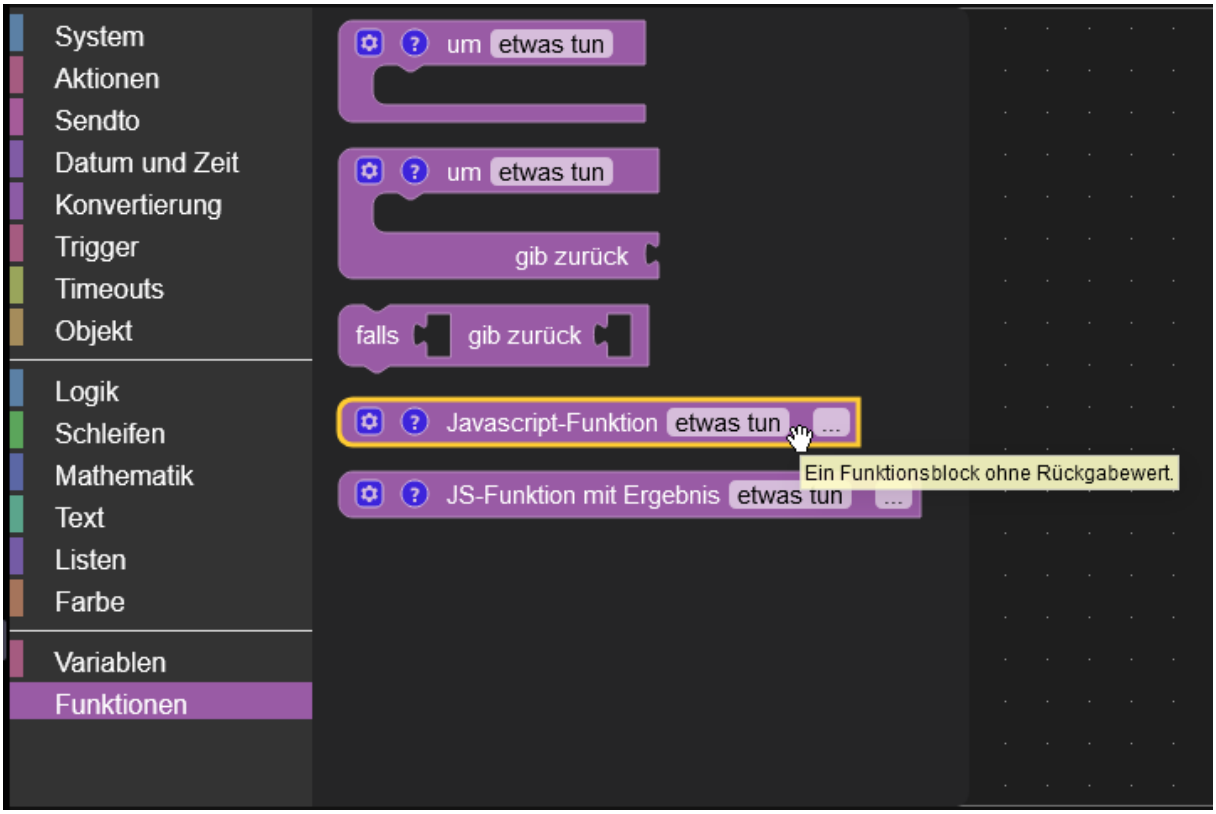
a. Danach neue Datei erstellen >> JS Script >> js_jsontotag als Name eingeben

Code einfügen und speichern (IP-Adresse von OEPL anpassen, ich habe als Beispiel 192.168.0.123 genommen):

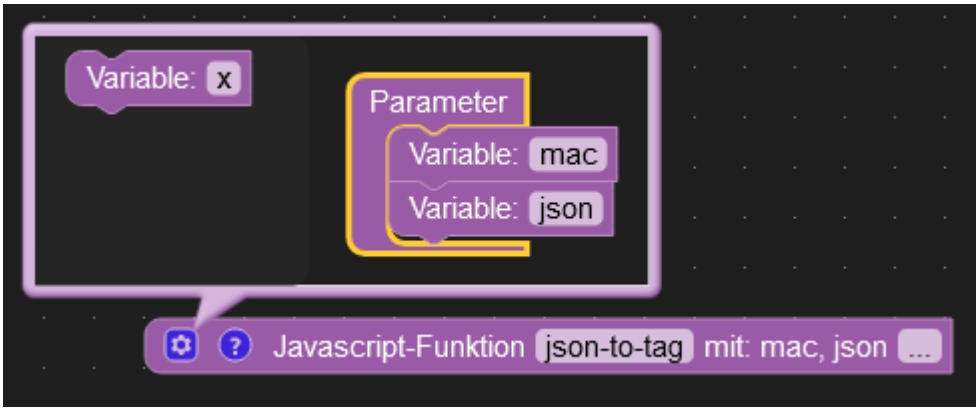
```
function jsontotag(fmac, fjson){
    const axios = require('axios');
    const qs = require('qs'); // Wird für die Formatierung benötigt (installiere es bei Bedarf mit `npm install qs`)

    // Zielserver-URL
    const serverUrl = "http://192.168.0.123/jsonupload"; // Passe die URL an

    // Daten ins x-www-form-urlencoded-Format umwandeln
    const formData = qs.stringify({
        mac: fmac,
        json: JSON.stringify(fjson) // JSON muss als String codiert werden
    });

    // Anfrage senden
    axios.post(serverUrl, formData, {
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        }
    }).then(response => {
        console.log("Erfolgreich gesendet:", response.data);
    }).catch(error => {
        if (error.response) {
            console.error("Fehler:", error.response.status, error.response.statusText);
            console.error("Server-Antwort:", error.response.data);
        } else {
            console.error("Fehler beim Senden der Anfrage:", error.message);
        }
    });
}
```

b. Jetzt kann man bequem Blockly erstellen:

Dafür erst die Funktion erstellen (z.B**. json-to-tag**) mit Parametern **mac** und **json:**

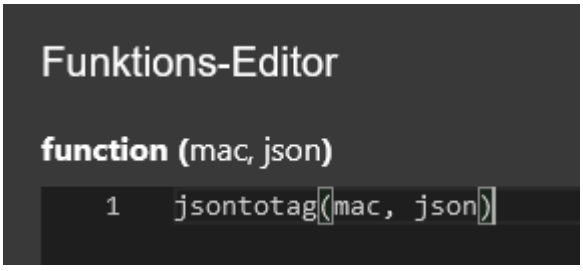


Die Funktion umbenennen, zwei Variablen ziehen und auch umbenennen:



Danach auf drei Punkte klicken und an unsere globale Funktion verweisen:

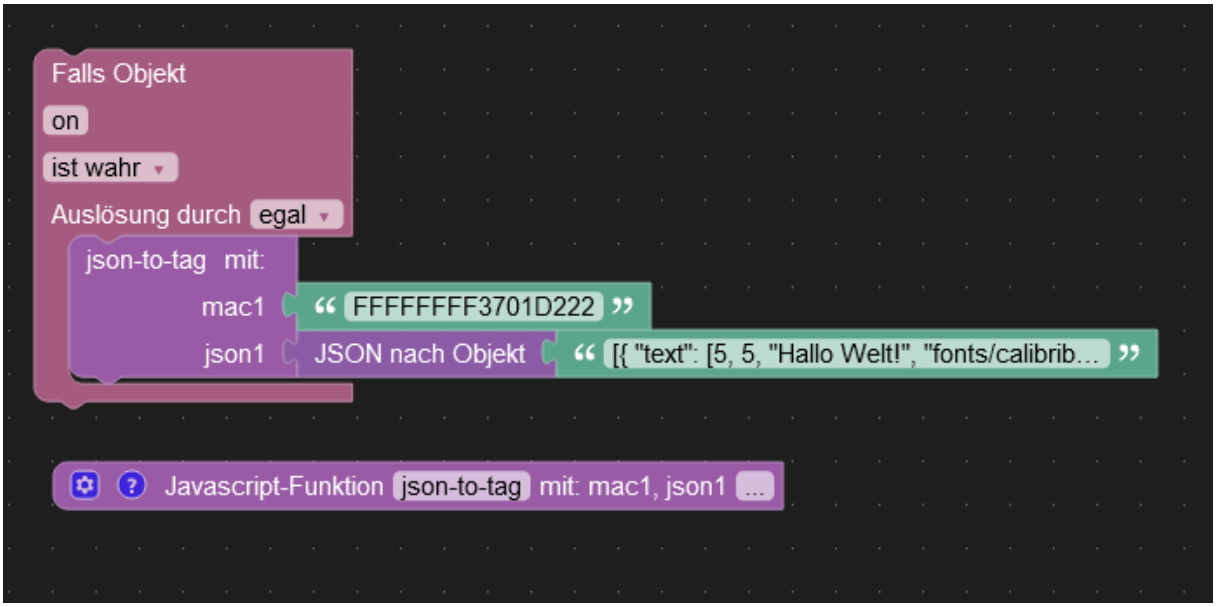
```
jsontotag(mac, json)
```



Jetzt kann man die Funktion in Blockly aufrufen. Hier ein Beispiel mit dem Ikea-Button (mac anpassen!):

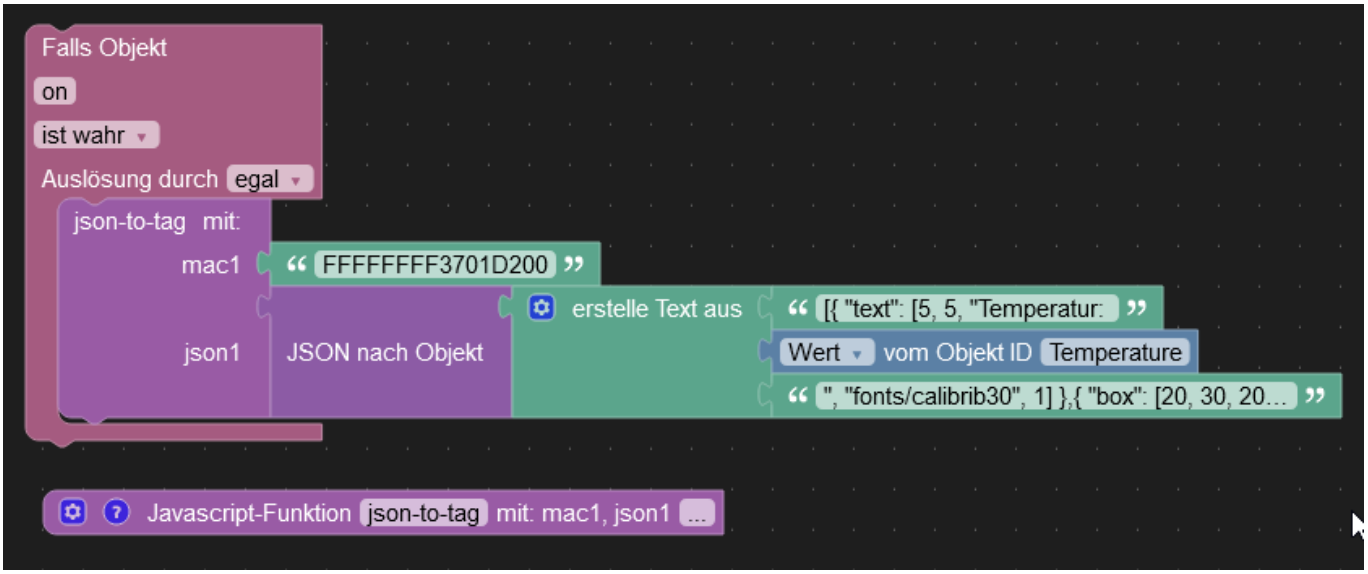
JSON-Text im Beispiel (hier mehr https://github.com/OpenEPaperLink/OpenEPaperLink/wiki/Json-template):

```
[{ "text": [5, 5, "Hallo Welt!", "fonts/calibrib30", 1] },{ "box": [20, 30, 20, 20, 3] },{ "textbox": [5, 50, 200, 50,
"SuperBox", "fonts/bahnschrift20", 3, 1] }]
```



Mit ein bisschen Glück soll es dann so aussehen :)

...oder statt „Hallo Welt!“ die Temperatur vom Sensor ausgeben



3. Die Infos aus ioBroker als Screenshot (z.B. von VIS) senden

a. Genau wie im Punkt 2 erstellen wir eine Datei unter Globale Scripte (global) mit einer globalen Funktion `takeScreenshots()` (der Dateiname ist unwichtig, bei mir heißt sie ScreenshotJet) mit dem Inhalt:

```
async function takeScreenshots(){
    const puppeteer = require('puppeteer');

    (async () => {
        // Browser starten
        const browser = await puppeteer.launch({
            headless: true, // Headless-Modus aktivieren
            args: ['--no-sandbox', '--disable-setuid-sandbox'] // Sicherheitsoptionen für Serverumgebungen
        });

        // Neue Seite öffnen
        const page = await browser.newPage();

        // Ziel-URL aufrufen
        await page.goto('http://192.168.0.111:8082/vis/index.html#DeinVisName', { waitUntil: 'networkidle2' });

        // Screenshot erstellen
        await page.screenshot({ path: '/tmp/screenshot1.png', fullPage: true });

        console.log('Screenshot gespeichert als screenshot1.png');

        // Browser schließen
        await browser.close();

        const sharp = require("sharp");

        const inputPath = "/tmp/screenshot1.png";
        const outputPath = "/tmp/screenshot1_epaper.jpg";  // Ausgabe als .jpg (JPEG Format)

        const targetWidth = 384;
        const targetHeight = 184;


        try {
            // Erhalte die Bildabmessungen
            const metadata = await sharp(inputPath).metadata();

            if (metadata.width < targetWidth || metadata.height < targetHeight) {
            throw new Error("Das Bild ist kleiner als die Zielgröße und kann nicht zugeschnitten werden.");
            }

            await sharp(inputPath)
            .extract({ left: 24, top: 30, width: targetWidth, height: targetHeight }) // Zuschneiden
            .resize(targetWidth, targetHeight)  // Skalierung des Bildes auf Display-Auflösung
            //.grayscale()  // Konvertiere es in Graustufen
            //.threshold(185)  // Wandelt es in Schwarz-Weiß basierend auf einem Schwellenwert
            .withMetadata({ density: 1 })  // Setze die DPI auf 1
            .jpeg({ quality: 100, chromaSubsampling: "4:4:4" }) // Speichern als JPG
            .toFile(outputPath);  // Als JPEG speichern

            console.log(`Bild erfolgreich für das E-Paper Display vorbereitet und gespeichert unter: ${outputPath}`);
        } catch (error) {
            console.error("Fehler beim Bearbeiten oder Speichern des Bildes:", error.message);
        }
    })();
}
```

Folgende Zeilen müssen angepasst werden:

Die Seite, wovon der Screenshot aufgenommen werden soll:
'http://192.168.0.111:8082/vis/index.html#DeinVisName'

Die Auflösung vom OEPL-Display
```
const targetWidth = 384;
const targetHeight = 184;
```

Der Abstand von oben links, wo der Screenshot beginnt (z.B. bei `left: 0, top: 0`**, wird er ohne Abstand von oben links anfangen).**
```
.extract({ left: 24, top: 30, width: targetWidth, height: targetHeight }) // Zuschneiden
```

Nach Bedarf kann man die sharp-Optionen hinzufügen / ändern
(z.B. //.grayscale() auskommentieren für S/W Bild)

```
await sharp(inputPath)
.extract({ left: 24, top: 30, width: targetWidth, height: targetHeight }) // Zuschneiden
.resize(targetWidth, targetHeight)  // Skalierung des Bildes auf Display-Auflösung
//.grayscale()  // Konvertiere es in Graustufen
//.threshold(185)  // Wandelt es in Schwarz-Weiß basierend auf einem Schwellenwert
.withMetadata({ density: 1 })  // Setze die DPI auf 1
.jpeg({ quality: 100, chromaSubsampling: "4:4:4" }) // Speichern als JPG
.toFile(outputPath);  // Als JPEG speichern
```

b. Nach dem wird das Bild aufgenommen, angepasst und gespeichert haben kann man es an Display senden. Dafür noch eine Datei unter Globale Scripte (global) mit einer globalen Funktion `sendScreenshot1totag(fmac)` erstellen und den Code einfügen:

```
function sendScreenshot1totag(fmac){
    const axios = require('axios');
    const FormData = require('form-data');
    const fs = require('fs');

    const filePath = '/tmp/screenshot1_epaper.jpg';
    const uploadUrl = 'http://10.192.0.123/imgupload';
    const macAddress = fmac;
    const ditherValue = '1'; // Setze den Dither-Wert entsprechend den API-Anforderungen

    async function uploadImage() {
        try {
            // Prüfen, ob die Datei existiert
            if (!fs.existsSync(filePath)) {
                throw new Error(`Datei nicht gefunden: ${filePath}`);
            }

            // FormData erstellen
            const form = new FormData();
            form.append("mac", macAddress);
            form.append("dither", ditherValue); // Dither-Parameter hinzufügen
            form.append("file", fs.createReadStream(filePath));

            // POST-Anfrage senden
            const response = await axios.post(uploadUrl, form, {
                headers: {
                    ...form.getHeaders(),
                },
            });

            console.log('Upload erfolgreich:', response.data);
        } catch (error) {
            if (error.response) {
                console.error('Server-Antwort:', error.response.status, error.response.data);
            } else if (error.request) {
                console.error('Keine Antwort erhalten:', error.request);
            } else {
                console.error('Fehler beim Hochladen:', error.message);
            }
        }
    }

    // Hochladen starten
    uploadImage();

}
```

Folgende Zeilen müssen angepasst werden:

Die IP-Adresse von OEPL-Access Point, ich habe als Beispiel 192.168.0.123 genommen:

```
'http://10.192.0.123/imgupload';
```

Das Dithering einschalten (1) / ausschalten (0) :

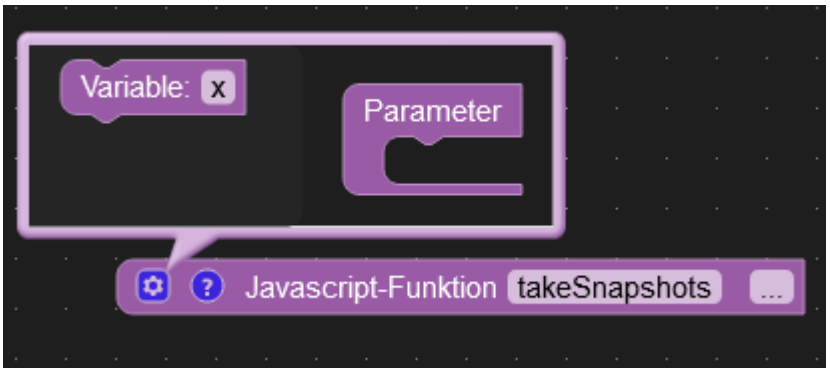
```
ditherValue = '1';
```

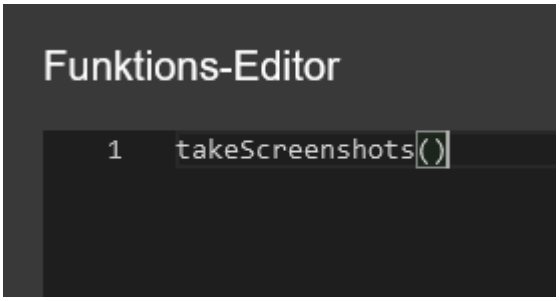
c. Jetzt kann man Blockly erstellen und da die MAC-Adresse vom Display eingeben:

Dafür machen wir zwei Funktionen takeScreenshots() und sendScreenshot1(mac)

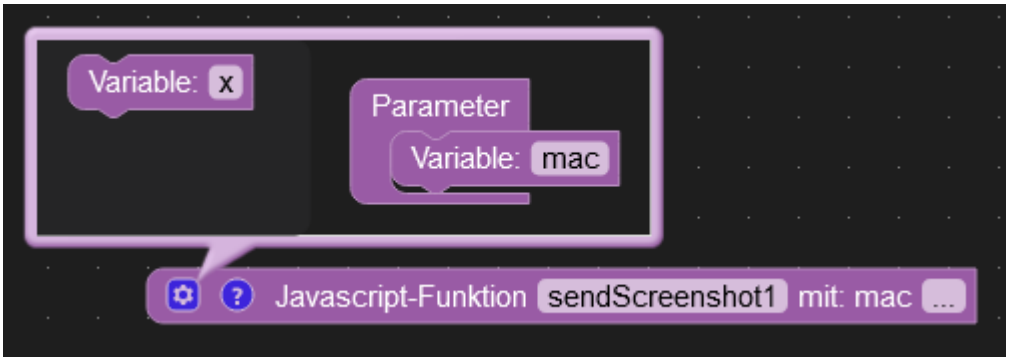
(Die Schritte sind gleich wie bei JSON)

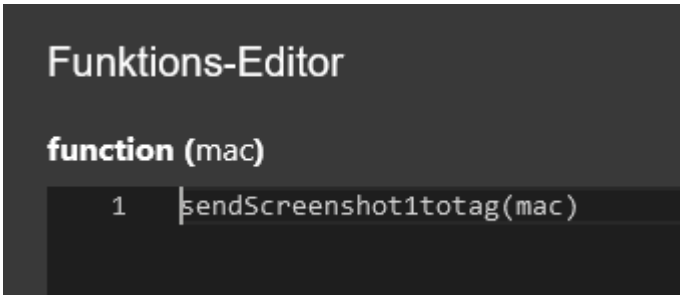
takeScreenshots()



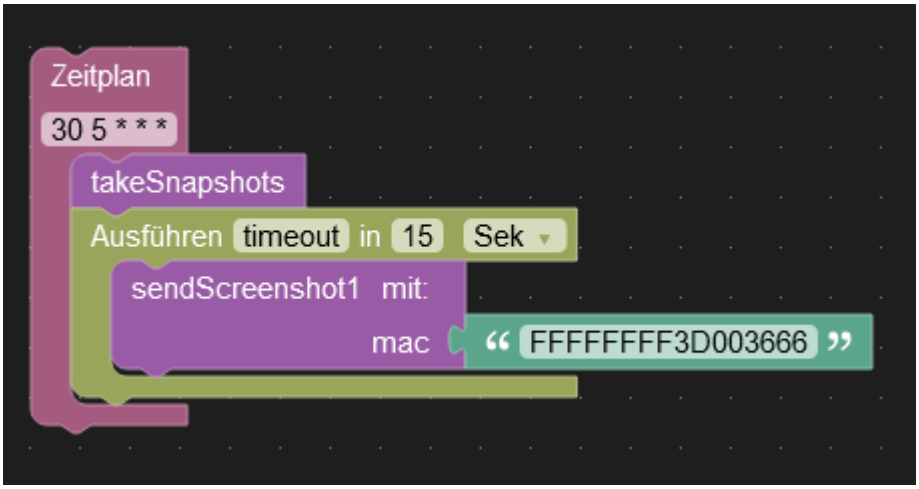


sendScreenshot1(mac):





Und als Beispiel: jeden Tag um 5:30 den Screenshot erstellen, dann 15 Sekunden Zeit für die Bildbearbeitung geben und an den OEPL-Tag mit der MAC-Adresse „FFFFFFFF3D003666“ senden:



Hier noch ein Beispiel:

Das ist die VIS-Seite mit dem Link: http://192.168.0.123:8082/vis/index.html#imgupload :

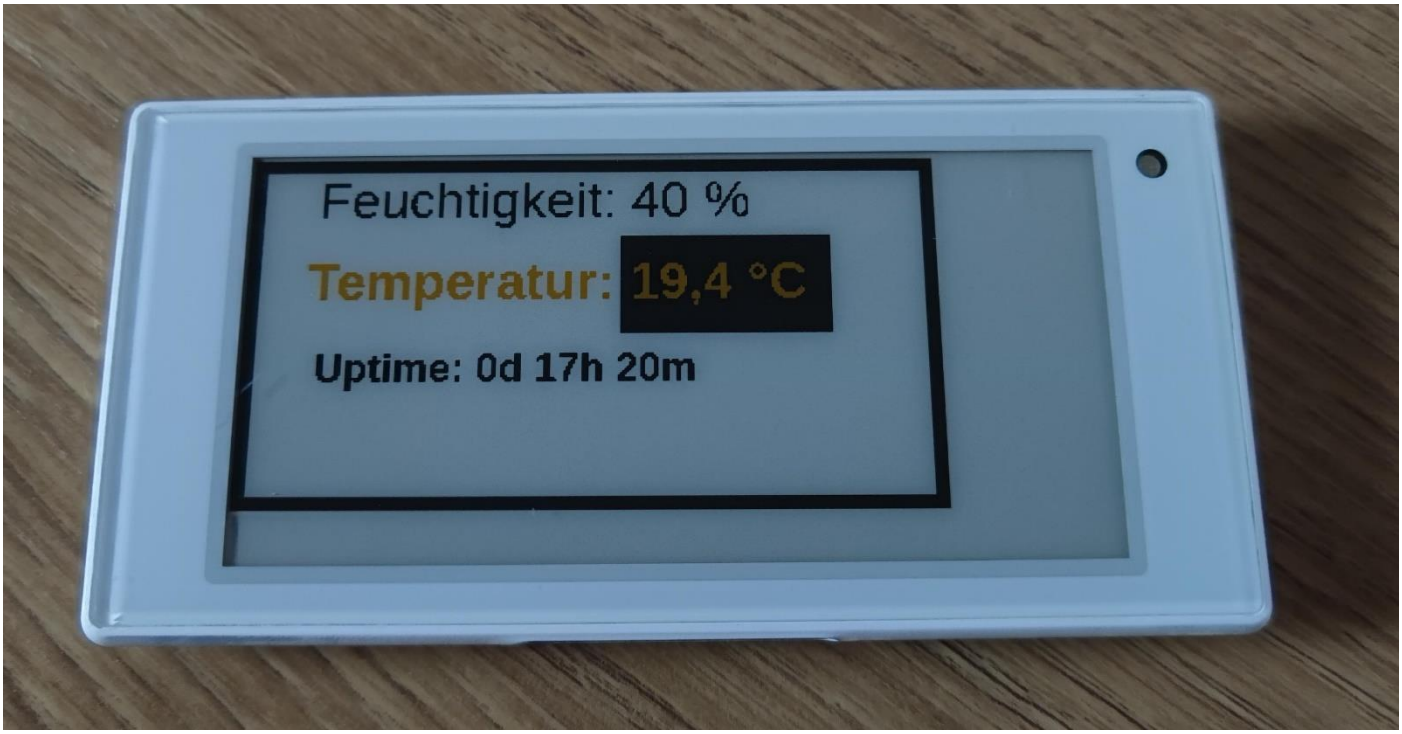
Der Abstand von oben 35px und von links auch 35px. Die Ramengröße ist 300x150px, und die Displayauflösung ist 384x184px. Wenn wir die Funktion **`takeScreenshots()`** so anpassen:

```
...
await page.goto(' http://192.168.0.123:8082/vis/index.html#imgupload', { waitUntil: 'networkidle2' });
...
.extract({ left: 35, top: 35, width: targetWidth, height: targetHeight }) // Zuschneiden
```

Da ich das Bild nur in schwarz, weiß und gelb habe und mein Display genau diese drei Farben ausgeben kann, kann man Dithering in der Funktion **`sendScreenshot1totag(fmac)`** deaktivieren:

```
const ditherValue = '0'; // Setze den Dither-Wert entsprechend den API-Anforderungen
```

...und das wird die Ausgabe sein:



Ich hoffe es hilft jemandem

have fun :)

AW