Web of Things at FOSDEM 2026

Quick start

- Print photos on the receipt printer (select a photo then click “Invoke”)
- Display images on the e-paper screen

Advanced: HTTP API

Web of Things is a set of standards for Internet of Things use cases. At the Matrix Hackathon and FOSDEM 2026, I brought the following hardware. They support support Web of Things via HTTP and Websockets.

Receipt printer

Current status: Ready

- Epson TM-T20III
- Native colors: 1-bit
  - `#ffffff` (white)
  - `#000000` (black)
- Native resolution: 576 pixels wide, any height (will be converted to be divisible by 8 pixels)
- Supported image formats: PNG (best results), JPG

URLs

- URL for printing images: https://wot-wrench.chrpaul.de/#try=https://wot-why.jaller.de/receipt-printers-for-events/photo
  - URL for scripts: https://wot-why.jaller.de/receipt-printers-for-events/photo/actions/print-photo
- UI for printing text: https://wot-wrench.chrpaul.de/#try=https://wot-why.jaller.de/receipt-printers-for-events/text
  - URL for scripts: https://wot-why.jaller.de/receipt-printers-for-events/text/actions/print-text

Code examples (images)

Bash:

```
curl -X POST \
  -H "Content-Type: application/json" \
  --data "{\"image\":\"$(base64 image.png | tr -d '\n')\"}" \
  https://wot-why.jaller.de/receipt-printers-for-events/photo/actions/print-photo
```

Or in NodeJS / Bun / Deno:

```
import { readFile } from "node:fs/promise";

const image = await readFile("image.png");
const response = await fetch("https://wot-why.jaller.de/receipt-printers-for-events/photo/actions/print-photo", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    image: image.toString("base64"),
  }),
});

if (!response.ok) {
  throw Error(`Failed to queue print job. HTTP ${response.status}`);
}
```

Code examples (text)

Bash:

```
curl -X POST \
  --header "Content-Type: application/json" \
  --data '{"text":"Hello World!"}' \
```

```
  https://wot-why.jaller.de/receipt-printers-for-events/text/actions/print-
text
```

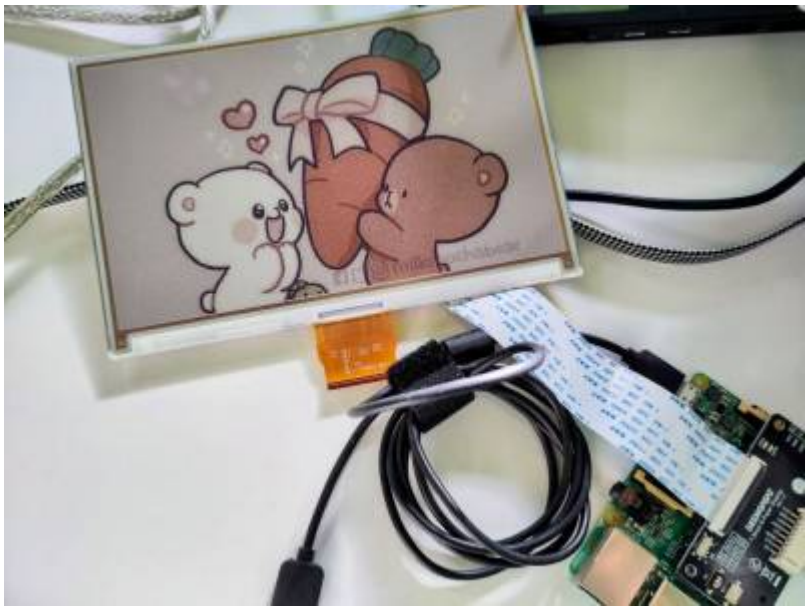
Or in NodeJS / Bun / Deno:

```
const response = await
fetch("https://wot-why.jaller.de/receipt-printers-for-events/text/actions/pr
int-text", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    text: "Hello World!",
  }),
});

if (!response.ok) {
  throw Error(`Failed to queue print job. HTTP ${response.status}`);
}
```

E-paper screen

Current status: Ready at
https://wot-wrench.chrpaul.de/#try=https://wot.chrpaul.de/image-storage/LDN3dJE3vu



- 7.3inch ACeP 7-Color E-Paper Display
- Native colors: 7 colors
  - `#000000` (black)
  - `#ffffff` (white)
  - `#ff0000` (red)
  - `#00ff00` (green)
  - `#0000ff` (blue)

    - `#ffff00` (yellow)
    - `#ff8000` (orange)
- Native resolution: 800×480 pixels
- Supported image formats: PNG (best results), JPG

URLs

Use these URLs to prepare for the event or test your scripts. They will not display anything on the screen, but the Thing has a preview property.

- Test URL: https://wot-wrench.chrpaul.de/#try=https://wot.chrpaul.de/image-storage/fosdem
    - Image property for scripts: https://wot.chrpaul.de/image-storage/fosdem/properties/image
- Production URL for scripts: https://wot-wrench.chrpaul.de/#try=https://wot.chrpaul.de/image-storage/LDN3dJE3vu
    - Image property for scripts: https://wot.chrpaul.de/image-storage/LDN3dJE3vu/properties/image

Code Examples

To upload a local PNG file `image.png` via a Bash:

```
curl -X PUT \
  -H "Content-Type: application/json" \
  --data "\"$(base64 image.png | tr -d '\n')\"" \
  https://wot.chrpaul.de/image-storage/fosdem/properties/image
```

Or in NodeJS / Bun / Deno:

Have a look at this demo project: https://codeberg.org/jaller94/wot-fosdem-demo

```
import { readFile } from "node:fs/promise";

const image = await readFile("image.png");
const response = await
fetch("https://wot.chrpaul.de/image-storage/fosdem/properties/image", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(image.toString("base64")),
});

if (!response.ok) {
  throw Error(`Failed to upload image. HTTP ${response.status}`);
}
```

Dithering

Images will be resized and dithered automatically. However, if you want full control over the result, you can send images in the right resolution using only the supported color palette.

For TypeScript, I recommend using my fork of dither-me-this. It supports custom color palettes which is optimal for the e-paper screens. Previously, I used my fork of canvas-dither. Its API is easier and more mature, but it only supports 1-bit dithering. It's most suitable for the receipt printer.