Siebenschläfer Display - Documentation

1. Introduction

1.1 About the Siebenschläfer Display System

The Siebenschläfer™ Display is a wireless, solar-powered electronic paper display system designed for long-term, maintenance-free operation. It combines e-paper technology with integrated solar cells and supercapacitors to create a display that requires no cables, no batteries, and virtually no maintenance once installed.

The system consists of three main components:

- **Siebenschläfer™ Display** - The actual e-paper display powered by an STM32L496 microcontroller with integrated solar panel and supercapacitor energy storage.
- **Hase (Gateway)** - A compact ESP32-S3-based gateway that bridges the LoRa radio network to WiFi or Ethernet, connecting your displays to the web backend.
- **Website** - The browser-based platform you access over the internet via https://www.siebenschläfer-display.eu/ to create layouts, manage devices, and configure data sources.

Key features:

- Solar-powered operation - even indoors (from 200 lux, even lower with Extension Panel)
- LoRa radio communication with range up to 7 km line-of-sight
- End-to-end encrypted communication (AES, Ed25519, X25519)
- Intuitive WYSIWYG layout designer with real-time preview via WebAssembly
- Lua scripting engine for advanced use cases
- Dynamic data binding via JSON/XML data sources
- Tilt-to-Act button for smart home automation via webhooks
- Long-term maintenance-free operation thanks to supercapacitors instead of aging batteries

1.2 What's in the Box?

Siebenschläfer™ Display package:

- Siebenschläfer™ Display (7-inch 6-color e-paper, 800×480 resolution)
- Integrated anti-glare glass with UV filter
- Integrated 32 GB SD card
- Mounting hardware for wall installation

Hase (Gateway) package:

- Hase Gateway device
- Integrated 32 GB SD card
- USB-C power cable
- USB wall power adapter

Optional accessories:

- Extension Panel - Additional solar surface for darker environments (chainable)
- Window mount

1.3 System Requirements

To use the Siebenschläfer™ Display system, you need:

For the Website:

- A modern web browser (Google Chrome, Microsoft Edge, Firefox, or Safari)
- Internet connection to access the web backend
- A free account on the Siebenschläfer platform

For the Gateway:

- USB-C power supply (5V)
- Ethernet connection to your router **or** WiFi network (2.4 GHz)

For the Display:

- Sufficient ambient light (minimum 200 lux for daily refresh, 400 lux for updates every 3 hours)
- Range of a Hase Gateway (LoRa 433 MHz @10mW)
- No power cable required - the display is fully self-powered

1.4 Quick Start Guide

For a step-by-step guide to getting your first display up and running, refer to the Quick Start Guide. It will walk you through:

1. Setting up the Hase (Gateway)
2. Registering your Display
3. Creating your first layout
4. Assigning the layout to your display

2. Hardware - Siebenschläfer Display

The Siebenschläfer™ Display is the core component of the system - a self-powered e-paper display that requires no cables and can operate long-term without maintenance.

2.1 Technical Specifications

2.1.1 E-paper Display (800×480) The Siebenschläfer™ uses a 7-inch color e-paper display with a resolution of 800×480 pixels. E-paper technology offers several advantages:

- **Static image retention** - The displayed content remains visible even without power, consuming zero energy between updates.
- **Sunlight readability** - Unlike LCD or OLED screens, e-paper is easily readable in direct sunlight with no glare.
- **Color support with dithering** - Through smart dithering algorithms, the display can render photos and colorful content that appears vivid despite the e-paper medium.
- **Long lifespan** - The e-paper panel has a rated lifespan of approximately 1 million refresh cycles.

The display is protected by anti-glare glass with an integrated UV filter, which both improves readability in bright environments and protects the sensitive e-paper panel from ultraviolet radiation.

2.1.2 Microcontroller (STM32L496) The display is powered by an STM32L496 microcontroller from STMicroelectronics, selected for its ultra-low power consumption:

- **ARM Cortex-M4** core with FPU (Floating Point Unit)
- Capable of running the LVGL graphics engine directly on-device for rendering layouts
- Supports Lua scripting through an integrated script engine
- Can enter deep sleep modes drawing as little as 20 µA in standby
- Allows for attaching the for low power microcontroller rather large RAM that is needed to render the layout on device

The microcontroller handles all display operations including: receiving encrypted radio data from the gateway, parsing layout definitions and data bindings, rendering the final image using LVGL, and driving the e-paper panel refresh cycle.

2.1.3 RAM and Flash Memory

Component	Specification
RAM	4 MB ultra-low-power parallel SRAM
Flash	32 MB internal flash memory for the firmware
Storage	32 GB SD card slot for media storage

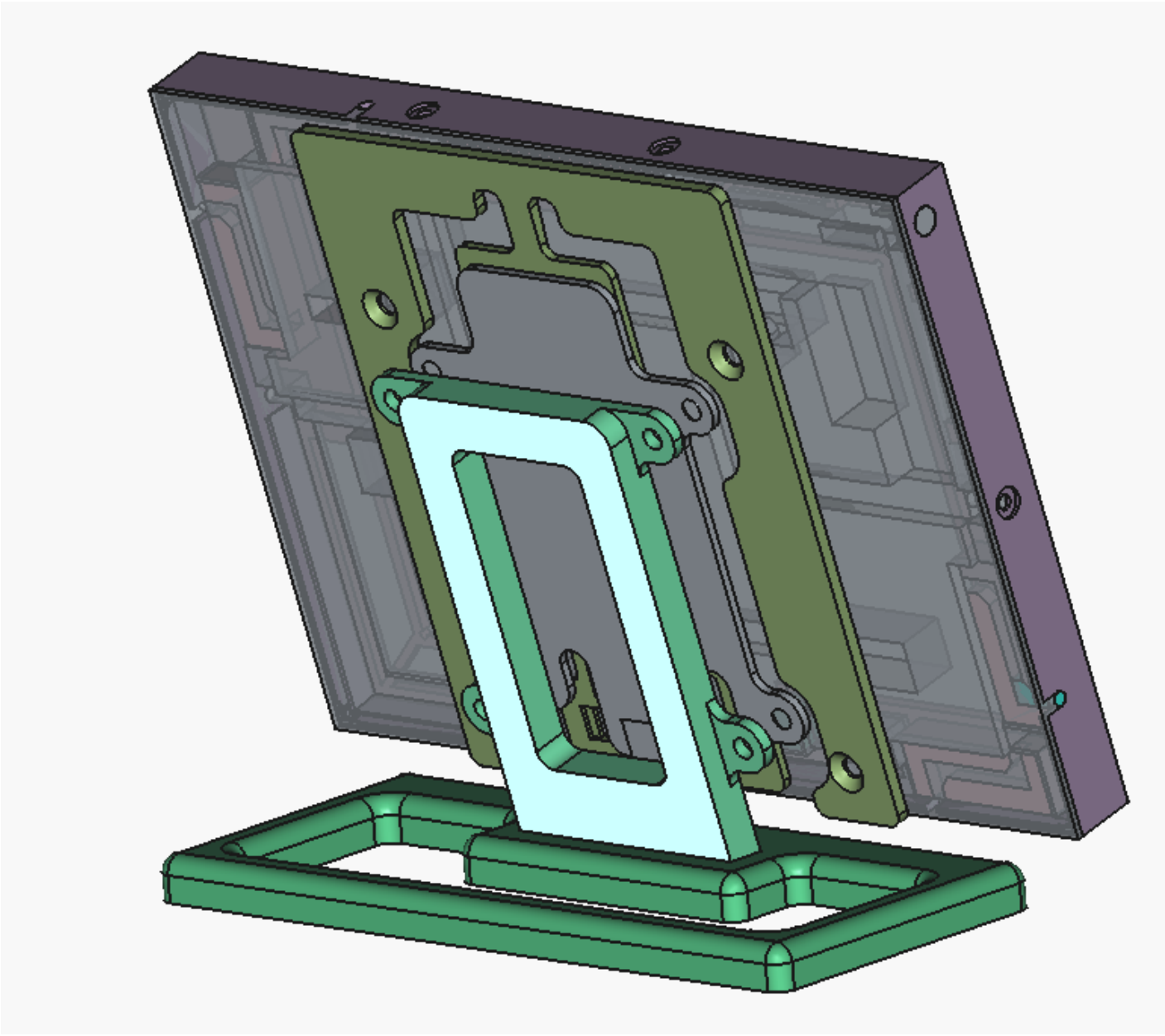
The generous memory allows the display to store multiple layouts, cached images, and Lua scripts locally. The SD card provides additional space for resource files such as fonts, icons, and image assets that are downloaded from the web backend.

2.2 Enclosure and Connectors

The Siebenschläfer™ Display features a slim, elegant enclosure designed to blend into any environment:

- **Enclosure dimensions**: 21.5 × 15.5 × 1.7 cm
- **Display area**: 16 × 9.6 cm (approximately 7 inches diagonal)
- **Solar frame width**: 2 cm around the display perimeter
- **Weight**: around 400g

The enclosure is designed to be repairable - it can be opened without destroying any components, and all parts are replaceable. STL files for enclosure components are available for 3D printing.



2.3 Solar Panel and Power Supply

2.3.1 Integrated Solar Panel A Topcon solar panel is integrated into the 2 cm wide frame surrounding the e-paper display. This panel captures ambient light to power all display operations:

- Captures even diffuse indoor lighting
- Powers both the microcontroller operations and the e-paper refresh cycle
- Charges the supercapacitor bank continuously during operation

The solar frame is designed to be visually unobtrusive while providing sufficient surface area for reliable indoor operation.

2.3.2 Supercapacitors - How They Work and Maintenance Instead of traditional lithium batteries, the Siebenschläfer™ uses **5.5F VINATech EDLC supercapacitors** for energy storage. This is a deliberate design choice with significant advantages:

- **No aging** - Unlike batteries that degrade over time, supercapacitors maintain their capacity for decades
- **Long service life** - No battery replacement required

- **Safe storage without charge** - The display can be stored uncharged for years without damage (unlike lithium batteries that can swell or become hazardous)
- **High charge/discharge cycles** - Supercapacitors can handle millions of cycles without degradation

The supercapacitor bank stores enough energy to perform multiple e-paper refreshes even during periods of low light. The display's power management system continuously monitors the charge level and adjusts the refresh rate accordingly.

2.3.3 Minimum Light Conditions (Lux Values) The Siebenschläfer™ Display requires a minimum amount of ambient light to operate:

Light Level	Refresh Rate
200 lux	Approximately 1 refresh per day
400 lux	Approximately 1 refresh every 3 hours
300+ lux (recommended)	Up to 1 refresh every 5 minutes when energy allows

For reference: - **200 lux** = Dimly lit room or overcast daylight through a window - **400 lux** = Normal office lighting - **500-1000 lux** = Well-lit room or bright indoor environment

In environments with insufficient light, the display will enter a low-power state and refresh less frequently. For darker locations, one or multiple Extension Panels can be added (see Section 2.4).

2.3.4 Refresh Rate and Power Consumption The display schedules a radio transmission window every 5 minutes when sufficient energy is available. The actual refresh rate depends on:

1. **Current charge level** - Higher charge allows more frequent updates
2. **Ambient light availability** - More light means faster recharging between updates
3. **Content complexity** - Simple text layouts consume less power to render than complex images
4. **Data volume** - New images take longer to transmit than simple JSON or XML data.
5. **Connection quality** - The transmission speed over the radio link depends on connection quality and uses LoRa or GFSK at the fastest possible rate.

State	Power Consumption
Deep sleep (idle)	100 µW
Rendering layout	Varies by content complexity
E-paper refresh	2-3% of capacitor

The display tracks its energy requirements for both rendering and the e-paper update cycle, reporting these values to the backend so you can monitor power usage in the device overview.

2.4 Extension Panel (Optional)

2.4.1 Additional Solar Surface for Dark Environments The Extension Panel is an optional accessory that provides additional solar surface area for environments where ambient light is limited:

- Connects to the Siebenschläfer™ Display to supplement the integrated solar panel
- Enables operation in darker rooms or locations with low lighting
- Can be placed separately from the display and connected via cable

2.4.2 Chaining Multiple Panels Multiple Extension Panels can be daisy-chained together for even more solar surface area:

- Connect panels in series using the provided connectors
- No limit on the number of panels that can be chained
- Each additional panel increases the available charging surface proportionally
- theoretically 10 lux possible (but currently untested, Extension Panel still in development)

3. Hardware - Hase (Gateway)

The Hase (German for “hare”) is the gateway device that bridges your Siebenschläfer™ displays to the internet. It communicates with displays via LoRa radio and connects to the web backend through WiFi or Ethernet. A single gateway can manage an unlimited number of displays.

3.1 Technical Specifications

3.1.1 Microcontroller (ESP32-S3) The Hase is powered by an ESP32-S3 microcontroller, a powerful and versatile chip designed for IoT applications:

- **Dual-core processor** with high-performance capabilities
- Integrated WiFi and Bluetooth radio
- Hardware acceleration for cryptographic operations
- Supports gRPC communication with the web backend
- Capable of running Lua scripts for local automation

The ESP32-S3 handles all gateway functions including: maintaining the encrypted connection to the web backend via gRPC, managing LoRa radio communication with displays, distributing firmware updates over-the-air (OTA), and monitoring network status.

3.1.2 RAM and Flash Memory

Component	Specification
RAM	8 MB
Flash	8 MB internal flash memory
Storage	32 GB SD card slot for media storage

The gateway stores device registries, cached firmware images, and communication logs locally. The SD card provides additional space for logging and resource storage.

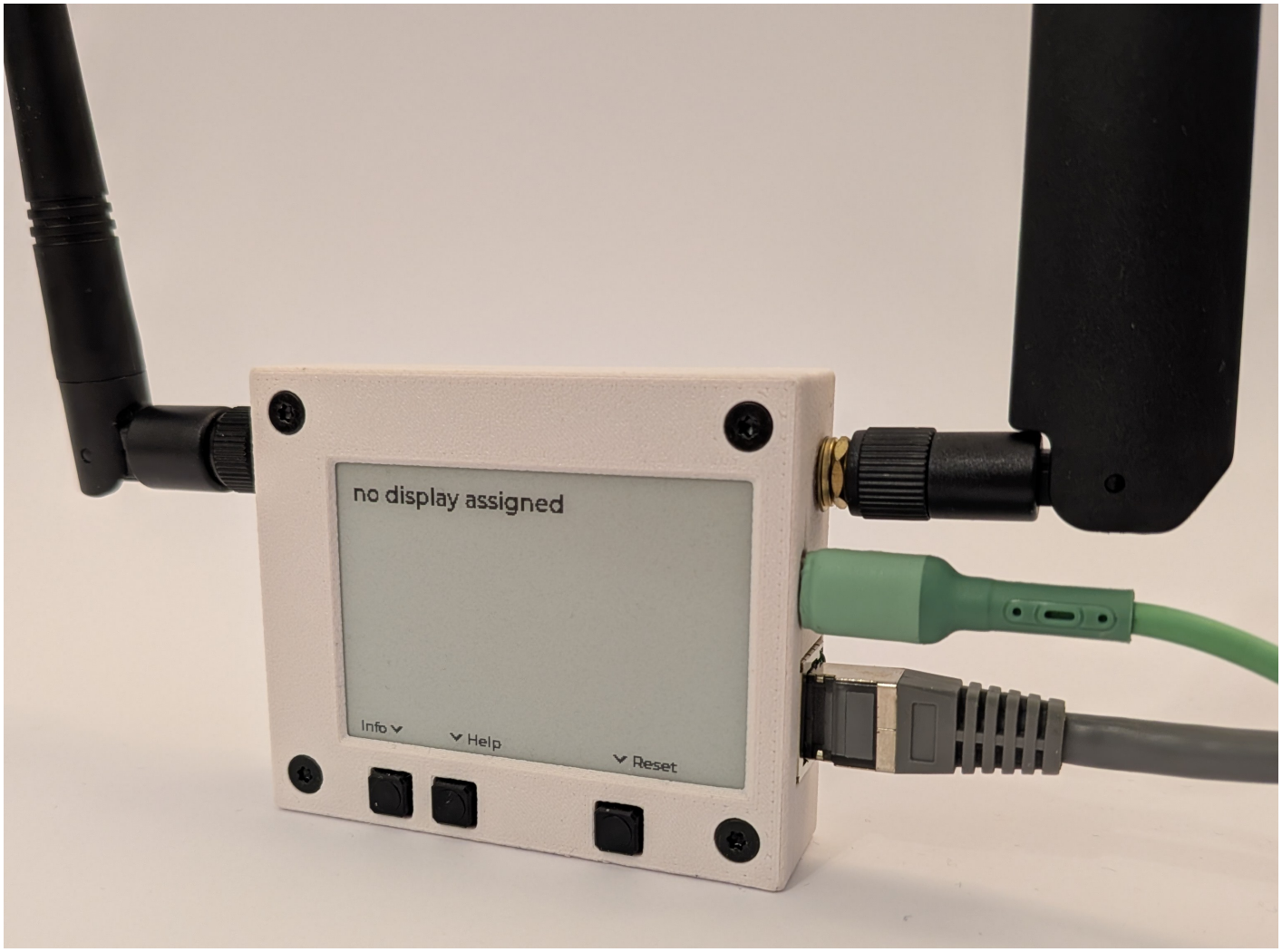
3.2 Connectors and LEDs

The Hase features a compact enclosure with the following connectors:

- **USB-C port** - Power supply (5V, 0.5 W consumption), serial connection for configuration
- **Ethernet port (RJ45)** - Wired network connection to your router
- **SD card slot** - For additional storage and logging
- **WIFI-Antenna** - Antenna for WIFI communication
- **LoRa-Antenna** - Antenna for display communication

Status indicators:

- The Hase includes a small e-paper display (5.7 × 3.8 cm area within a 7.7 × 6.4 × 2.1 cm enclosure) that shows status information including QR codes for registration, connection status, and assigned displays.



3.3 Network Connections

The Hase supports multiple network connection methods to ensure reliable connectivity in any environment. IPv4 and IPv6 is supported.

3.3.1 Ethernet The recommended connection method is via Ethernet cable:

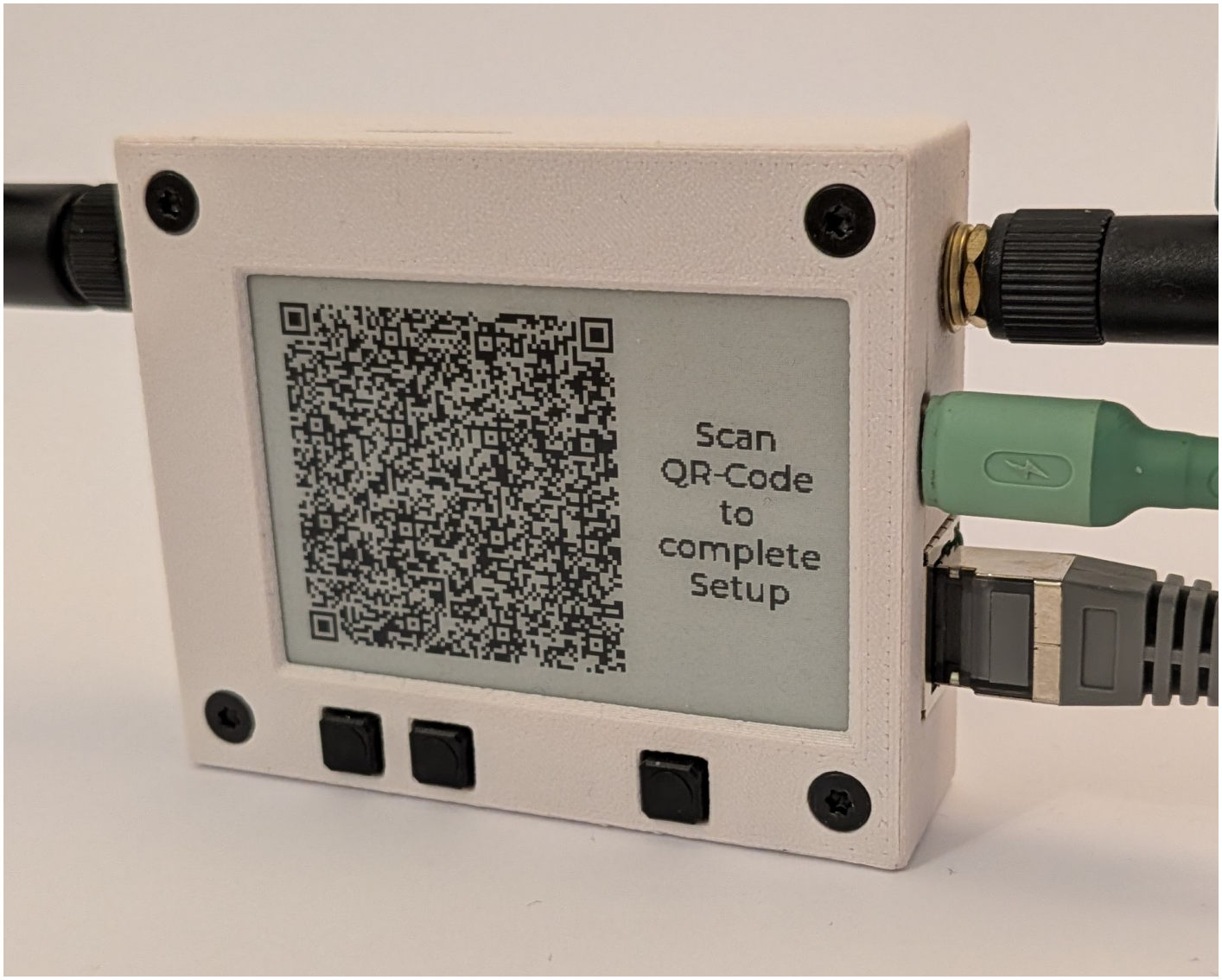
1. Connect one end of an Ethernet cable to the Hase's RJ45 port
2. Connect the other end to your router or network switch
3. The Hase will automatically obtain an IP address via DHCP

Ethernet provides the most stable and reliable connection, with no configuration required beyond physical connection.

3.3.2 Wi-Fi (WPS) For quick setup without entering credentials, the Hase supports WPS (WiFi Protected Setup):

1. Press the WPS button on your Hase
2. Within two minutes, initiate WPS pairing on your router
3. The Hase will automatically connect to your WiFi network

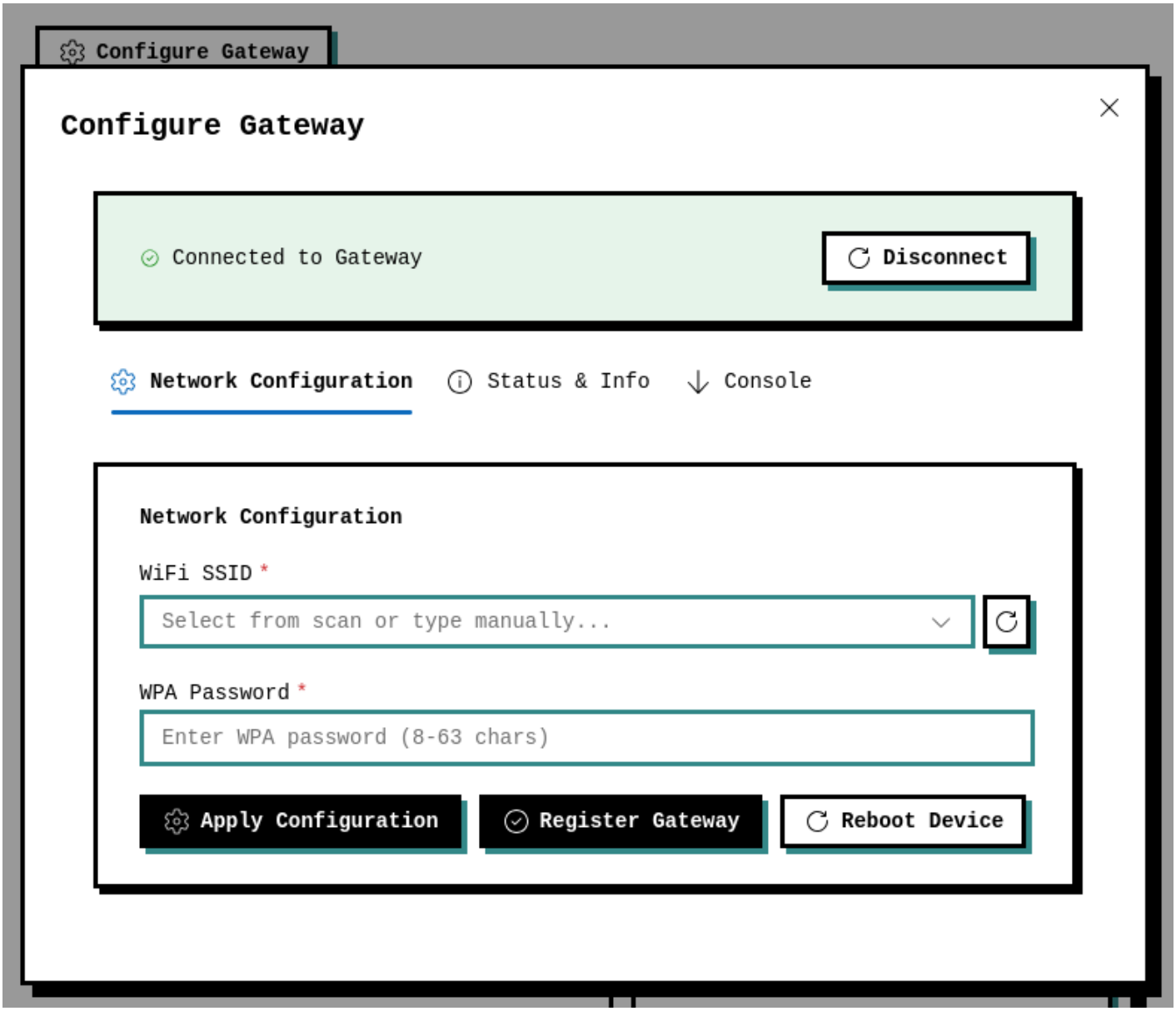
WPS provides a convenient alternative when you prefer not to enter credentials manually or when the password is unknown.



3.3.3 Wi-Fi (Manual Configuration) If Ethernet is not available, you can connect the Hase to your WiFi network manually:

1. Access the gateway configuration via device management on the website
2. Connect the gateway to your PC using the USB-C cable
3. Select Serial Interface on the gateway configuration page
4. Configure your device

The gateway will scan for available networks and display them in a list for selection. You can also type the SSID manually if your network does not appear in the scan results.



4. Installation and Commissioning

This chapter provides detailed step-by-step instructions for setting up your Hase Gateway and Siebenschläfer Display. For a condensed version, see the Quick Start Guide.

4.1 Setting Up the Hase (Gateway)

4.1.1 Connecting Power Supply (USB-C)

1. Locate the USB-C port on the Hase Gateway
2. Connect the included USB-C power cable to the Hase
3. Plug the other end into a USB power adapter (5V) or a USB port on your router/computer
4. The Hase will power on and begin its startup sequence

The Hase consumes approximately 0.5 W during normal operation. Any standard 5V USB power source is sufficient.

4.1.2 Ethernet Configuration The simplest way to connect the Hase to your network is via Ethernet:

1. Connect one end of an Ethernet cable (RJ45) to the Hase's Ethernet port
2. Connect the other end to a free LAN port on your router or network switch
3. The Hase will automatically obtain an IP address via DHCP and connect to the internet
4. Wait for the Hase to establish its connection to the web backend (this may take up to 2 minutes)

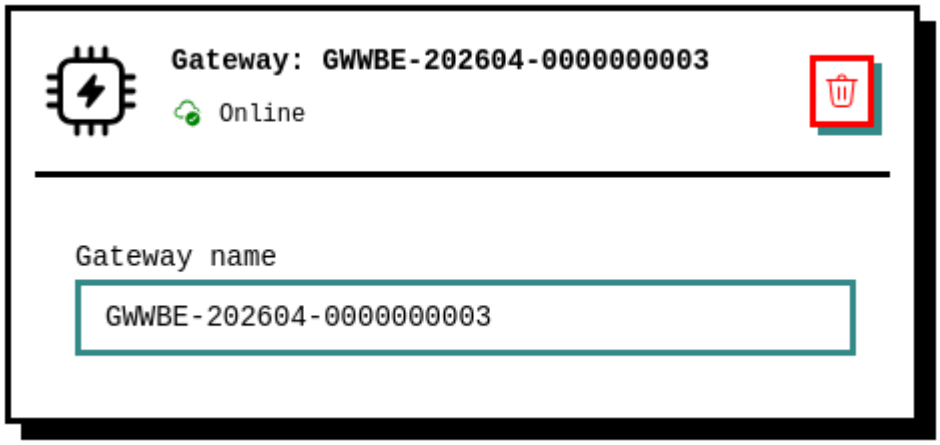
Once connected, the Hase's small e-paper display will show a QR code for registration.

Wi-Fi configuration (manual and WPS) is described in Sections 3.3.3 and 3.3.4. See there for detailed instructions.

4.1.3 Scanning QR Code and Registering Once the Hase is connected to the internet, it displays a QR code on its small e-paper screen:

1. Scan the QR code displayed on the Hase using your smartphone or camera
2. Follow the registration link to associate the Hase with your account
3. Assign a display name to your gateway (e.g., “Homelab Gateway”)

After successful registration, the Hase will appear as “Online” in your device overview on the website.



4.1.4 Configuring Gateway on the Website After registration, you can manage your gateway from the website under “Manage Devices”:

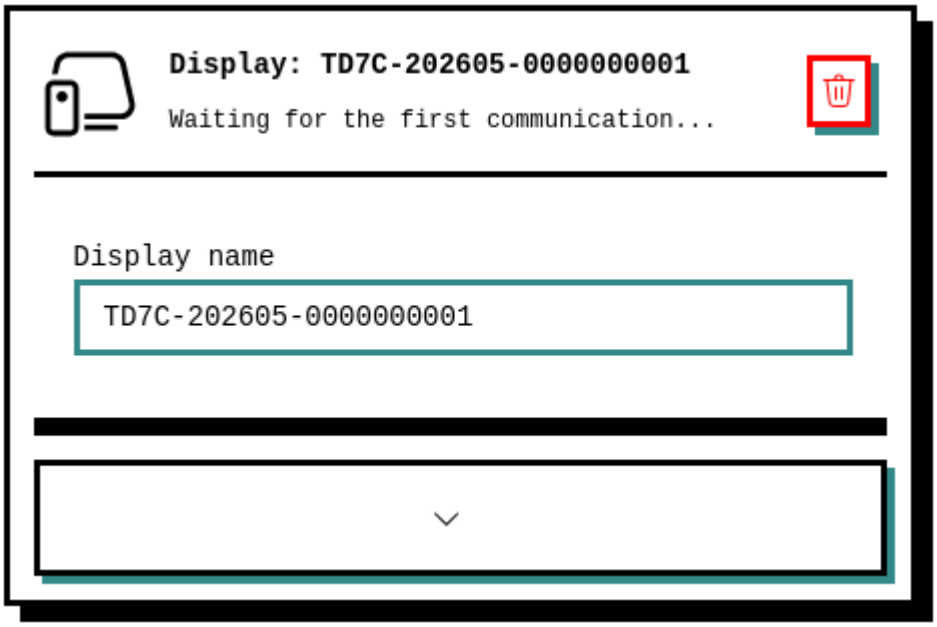
- **View status** - Check if the gateway is online and see its last seen timestamp
- **Change name** - Change the display name

4.2 Setting Up the Display

4.2.1 Scanning QR Code and Registering Each Siebenschläfer Display has a unique QR code on its back:

1. Locate the QR code sticker on the rear of the display
2. Scan it with your smartphone camera and open the website
3. Follow the registration link to associate the display with your account
4. Assign a display name (e.g., “Kitchen Display”)

After registration, the display will appear in your device overview as “Searching...” until it establishes contact with a gateway.



4.2.2 Placement and Light Conditions Proper placement is critical for reliable operation:

1. **Light requirements** - Place the display where it receives at least 200 lux of ambient light (for daily refresh) or 400 lux (for updates every 3 hours). A well-lit office environment typically provides

400-500 lux.
2. **Gateway range** - Position the display within LoRa radio range of your Hase Gateway. The typical indoor range is through approximately 3 thick walls or across several rooms.
3. **Orientation** - Decide whether you want horizontal (landscape) or vertical (portrait) orientation and create a layout accordingly. An orientation sensor will ensure the display output has the correct orientation.

Avoid placing the display in direct sunlight for extended periods, as excessive heat can affect the super-capacitors. The anti-glare glass will protect against UV radiation, but extreme temperatures should be avoided.

4.2.3 Connecting Display to Gateway The display automatically searches for a gateway after registration:

1. Place the display within range of your Hase (Gateway)
2. Ensure the display receives sufficient light to build up charge
3. In the website, navigate to “Manage Devices” and select your display
4. Assign the gateway in the “Gateway” property
5. The display will establish contact with the gateway automatically

Once connected, you can monitor the connection status in the device overview. The display state will change from “Searching…” to “Collecting Charge” as it builds up energy, then to “Fully Charged and Ready” when ready for content updates.

Display: TD7C-202605-0000000001

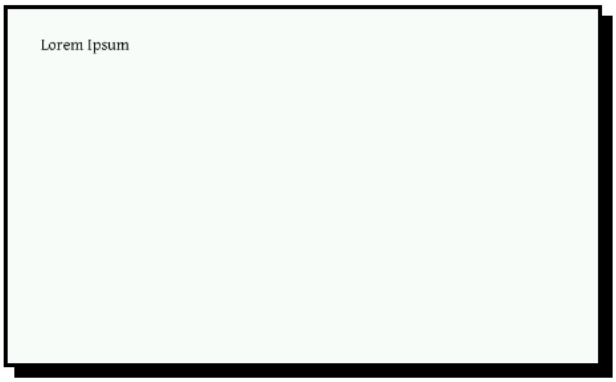
Broadcasting update

Display name

TD7C-202605-0000000001

Template

Layout 1 (Horizontal)



Last Content Update

3.8.2026, 18:05:15

Gateway

GWWBE-202604-0000000003

Data Binding

Unassigned

Power History

Energy used for rendering

7.47 %

Energy used for e-paper update

2.36 %

Firmware

0.8.0

Physical orientation

Vertical

Webhook URL

Enter a value

4.3 Mounting and Installation

The mounting system consists of three components that can be combined flexibly:

- **Display** - Features a VESA 100×100 mounting pattern on the back
- **Wall mount** - Two-part system with integrated click mechanism. One half attaches to the display's VESA 100×100 threads, the other half mounts directly to the wall (with screws). The display is then clicked into the wall-side half and locked in place. The second half also features a VESA 75×75 pattern for external brackets. Provides tilt functionality (Tilt-to-Act)
- **Stand** - Combined with the second half of the wall mount, which attaches to the stand via its VESA 75×75 pattern

4.3.1 Direct VESA 100×100 Mount (No Tilt) The display can be mounted directly to any VESA 100×100-compatible bracket:

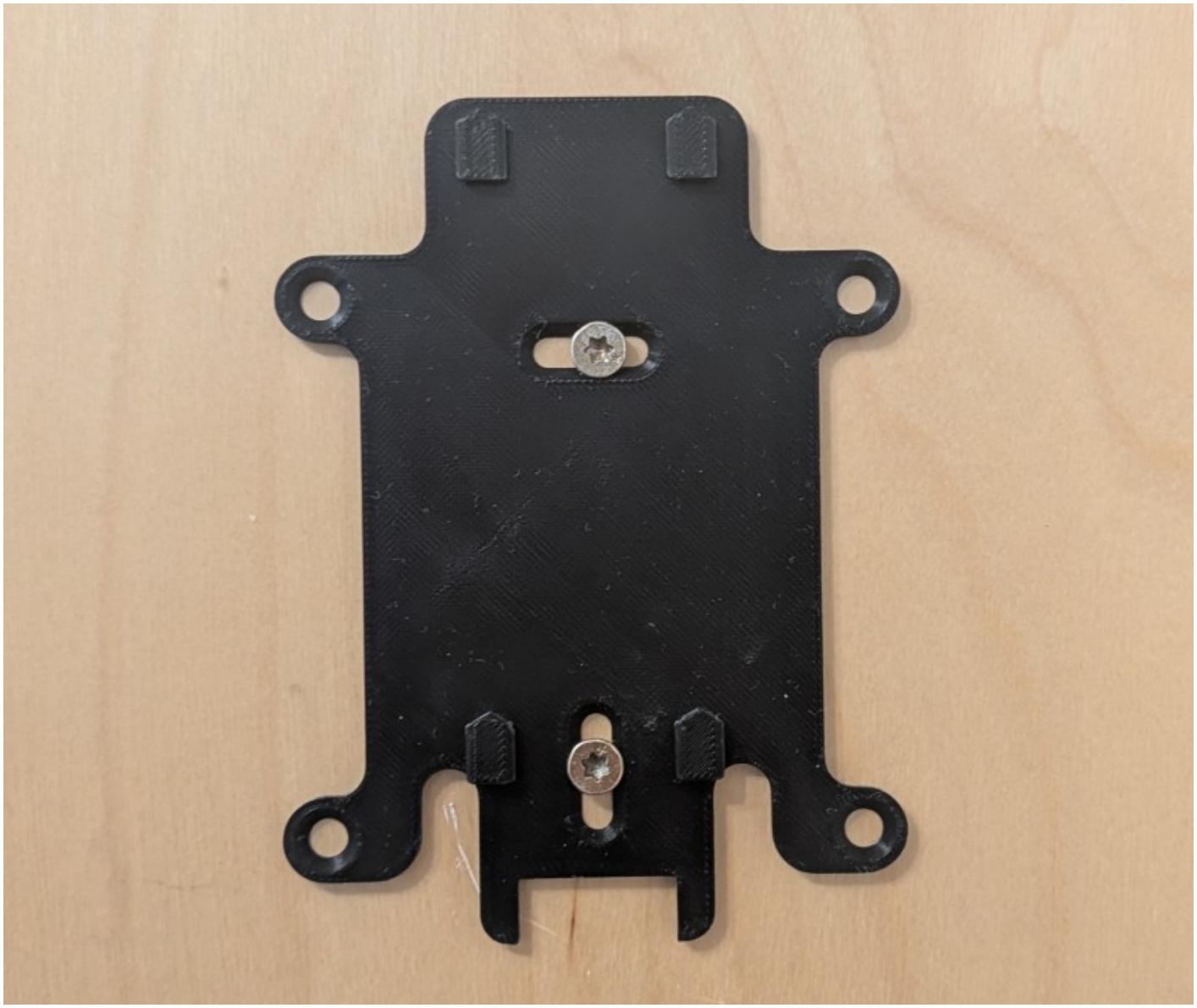
1. Attach the display directly to the VESA 100×100 bracket or monitor arm
2. Ensure the mount allows sufficient light exposure to the solar frame

Note: This mounting option does not provide tilt functionality. Tilt-to-Act is not available in this configuration.

4.3.2 Wall Mount with Tilt (Tilt-to-Act) For direct wall mounting with tilt:

1. Attach one half of the wall mount to the VESA 100×100 threaded holes on the back of the display
2. Mark and drill holes at the desired position on the wall
3. Insert wall plugs and secure the other half of the wall mount directly to the wall using the included screws
4. Click both halves together to lock the display in place on the wall

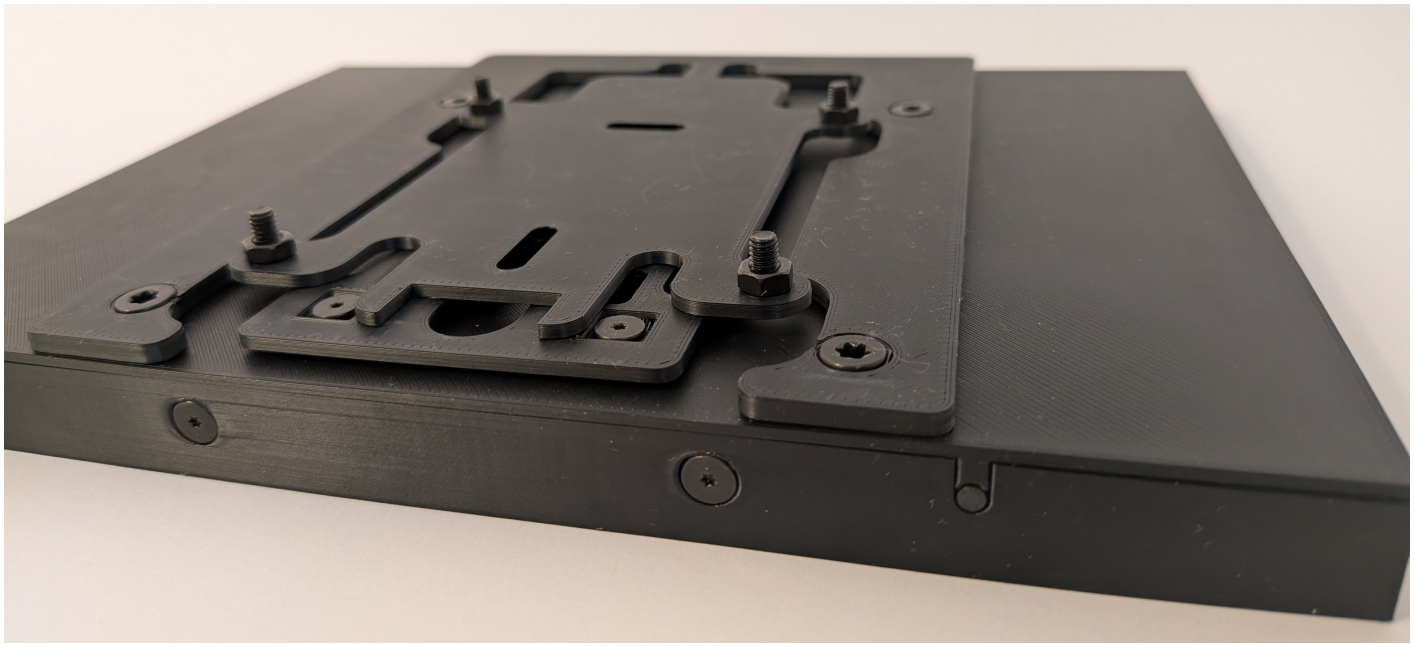
The tilt of the mount enables Tilt-to-Act functionality (see Chapter 10).



4.3.3 VESA 75×75 Mount with Tilt (Tilt-to-Act) Through the wall mount, the display can also be attached to VESA 75×75 brackets:

1. Attach one half of the wall mount to the VESA 100×100 threaded holes on the back of the display
2. Mount the other half (via its VESA 75×75 pattern) to your desired VESA bracket or monitor arm
3. Click both halves together
4. Ensure the mount allows sufficient light exposure to the solar frame

This configuration provides tilt and therefore Tilt-to-Act functionality (see Chapter 10).



4.3.4 Table Placement with Stand (Tilt-to-Act) For placement on a desk or other flat surface:

1. Attach one half of the wall mount to the VESA 100×100 threaded holes on the back of the display
2. Attach the other half of the wall mount (via its VESA 75×75 pattern) to the stand, similar to a wall mount installation
3. Click both halves together
4. Place the display on your desired surface
5. Ensure the display receives adequate ambient light from above or in front

This configuration also provides tilt and therefore Tilt-to-Act functionality (see Chapter 10). No cables are required - the display is completely self-powered.

5. Website - Getting Started

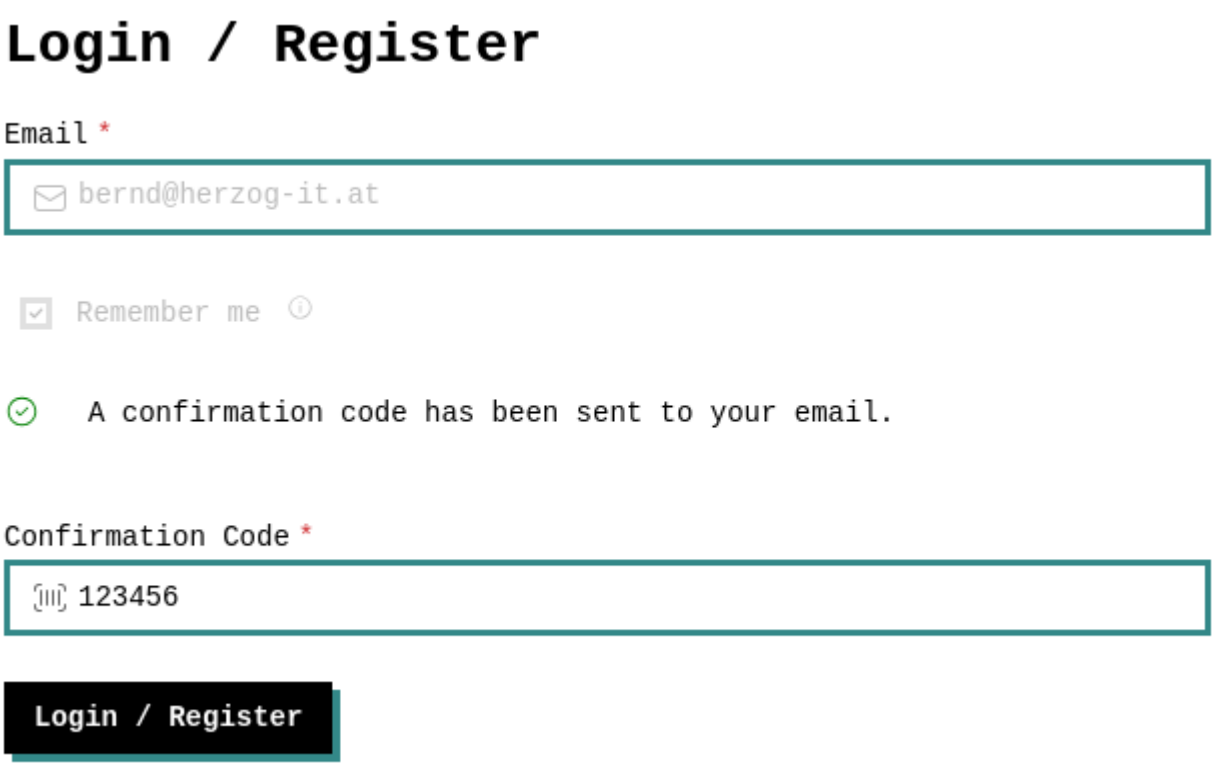
The Siebenschläfer website is your central control panel for managing all devices, creating layouts, and configuring data sources. It runs entirely in your browser and requires no software installation.

5.1 Creating an Account and Logging In

To get started:

1. Navigate to the Siebenschläfer website
2. Click “Menu” and then “Register”
3. Enter your email address
4. A confirmation code will be sent to your inbox - enter it to verify your account
5. Once verified, you are logged in to your account

Remember Me: You can enable the “Remember me” option during login, which installs a cookie for automatic session restoration. This stores a JWT in the local storage and a secure cookie to guard against session hijacking.



5.2 Navigation and Menu Structure

The main navigation menu provides access to all application areas:

- **Manage Devices** - View and configure all Gateways and Displays
- **Layout Designer** - Create and edit display layouts (WYSIWYG editor)
- **Data Sources** - Manage JSON/XML data sources for dynamic content
- **Documentation** - Access this documentation and API reference
- **Settings** - User preferences, timezone, language settings

The header bar shows your current location in the application and provides quick access to key functions.

5.3 Language Settings

The website supports multiple languages:

1. Navigate to **Menu**
2. Select your preferred language from the dropdown
3. The interface will update immediately

Your language preference is saved with your browser. Currently supported languages include English and German, with more planned for future releases.

6. Managing Devices

The “Manage Devices” section is where you configure and monitor all your Gateways and Displays.

6.1 Managing the Gateway

6.1.1 Checking Gateway Status (Online/Offline) Each gateway shows its connection status in the device overview:

- **Online** - The gateway is actively connected to the web backend via gRPC
- **Offline** - Last seen timestamp displayed; check network connectivity and power supply

Due to the gRPC KeepAlive mechanism (server ping every 2 hours), a gateway may appear "Online" for up to 2 hours after an actual connection loss, depending on client activity.

6.2 Managing the Display

6.2.1 Understanding Display Status and States The display goes through several states during its operation cycle:

6.2.1.1 "Searching ..." The display is searching for a gateway but has not yet established contact. This is normal immediately after registration, as long as the display has not yet made contact with a gateway. Check that: - The display is within range of the assigned gateway - The correct gateway is assigned in the device configuration - The gateway is online and connected to the internet

6.2.1.2 "Collecting charge" This is the idle state of the display when it has nothing to do and the supercapacitor is not fully charged. In this state, the display collects solar energy until the supercapacitor is fully charged or a content update arrives.

6.2.1.3 "Fully charged and ready" The display has sufficient energy and is ready to receive content updates. The gateway will send new content when available.

6.2.1.4 "Successfully updated content." The display has successfully received and applied a content update. This status is displayed briefly after a successful update (within 5 minutes of the last content change).

6.2.1.5 "Broadcasting update" The gateway has prepared a content update for the display and is waiting for the next transmission window to send the data via LoRa radio.

6.2.1.6 "Transferring Data" The display is receiving and processing the transmitted data (layout definition, images, or data bindings). Large layouts with many images may take longer in this state.

6.2.1.7 "Preparing image" The display's LVGL engine is rendering the layout into a final image for the e-paper panel. This includes applying dithering algorithms and preparing the refresh pattern. This is followed by the physical e-paper panel refresh, which takes an additional ~30 seconds.

6.2.1.8 "Error rendering" An error occurred during content rendering. Possible causes: - Images or fonts too large for available RAM (4 MB SRAM) - Layout too complex for the rendering engine - Missing resource references in the layout definition - Power demand exceeds supercapacitor capacity Check the layout configuration and reduce complexity or size of resources used if necessary.

6.2.2 Changing Display Name To rename your display:

1. Navigate to "Manage Devices" -> Find your Display in the list
2. Click on the current name
3. Enter a new display name (e.g., "Kitchen Display", "Office Board")
4. Save the changes

The display name is used for identification in the website and does not affect the device itself. It also appears on the small e-paper display of the gateway.

6.2.3 Assigning a Layout To assign a layout to your display:

1. Navigate to "Manage Devices" -> Find your Display in the list
2. Find the "Layout" property in the configuration panel
3. Click the dropdown to select from available layouts
4. Choose the desired layout and save

The display will receive the new layout at its next refresh cycle. The preview of the selected layout is displayed directly below the dropdown menu.

6.2.4 Assigning a Data Source To bind dynamic data to your display:

1. Navigate to “Manage Devices” -> Find your Display in the list
2. Find the “Data Binding” property
3. Click the dropdown to select from available data sources
4. Choose the desired data source and save

When the data source changes, the new data is sent to the display. For more information on data binding in layouts, see Data Sources.

6.2.5 Assigning and Changing Gateway To assign a gateway to your display:

1. Navigate to “Manage Devices” -> Find your Display in the list
2. Find the “Gateway” property
3. Click the dropdown to select from available gateways
4. Choose the desired gateway and save

6.2.6 Viewing Charge History Monitor your display’s energy levels over time:

1. Navigate to “Manage Devices” -> Find your Display in the list
2. Scroll to the “Power History” section
3. View a chart showing charge level trends over time

The power history helps you understand if your display location provides sufficient light and whether an Extension Panel might be needed for more frequent updates.

Additional energy information: - **Energy used for rendering** - Power consumed by the LVGL engine to process the layout - **Energy used for e-paper update** - Power consumed by the physical panel refresh cycle - **Last Content Update** - Timestamp of the most recent successful content refresh

6.2.7 Configuring Webhook URL Configure a webhook endpoint for Tilt-to-Act and automation:

1. Navigate to “Manage Devices” -> Find your Display in the list
2. Find the “Webhook URL” property
3. Enter the full URL of your webhook endpoint (e.g., `https://your-server.com/webhook/display-action`)
4. Save the changes

When the display’s tilt button is triggered, it sends an HTTP POST request to this URL with device identification data. This enables integration with smart home systems like Home Assistant, Node-RED, or custom automation servers.

Display: TD7C-202605-0000000001

Broadcasting update

Display name

TD7C-202605-0000000001

Template

Layout 1 (Horizontal)

Lorem Ipsum

Last Content Update

3.8.2026, 18:05:15

Gateway

GWWBE-202604-0000000003

Data Binding

Unassigned

Power History

Energy used for rendering

7.47 %

Energy used for e-paper update

2.36 %

Firmware

0.8.0

Physical orientation

Vertical

Webhook URL

Enter a value

7. Layout Designer

The Layout Designer is a WYSIWYG (What You See Is What You Get) editor that allows you to create display layouts visually without writing code. It runs entirely in your browser using WebAssembly to render the same LVGL engine that powers the device firmware, ensuring pixel-perfect previews.

7.1 Basics

7.1.1 Creating a Layout (Horizontal/Vertical) To create a new layout:

1. Navigate to **Layout Designer** from the main menu
2. Click “Add Layout”
3. Enter a name for the layout
4. Choose orientation: **Horizontal** (landscape, 800×480) or **Vertical** (portrait, 480×800)
5. Click “Create”
6. A new entry appears. Click “Edit” to start the design

The canvas represents your display screen at actual resolution. You can drag controls from the palette onto the canvas and position them freely.

7.1.2 Renaming and Deleting Layouts **Renaming:** - Click on the layout name in the header - Type a new name - Press Enter to confirm

Deleting: - Select the layout in the sidebar - Click the trash icon - Confirm the deletion dialog

> Deleting a layout is permanent. If the layout is currently in use by a display, it can not be deleted.

7.1.3 Saving and Publishing The Layout Designer uses a two-stage workflow:

Save (Draft): Click “Save” to store your current work as a draft. Draft layouts are not visible on displays and can be edited freely.

Publish: Once satisfied with the layout, click “Publish” to make it available for assignment to displays. Published layouts become read-only; create a new version if you need further changes.

Displays will automatically receive the published layout at their next refresh cycle.

Discard changes:

7.1.4 Real-Time Preview (WebAssembly Rendering in Browser) The Layout Designer features a real-time preview powered by WebAssembly:

- The same LVGL rendering engine that runs on the display is compiled to WebAssembly and runs in your browser
- Every change you make appears instantly in the preview pane with pixel-perfect accuracy
- Dithering effects, font rendering, and color quantization match exactly what will appear on the e-paper display, if you enable dithering by clicking “Enable dithering”
- A physical device is not required to design layouts

This technology ensures that what you see in the editor is exactly what your display will show.

7.2 Controls (Widgets)

The Layout Designer provides a comprehensive set of controls based on LVGL widgets. Drag them from the palette onto the canvas.

7.2.1 Text Simple text display with configurable font, size, and color. Use Labels for static text like titles and headers. Use Spangroup when you need mixed formatting in a single line.

Property	Type	Default	Description
Text	string	`"Lorem Ipsum"`	Display text (supports data binding)
Font	string	`"DavidLibre-Regular"`	Font family
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Text color (hex code)
Text Align	enum	`AUTO`	AUTO, LEFT, CENTER, RIGHT
Text Decoration	enum	`NONE`	NONE, UNDERLINE, STRIKETHROUGH

7.2.2 Image Static image display supporting PNG, JPG, and SVG formats. Images are uploaded to the web backend and referenced by ID in your layout. The system automatically optimizes images for e-paper rendering with dithering.

Property	Type	Default	Description
Source	file	`"icons/traffic-cone.svg"`	Image file (file picker, PNG/JPG/SVG)
Rotation	number	`0`	Rotation angle in degrees
Scale	number	`256`	Scale factor (256 = 100%)
Inner Align	enum	`CONTAIN`	Display mode: DEFAULT, TOP_LEFT, CENTER, STRETCH, CONTAIN, COVER, TILE, etc.

7.2.3 Calendar Displays a monthly calendar view with configurable current date highlighting, weekday header customization, and support for data binding to dynamic event markers.

Property	Type	Default	Description
Show today's date	boolean	`false`	Highlight today's date (sets year/month/day to current date)
Font	string	`"DavidLibre-Regular"`	Font family
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Text color (hex code)

7.2.4 Progress Bar Linear progress bar with configurable fill percentage and color. Supports data binding for dynamic values from JSON/XML sources.

Property	Type	Default	Description
Value	number	`50`	Current value of the progress bar
Min	number	`0`	Minimum value of the range
Max	number	`100`	Maximum value of the range
Color	color	`"#0000FF"`	Fill indicator color (hex code)

7.2.5 Chart Renders line or bar charts with configurable number of series and data points, axis labels, and grid lines. Supports real-time data updates via data binding.

Property	Type	Default	Description
Font	string	`"DavidLibre-Regular"`	Font for axis labels
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Label text color (hex code)
Chart Data (Array)	string (JSON)	`null`	JSON array of data points, e.g., `[1,2,3,4]` (supports data binding)
Series Red	number	`255`	Red channel of chart line
Series Green	number	`0`	Green channel of chart line
Series Blue	number	`0`	Blue channel of chart line

7.2.6 Checkbox Standard checkbox with label text. Primarily visual indicators on the e-paper display, as the display does not have touch input.

Property	Type	Default	Description
Text	string	`"Lorem Ipsum"`	Label text (supports data binding)
Font	string	`"DavidLibre-Regular"`	Font family
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Text color (hex code)
Text Align	enum	`AUTO`	AUTO, LEFT, CENTER, RIGHT
Text Decoration	enum	`NONE`	NONE, UNDERLINE, STRIKETHROUGH
Checked	boolean	`false`	Enables/disables the checked state

7.2.7 LED Indicator A small circular LED indicator that can be colored. Configurable size and color. Useful for status indicators (e.g., system health, alerts).

Property	Type	Default	Description
Color	color	`"#000000"`	LED color (hex code, split into RGB components)

7.2.8 QR Code Generates a QR code from text or URL data. Automatically encodes the provided string into a scannable QR code with configurable size and error correction level. Supports data binding for dynamic URLs.

Property	Type	Default	Description
Dark Color	color	`"#000000"`	Color of dark QR code modules (hex code)
Light Color	color	`"#FFFFFF"`	Color of light QR code modules (hex code)
Text	string	`"Lorem Ipsum"`	Text/URL to encode (supports data binding)

7.2.9 Scale Measurement scale with tick marks and labels. Configurable value range and rotation angle for various display orientations.

Property	Type	Default	Description
Min	number	`0`	Minimum value of the scale
Max	number	`100`	Maximum value of the scale
Rotation	number	`0`	Rotation angle in degrees
Font	string	`"DavidLibre-Regular"`	Font for labels
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Marker text color (hex code)

7.2.10 Switch Toggle switch (on/off) visual element. Primarily visual indicators on the e-paper display, as the display does not have touch input.

Property	Type	Default	Description
Checked	boolean	`false`	Enables/disables the on/off state
Color	color	`"#0000FF"`	Active indicator color (hex code)

7.2.11 Text Field Multi-line text area with configurable number of visible lines, scrollable content for longer text, and support for data binding to dynamic multi-line content.

Property	Type	Default	Description
Text	string	`"Lorem Ipsum"`	Display text (supports data binding)
Font	string	`"DavidLibre-Regular"`	Font family
Font Size	number	`22`	Font size in pixels
Text Color	color	`"#000000"`	Text color (hex code)
Text Align	enum	`AUTO`	AUTO, LEFT, CENTER, RIGHT
Text Decoration	enum	`NONE`	NONE, UNDERLINE, STRIKETHROUGH

7.3 Control Properties

Every control has configurable properties accessible in the right-side property panel when selected on the canvas.

7.3.1 Position and Size (X, Y, Width, Height)

- **X / Y** - Top-left corner position in pixels
- **Width / Height** - Dimensions in pixels
- Controls can be resized by dragging their edges on the canvas
- Snap-to-grid alignment assists with precise positioning

7.3.2 Rotation and Scale

- **Rotation** - Rotate the control by a specified angle (0-360°)
- **Scale X / Y** - Uniform or non-uniform scaling factor (1.0 = original size)

Useful for creating angled elements or resizing images proportionally.

7.3.3 Colors (Text Color, Background Color, Border Color, Opacity)

- **Text Color** - Font color (hex code or color picker)
- **Background Color** - Fill color behind the control
- **Border Color** - Outline color with configurable border width
- **Opacity** - Transparency level (0% = fully transparent, 100% = opaque)

Note: E-paper displays have limited color support. Colors are converted to grayscale or dithered patterns during rendering. The preview shows the actual result.

7.3.4 Font and Font Size

- **Font Family** - Select from available system fonts (e.g., DejaVu Sans, Noto Sans)
- **Font Size** - Text size in pixels
- **Bold / Italic** - Text style modifiers (available through Spangroup control)

The system includes a comprehensive font library optimized for e-paper readability.

7.3.5 Alignment and Text Decoration

- **Horizontal Alignment** - Left, Center, Right
- **Vertical Alignment** - Top, Middle, Bottom
- **Text Decoration** - Underline, Strikethrough

7.3.6 Enabling/Disabling Dithering The e-ink display supports only 6 color levels. The dithering setting in the preview controls how colors are rendered in the layout designer:

- **Dithering disabled** - Colors are displayed at 16-bit color depth. This reflects the original color definition but does not show the actual display result.
- **Dithering enabled** - Colors are shown as they will appear on the display. The dithering algorithm simulates the limited 6-color palette of the e-ink display using patterns of black and white pixels.

Note: On the actual display, dithering is always enabled regardless of the designer setting. Enable dithering in the preview for a realistic representation of your layout.

7.4 Managing Images

7.4.1 Adding Images To add an image to your layout:

1. **Drag & Drop** the **Image icon** from the controls panel onto the canvas
2. Select the placed image element on the canvas to open its properties in the right panel
3. Set the desired image in the **Source** property:
   - **Icon library (balloon icon)** – Opens the icon picker where you can select an existing icon from the built-in library
   - **Upload file (upload icon)** – Opens the file dialog to upload an image file from your computer (PNG, JPG, or SVG)

7.4.2 Image Quality and Scaling When uploading JPG images, a popup appears allowing quality scaling. This lets you adjust the file size directly during upload to stay within the limit. The system automatically optimizes images for e-paper rendering: - Large images are scaled down to fit within the display resolution (800×480) - Aspect ratio is preserved during scaling

7.4.3 File Size and Resolution Limits Images must not exceed a maximum file size of **64 KB** and the resolution must be at most **800×480 pixels**:

Format	Maximum File Size	Maximum Resolution
PNG	64 KB	800×480 pixels
JPG	64 KB	800×480 pixels
SVG	64 KB	Vector (resolution-independent)

Use the JPG scaling in the upload popup to reduce large images to the desired file size. For PNG and SVG files, optimize the file before uploading.

7.4.4 Supported Formats (PNG, JPG, SVG)

- **PNG** - Best for graphics with transparency or sharp edges
- **JPG** - Best for photographs and complex gradients
- **SVG** - Best for logos, icons, and scalable vector graphics

7.5 Data Binding

Data binding allows you to insert dynamic values from data sources into your layout controls.

7.5.1 Inserting Dynamic Values To bind a control to a data field:

1. Select the control on the canvas
2. In the property panel, find the “Text” or “Value” property
3. Click the data binding icon ({}) next to the property
4. Select the data source and field from the dialog

7.5.2 Selecting Data Fields The data binding dialog shows: - Available data sources assigned to your account - Field hierarchy within each data source (JSON/XML structure) - Preview of current values for verification

7.5.3 Syntax and Placeholders Data binding uses **Lupa**, a Jinja2 template engine implementation written in Lua, which supports Lua syntax within tags and variables.

Data binding uses placeholder syntax in text content:

```
{{dataSource.fieldName}}
```

Examples: - `{{weather.temperature}}` - Inserts the temperature value from a weather data source - `{{schedule.nextMeeting}}` - Shows the next meeting time - `Current temperature: {{sensors.temp}}°C` - Mixed static and dynamic text

Nested fields are accessed using dot notation. Array elements can be referenced by index (e.g., `{{data.items[0].name}}`).

Since Lupa supports Jinja2 syntax, advanced template features are also available, such as conditionals (`{% if %}`), loops (`{% for %}`), and filters. Lua code can also be used within tags.

7.6 Undo/Redo

The Layout Designer supports full undo/redo functionality:

- **Undo** - Click the undo button in the toolbar
- **Redo** - Click the redo button

All actions including control placement, property changes, and image uploads can be undone. The undo history is maintained during your editing session.

8. Data Sources

Data sources provide dynamic content for your display layouts. Instead of static text and images, you can bind layout controls to data that updates automatically from external systems, APIs, or manual input.

8.1 Creating a Data Source

To create a new data source:

1. Navigate to **Data Sources** from the main menu
2. Click “Add Data Source”

3. Enter a display name
4. Click “Edit” on the newly created element
5. Paste or type your data into the content editor
6. Click “Update” to save

Displays are bound to data sources (not layouts). When a display is bound to a data source and the layout assigned to that display is edited, the data source becomes automatically available in the Layout Designer. Alternatively, content can be set manually directly in the designer — however, this must be repeated every time the layout is edited.

8.2 JSON Data Format

JSON (JavaScript Object Notation) is the recommended format for data sources due to its simplicity and wide API support:

```
{
  "temperature": 23.5,
  "humidity": 65,
  "weather": "Partly Cloudy",
  "forecast": [
    { "day": "Monday", "temp": 24 },
    { "day": "Tuesday", "temp": 22 }
  ]
}
```

Binding examples in layouts: - `{{temperature}}` -> Displays `23.5` - `{{weather}}` -> Displays `Partly Cloudy` - `{{forecast[0].day}}` -> Displays `Monday`

JSON supports nested objects, arrays, strings, numbers, booleans, and null values. The display firmware parses JSON efficiently with minimal memory overhead.

8.3 XML Data Format

XML is supported for compatibility with legacy systems and enterprise data feeds:

```xml
<data>
  <temperature>23.5</temperature>
  <humidity>65</humidity>
  <weather>Partly Cloudy</weather>
  <forecast>
    <item><day>Monday</day><temp>24</temp></item>
    <item><day>Tuesday</day><temp>22</temp></item>
  </forecast>
</data>
```

Binding examples in layouts: - `{{temperature}}` -> Displays `23.5` - `{{weather}}` -> Displays `Partly Cloudy` - `{{forecast.item[0].day}}` -> Displays `Monday`

The display firmware includes an XML parser (xml2lua) that converts XML to a Lua table structure for data binding access.

8.4 Manual Data Updates

To update a data source manually:

1. Navigate to **Data Sources**
2. Click “Edit” on the desired data source
3. Modify the content in the editor
4. Click “Update” to save the changes

The display will automatically reflect the updated data at its next refresh cycle (typically within 5 minutes when energy is available). Manual updates are useful for: - Static information that changes infrequently (e.g., opening hours, announcements) - Testing layout designs with sample data before connecting to an API - Emergency content overrides

8.5 Automatic Updates via API/Webhook

For automated data updates, use the REST API or webhook integration:

Update via cURL:

```bash
curl -X PUT "https://api.siebenschlaefer.com/v1/datasources/{id}/content" \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"temperature": 24.1, "humidity": 62}'
```

Update via PowerShell:

```powershell
Invoke-RestMethod -Uri "https://api.siebenschlaefer.com/v1/datasources/{id}/content" `
  -Method Put `
  -Headers @{ Authorization = "Bearer YOUR_API_TOKEN" } `
  -ContentType "application/json" `
  -Body '{"temperature": 24.1, "humidity": 62}'
```

The API accepts both JSON and XML content. Replace the example payload with your actual data. The display will automatically update once it refreshes.

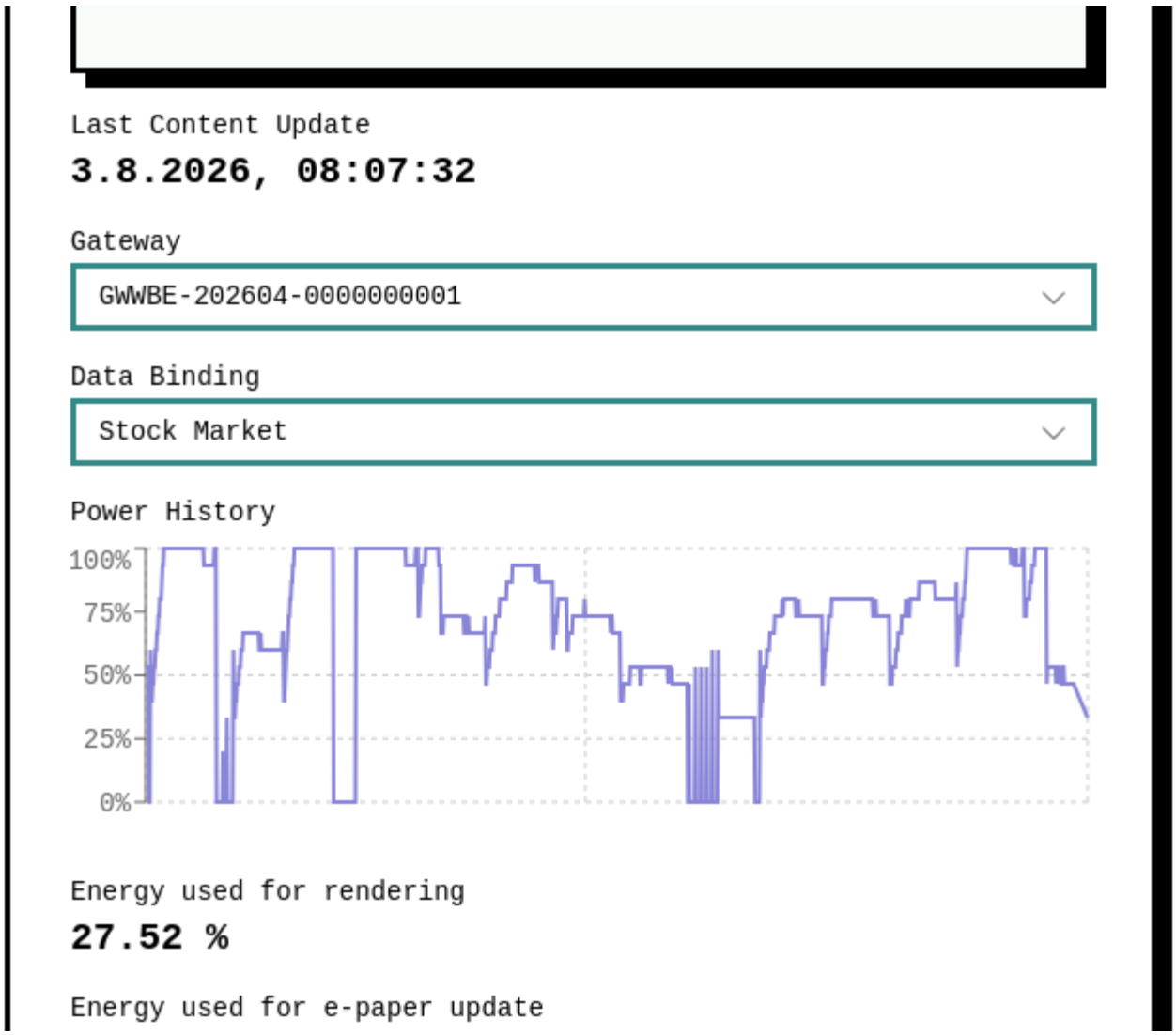
> For detailed API documentation including authentication, rate limits, and all endpoints, see Chapter 16 (API Reference).

8.6 Assigning Data Sources to Multiple Displays

A single data source can be shared across multiple displays:

1. Create the data source once in **Data Sources**
2. In each layout that uses this data, bind controls to the same data source fields
3. Assign the layouts to different displays through device configuration

When you update the data source content (manually or via API), all displays using it will reflect the changes at their next refresh cycle. This is ideal for: - Company-wide announcements displayed on multiple screens - Shared metrics dashboards across departments - Consistent branding with centrally managed images and text



9. Lua Scripting

Lua scripting provides advanced customization capabilities beyond the visual Layout Designer. With Lua, you can create dynamic layouts, process data sources programmatically, draw custom graphics, and implement complex logic directly on the device.

9.1 Introduction to Lua on the Siebenschläfer Display

The Siebenschläfer™ Display includes a built-in Lua scripting engine (Lua 5.x) that runs directly on the STM32L496 microcontroller:

- **Full Lua language support** - Variables, functions, loops, conditionals, tables

- **LVGL bindings** - Direct access to LVGL graphics primitives for custom rendering
- **JSON/XML parsing** - Built-in libraries for processing data sources
- **Limited resources** - Scripts run within the device's memory constraints (4 MB RAM)

Lua scripts are transmitted from the web backend via the gateway and executed on-device during the layout rendering phase. This approach keeps bandwidth low while enabling powerful customization.

9.2 Embedding Lua Scripts in Layouts

Every layout has a Lua script that controls all its logic. To open the script editor:

1. In the Layout Designer, click **"Lua Script Settings"** ("Lua-Skript Einstellungen")
2. The script editor panel opens directly and displays the complete Lua code

The script structure is organized as follows: - **Upper section:** All elements you placed in the visual Layout Designer are automatically converted to Lua code and displayed here. This allows you to learn the correct syntax and API usage by observing the auto-generated code. - **Lower section:** Starting from the marker `-- Add additional commands below`, you can add your own custom Lua code. Here you can directly modify, manipulate, or extend the elements created by the Layout Designer with additional logic.

This approach enables you to learn LVGL syntax through the automatically generated code while maintaining full control over the layout by programmatically adjusting elements. Scripts can: - Create custom widgets not available in the visual designer - Process data source content before binding - Draw complex graphics using LVGL primitives - Implement conditional logic for dynamic layouts

9.3 Practical Example: Displaying Gold Daily Price Using n8n

In this section, we will create a complete example: Retrieving the current gold daily price via n8n, processing it, and displaying it on the Siebenschläfer display.

Step 1: Create n8n Workflow Create a new workflow in n8n with the following three steps:

Step 1 – Retrieve Gold Data (HTTP Request)

Add an **HTTP Request** node and configure it as follows:

Setting	Value
Method	GET
URL	`https://charting.stock3.com/d/q?iid=133979&res=`
Authentication	None
Send Body	No

> **License Notice:** Only use data sources that are legally permitted for your intended use. The endpoint mentioned above may require a paid account. Always review the terms of service of the respective provider and ensure you have the necessary permissions before using the workflow in production.

This step downloads the raw gold price data. The response contains arrays for Open (`o`), High (`h`), Close (`c`) and Low (`l`) values.

Step 2 – Process Data (Code)

Add a **Code** node (JavaScript) to transform the raw data into the desired format:

```
// Get data from API response
const data = items[0].json.data;

const o = data.o || [];
const h = data.h || [];
const c = data.c || [];
const l = data.l || [];
```

```
let cur = 0;
const resultItems = [];

for (let i = 0; i < o.length; i++) {
    cur = cur + o[i];

    resultItems.push({
        json: {
            open: cur / 100,
            low: (cur - l[i]) / 100,
            high: (cur + h[i]) / 100,
            close: (cur + h[i] - c[i]) / 100
        }
    });

    cur = cur + h[i] - c[i];
}

// Take only the last 60 entries
const last60 = resultItems.slice(-120);

// Calculate date (yesterday)
const yesterday = new Date();
yesterday.setDate(yesterday.getDate());

// Format as DD.MM.YYYY
const day = String(yesterday.getDate()).padStart(2, '0');
const month = String(yesterday.getMonth() + 1).padStart(2, '0');
const year = yesterday.getFullYear();
const dateStr = `${day}.${month}.${year}`;

// Create JSON object and stringify it
const payloadObject = {
    symbol: "GOLD",
    date: dateStr,
    data: last60.map(item => item.json)
};

return [{
    json: {
        payload: JSON.stringify(payloadObject)
    }
}];
```

This code calculates the actual price values from the incremental raw data, limits them to the last 120 data points, and packages everything into a JSON object with symbol, date, and data table.

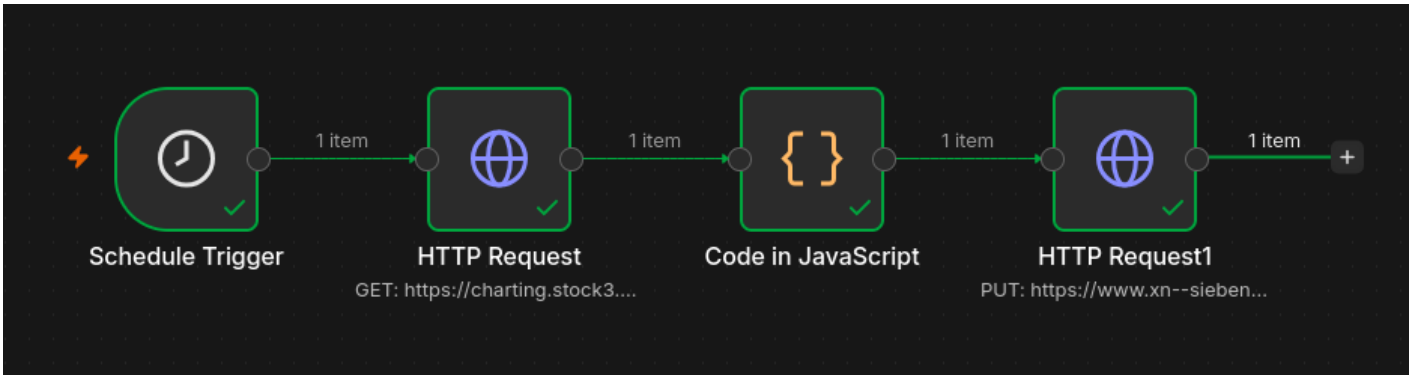
Step 3 – Send to Display (HTTP Request)

Add a second **HTTP Request** node to send the processed data to your data source in the Siebenschläfer system:

Setting	Value
Method	PUT
URL	`https://www.siebenschläfer-display.eu/api/databindings/update/{ResourceLin`
Authentication	None
Send Query Parameters	No
Send Header	Yes
Specify Headers	Using Fields Below

Setting	Value
Header	`Content-Type: application/json`
Send Body	Yes
Body Content Type	JSON
Specify Body	Using JSON
JSON	`{{ JSON.stringify($json.payload) }}`

Replace `{ResourceLink}` and `{Token}` with your actual values. You receive these when opening your data source on the website – the update endpoint is displayed there.



Save the workflow and test it. On successful execution, your data source now contains the gold price data in the following format:

```
{
  "symbol": "GOLD",
  "date": "28.07.2026",
  "data": [
    {"open": 3412.5, "low": 3405.2, "high": 3420.8, "close": 3418.0},
    ...
  ]
}
```

In the next step, we will create the layout on the display to present this data in an appealing way.

Step 2: Create Layout in Designer Open the Layout Designer and create a new horizontal layout. The gold price layout consists of the following elements:

1. **SVG Icon** (Image) – A predefined candlestick chart icon as a visual identifier
2. **Symbol Label** (Label) – Displays the ticker symbol (“GOLD”), bound to `{{symbol}}`
3. **Date Label** (Label) – Shows the data date so you can quickly verify the information is current, bound to `{{date}}`
4. **Scale** (Scale) – Y-axis with dynamic range for price values
5. **Candlestick Chart** – Implemented entirely in custom Lua code

Place the first four elements visually in the designer:

Element	Position (X, Y)	Width	Height	Properties
Symbol Label	77, 17	189	59	Text: `{{symbol}}`, Font: DejaVuSansMono-Bold, 50px
Date Label	649, 12	143	31	Text: `{{date}}`, Font: DejaVuSansMono-Regular, 24px

Element	Position (X, Y)	Width	Height	Properties
SVG Image	22, 19	51	51	Source: `candlestick-chart.svg`, Scale: 256, Inner Align: Center
Scale	66, 102	41	295	Range: 43000–51000 (dynamically adjusted later), Rotation: 1×10°, Font: DavidLibre-Regular, 22px

Data binding for the labels is done via the `{{}}` symbol in the Text property. The scale receives a preliminary value range that we will dynamically adjust in the Lua script.

Step 3: Write Custom Lua Script Click **"Lua Script Settings"** to open the script editor. The upper section shows the auto-generated code for the elements placed above. Add the following custom code starting from the `-- Add additional commands below` marker:

```lua
-- Add additional commands below
scale_3:set_mode(2)

num_elements = #_G.dormouse.bindingData.data
print_elements = num_elements

local min_val = nil
local max_val = nil

for k = 1, num_elements do
    v = _G.dormouse.bindingData.data[k]

    if min_val == nil then
        min_val = v.low
        max_val = v.high
    else
        if v.open < min_val then min_val = v.open end
        if v.close < min_val then min_val = v.close end
        if v.low < min_val then min_val = v.low end

        if v.open > max_val then max_val = v.open end
        if v.close > max_val then max_val = v.close end
        if v.high > max_val then max_val = v.high end
    end
end

padding = (max_val - min_val) * 0.05
low_line = min_val - padding
high_line = max_val + padding

scale_3:set_range(math.floor(low_line), math.floor(high_line))

x_start = 120

for k2 = num_elements-print_elements+1,num_elements,1 do
    v = _G.dormouse.bindingData.data[k2]
    k = k2 - num_elements+print_elements
```

```
    open = v.open
    close = v.close

    y_open = (open - low_line) / (high_line - low_line) * 300
    y_close = (close - low_line) / (high_line - low_line) * 300

    -- Calculate wick (from low to high)
    wick_y_low = (v.low - low_line) / (high_line - low_line) * 300
    wick_y_high = (v.high - low_line) / (high_line - low_line) * 300
    wick_height = math.abs(wick_y_low - wick_y_high)

    -- Draw wick (before the body so it is not covered)
    wick_i = lvgl.Line(nil)
    wick_x = x_start + k*5
    wick_y_start = 400 - math.max(wick_y_low, wick_y_high)
    wick_i:set_pos({x = wick_x+1, y = wick_y_start})
    wick_i:set({width = 2, height = wick_height, text= "", bg_color = "#000000", bg_opa = 255})

    -- Candle body
    label_i = lvgl.Line(nil)

    if (close > open) then
        label_i:set_pos({x = x_start+k*5, y = 400 - y_close});
        label_i:set({width = 4, height = y_close-y_open, text= "", bg_color = "#00ff00", bg_opa = 25
    else
        label_i:set_pos({x = x_start+k*5, y = 400 - y_open});
        label_i:set({width = 4, height = y_open-y_close, text= "", bg_color = "#ff0000", bg_opa = 25
    end
end
```

How the script works in detail:

1. **Set scale mode:** `scale_3:set_mode(2)` sets the scale to vertical mode.
2. **Calculate min/max values:** The first loop through all data points determines the lowest (`min_val`) and highest (`max_val`) value from all Open, Close, High, and Low values.
3. **Dynamic range with padding:** A 5% padding is added so candles don't clip at the edges. Then the scale range is dynamically adjusted: `scale_3:set_range(...)`.
4. **Draw candles:** The second loop creates for each data point:
   - **Wick:** A black line (`lvgl.Line`) from Low to High, drawn before the candle body
   - **Candle body:** A colored line – green (`#00ff00`) when Close > Open (rising); red (`#ff0000`) when Close < Open (falling)
5. **Positioning:** Y-position is calculated relative to the dynamic range (`low_line` to `high_line`) and scaled to display height (300px). X start position is at 120px, each candle is 5px wide.

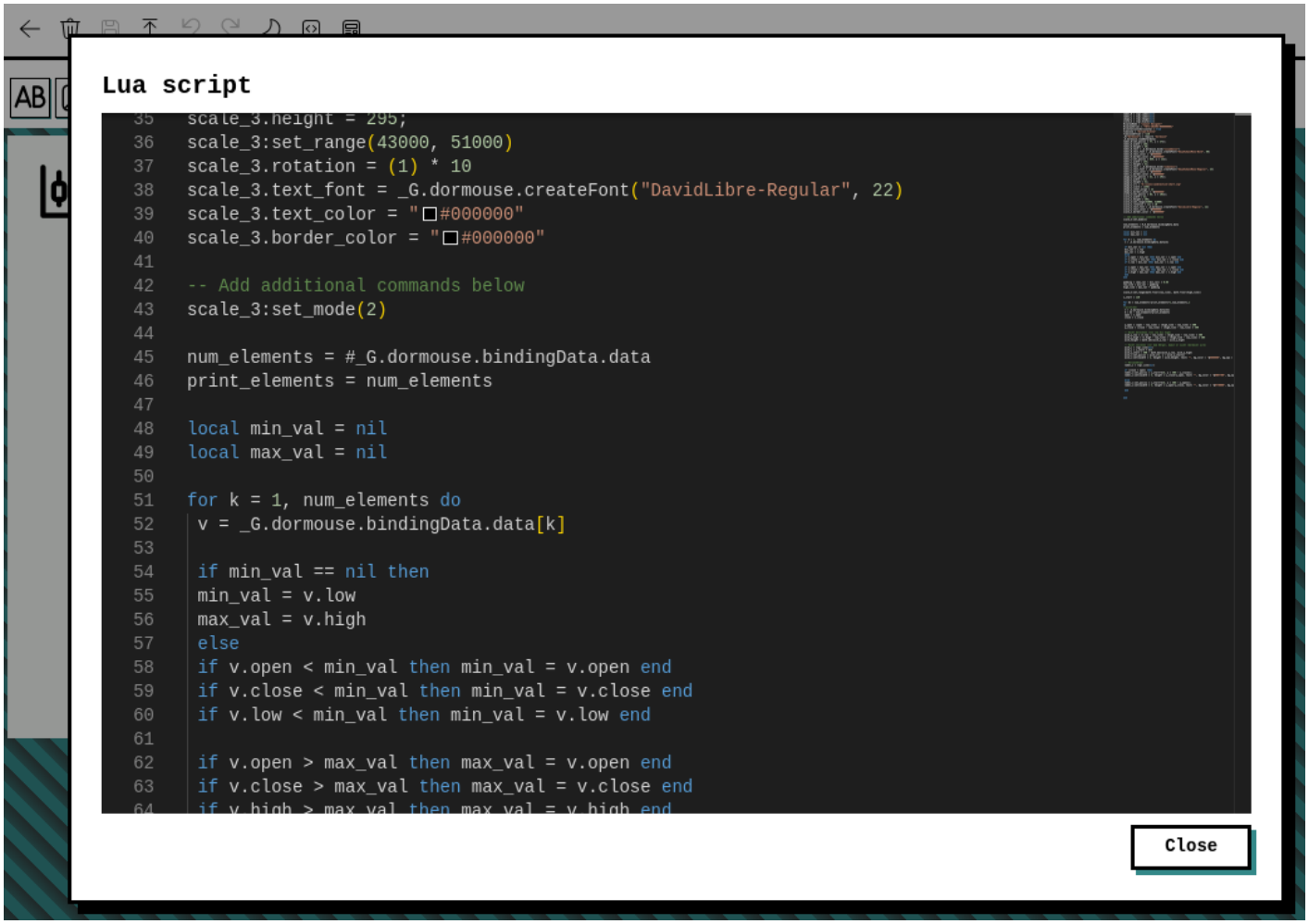
Save the layout and publish it. Assign it to the display bound to your gold price data source. The display now shows the current gold daily price as a candlestick chart with dynamic Y-axis, symbol label, and date.

9.4 Debugging and Troubleshooting

Debugging Lua scripts on the device:

1. **Use `print()` statements** – Output messages appear in the browser console (F12)
2. **Check error state** – If a script fails, the display shows "Error rendering" status. This status is visible in the device overview on the website.
3. **Simplify incrementally** – Start with basic scripts and add complexity gradually

Common issues: - Memory limits - Keep scripts concise; avoid large data structures - Missing data fields - Always check for nil values before accessing data source fields - LVGL API differences - Some LVGL functions may have limited Lua bindings



10. Webhooks and Automation

The Siebenschläfer™ Display can trigger external actions through webhooks, enabling integration with smart home systems, automation servers, and custom applications.

10.1 Processing Tilt-to-Act Button Presses with n8n

> The webhook is called by the **Gateway** (Hase), not directly by the display. The gateway receives the button press via LoRa from the display and forwards it as an HTTP POST to the configured webhook URL.

In n8n, you create a **Webhook** node that acts as the receiver for display button press events. The webhook node provides a test endpoint that looks approximately like this:

`http://192.168.0.3/webhook-test/c269e3c5-e4d7-40de-9056-8b0e80df3490`

Enter this endpoint as the **Webhook URL** in the display configuration (see Chapter 6.2.7).

Configure Webhook Node

1. Add a **Webhook** node to your n8n workflow
2. Configure the properties: | Property | Value | |———-|——-| | Method | `POST` | | Path | `display-tilt` (or your desired path) | | Respond | `Immediately` (to proceed with the workflow) |
3. Save the node and activate the workflow
4. Copy the complete webhook URL from the node

JSON Payload for Tilt-to-Act Button Presses The display has **two physical buttons** (left and right) that can be triggered by tilting. On each button press, the **gateway** sends an HTTP POST request to the configured webhook URL with the following JSON payload:

```
{
  "leftButton": true,
  "rightButton": false
}
```

The fields have the following meaning:

Field	Description
`leftButton`	`true` if the left button was pressed, otherwise `false`
`rightButton`	`true` if the right button was pressed, otherwise `false`

11. Power and Performance

This chapter covers everything related to the Siebenschläfer™ Display's solar power system, energy management, and performance optimization.

11.1 Understanding Solar Power

The Siebenschläfer™ Display operates entirely on solar energy, requiring no electrical connection or battery replacement:

- **Solar frame** - A thin solar cell strip surrounds the e-paper display area, continuously harvesting ambient light
- **Supercapacitors** - Collected energy is stored in high-capacity supercapacitors (not batteries), providing instant power for display refreshes
- **No degradation** - Unlike batteries, supercapacitors do not degrade over time and have virtually unlimited charge/discharge cycles
- **Zero maintenance** - The system requires no battery changes or electrical connections

The energy flow works as follows: ambient light -> solar cells -> charging circuit -> supercapacitors -> display refresh. When sufficient energy is accumulated, the display wakes from deep sleep, contacts the gateway, receives new content, renders it, and returns to sleep.

11.2 Minimum Light Conditions (Lux Values)

The display's update frequency depends directly on ambient light levels:

11.2.1 200 Lux - 1 Refresh per Day At approximately 200 lux of ambient light, the display can accumulate enough energy for one full refresh cycle per day. This level is typical of: - Dimly lit hallways or corridors - Rooms with small windows far from the display - Heavily overcast days indoors

11.2.2 400 Lux - 1 Refresh every 3 Hours At approximately 400 lux, the display can refresh roughly every 3 hours. This level is typical of: - Well-lit office environments with overhead lighting - Rooms near windows during daytime - Retail stores with standard commercial lighting without large storefront windows

11.2.3 Darker Environments with Extension Panel For locations with insufficient ambient light (below 200 lux), the optional Extension Panel provides additional solar harvesting area:

- Doubles the available solar cell surface area
- Enables operation in lower-light environments
- Connects via a short cable to the main display unit
- Maintains the same wireless, self-powered design

11.3 Refresh Rate and Power Consumption

The energy required for each refresh cycle depends on several factors:

Factor	Impact on Energy
Layout complexity	More controls = more rendering time = higher energy use
Resource count	Each image requires additional data transfer and processing

Optimization tips: - Use simple layouts with fewer controls for faster, lower-energy refreshes - Prefer text over images where possible (text rendering is very efficient)

11.4 Interpreting Charge History

The charge history chart in “Manage Devices” shows your display’s energy levels over time:

- **Rising curve** - Display is collecting solar energy (good light conditions)
- **Flat line at high level** - Supercapacitors are fully charged and maintaining
- **Sharp drops** - Content refresh cycles consuming stored energy
- **Gradual decline** - Insufficient light; consider relocating or adding Extension Panel

A healthy display shows a sawtooth pattern: gradual rise during charging, sharp drop during refresh, repeat. If the curve stays low without reaching full charge, the location lacks sufficient light.

11.5 Standby Power Consumption (100 µW)

The Siebenschläfer™ Display achieves ultra-low standby power consumption of approximately 100 microwatts (µW):

- **Shutdown mode** - The STM32L496 microcontroller enters shutdown mode with active RTC, which periodically wakes it up
- **RTC wake-up cycle** - The real-time clock wakes the controller approximately every 5 minutes for periodic gateway check-ins
- **DCDC converter in duty cycle** - The DCDC converter also runs in duty cycle operation to keep power consumption low enough

At 100 µW standby consumption and typical supercapacitor capacity, the display can remain in standby for about 4 days without any light exposure before energy is depleted. This ensures reliability even during extended periods of darkness (e.g., holidays, nighttime).

12. Communication and Network

This chapter explains the communication technology behind the Siebenschläfer™ system, from LoRa radio to website connectivity and security.

12.1 LoRa Radio Technology (433 MHz, E22M22S)

The display communicates with the Hase Gateway using LoRa (Long Range) radio technology via the E22-400M22S module:

- **Frequency** - 433 MHz ISM band (Europe), providing good penetration through walls and obstacles
- **Power output** - 10 dBm, enabling long-range communication with minimal power consumption
- **Modulation** - LoRa spread spectrum modulation for robust interference resistance

12.1.1 Range (7 km Line-of-Sight / ~3 Thick Walls) The effective range depends on the environment:

Environment	Approximate Range
Line of sight (outdoor)	Up to 7 km
Open office space	50-100 m

Environment	Approximate Range
Residential building	Through ~3 thick walls
Industrial/RF-noisy	Reduced range, depends on interference

For large installations with multiple floors or buildings, deploy additional gateways to ensure coverage.

12.1.2 Adaptive Data Rate (ADR) The LoRa module supports Adaptive Data Rate, which adjusts transmission speed based on signal quality:

- **Strong signal** - Higher data rate for faster content transfer
- **Weak signal** - Lower data rate with higher spreading factor for reliability
- **Automatic adjustment** - The system continuously monitors link quality and adapts without manual intervention

This ensures reliable communication even at the edge of range, trading speed for robustness when needed.

12.2 Smart Heartbeat - Time Slot Algorithm

The Siebenschläfer™ system uses an intelligent "Smart Heartbeat" protocol to manage communication between displays and gateways efficiently:

12.2.1 Millisecond-Precise Clock Synchronization Each display synchronizes its internal RTC (Real-Time Clock) with the gateway to millisecond precision:

- The gateway provides accurate time via NTP (Network Time Protocol) from internet sources
- Displays synchronize their clocks during each communication cycle
- This ensures all devices share a common time reference for coordinated wake-ups

12.2.2 Collision Avoidance with Hundreds of Devices In installations with many displays, simultaneous transmission attempts could cause packet collisions. The Smart Heartbeat algorithm prevents this:

- Each display has a unique device key generated during manufacturing
- A deterministic hash function calculates an optimal time slot for each device
- Displays wake only in their assigned time window to check for updates
- Time slots are distributed evenly across the communication cycle

This approach allows hundreds of displays to share a single gateway without collisions, while maintaining ultra-low power consumption through predictable sleep schedules.

12.3 Encryption and Security

All communications in the Siebenschläfer™ system are encrypted end-to-end using modern cryptographic standards:

12.3.1 End-to-End Encryption (AES-GCM) All data transmitted between display, gateway, and web backend is encrypted using AES-GCM (Advanced Encryption Standard in GCM mode) with 256-bit keys. The GCM mode provides not only confidentiality through encryption, but also:

- **Integrity protection** - Every data block is accompanied by an authentication code (GCM tag) that detects and rejects any tampering
- **Replay attack protection** - Unique initialization vectors (IVs) prevent captured messages from being replayed
- **Authenticity guarantee** - The receiver can verify that the data originates from the expected sender and has not been modified

Encryption is applied across all communication layers:

- **Payload encryption** - Content updates, layout definitions, and data source content are AES-GCM encrypted
- **Transport security** - Gateway-to-website communication uses TLS 1.3
- **Radio encryption** - LoRa transmissions between display and gateway use AES-GCM with 256-bit keys

12.3.2 Digital Signatures (Ed25519) All firmware updates and critical configuration changes are digitally signed using Ed25519 elliptic curve signatures:

- Ensures authenticity of firmware packages
- Prevents unauthorized modifications to device configurations
- The display verifies the signature before applying any update

12.3.3 Key Exchange (RSA/X25519) Secure key exchange protocols establish encrypted communication channels:

- **X25519** - Used for LoRa radio key exchange between display and gateway
- **RSA** - Used for website API authentication and certificate management
- Keys are derived using modern SHA-based key derivation functions (HKDF)

12.3.4 Device-Specific Keys (Manufacturing) Each device receives unique cryptographic keys during manufacturing:

- Generated using a hardware TRNG (True Random Number Generator)
- Each display-gateway pair establishes an independent encryption channel
- Compromising one device does not affect the security of others

This per-device key architecture ensures that even at scale, each communication link is individually protected and authenticated.

13. Firmware Updates

Firmware updates keep your devices secure, add new features, and fix bugs. All firmware updates are fully automatic — no manual action is required. The goal of the Siebenschläfer™ system is complete maintenance-free operation.

13.1 Fully Automatic Firmware Updates

Both gateway and display firmware are updated automatically:

- **Gateway** - New firmware is sent directly from the web backend to the gateway and installed. The gateway reboots automatically after installation.
- **Display** - Firmware is distributed through the normal communication cycle. Each display receives, verifies (Ed25519 signature), and installs the update during its scheduled wake-up.

The entire process runs in the background without any action needed from you. Updates are transmitted encrypted and digitally signed to ensure security and integrity.

13.2 Checking Update Status

You can check the current firmware version at any time:

- **Gateway** - Press the info button on the gateway itself to display the current firmware version
- **Display** - The firmware version is visible in the device overview on the website

14. Gateway

The Hase Gateway features an integrated e-paper display (2.7-inch, 176×264 pixels) for local operation and status display. The user interface is controlled by two physical buttons and provides full access to all essential gateway functions without external tools.

14.1 Display and Buttons

The gateway display shows various screens navigable through two buttons:

Button	Function
Button A (left button)	Confirm / Next / Info
Button B (right button)	Back / Cancel / Cycle options
Both buttons simultaneously	Execute action (e.g., confirm reset)
Reset button (physical MCU reset)	MCU restart (direct hardware reset)

14.2 Screen: Devices Overview (Devices Overview Screen)

The main screen of the gateway displays an overview of all connected Siebenschläfer™ displays and the cloud connection status.

Displayed Information:

- **Cloud Status** - Shows `offline` when no connection to the web backend is established
- **List of Displays** - Status of each registered display, including signal strength and last communication time

Navigation:

Action	Button	Target Screen
Show Info	A (Info ↓)	Info Screen
Show Help	B (↓ Help)	Help Screen

14.3 Screen: Information (Info Screen)

The information screen shows detailed technical data about the gateway device.

Displayed Information:

- **Device Information** - Serial number and contact information
- **Firmware Version** - Currently installed firmware version of the gateway
- **Ethernet MAC Address** - MAC address of the Ethernet interface with status (Up/Down)
- **WiFi MAC Address** - MAC address of the WiFi interface with status (Up/Down)
- **IP Address and Subnet Mask** - Current IPv4 configuration
- **IPv6 Address** - If an IPv6 connection is active
- **Gateway and DNS** - Default gateway and DNS server

Navigation:

Action	Button	Target Screen
Next to Options	A (Next ↓)	Options Screen

14.4 Screen: Options (Options Screen)

The options screen provides access to gateway reset functions. Selection is done by pressing Button B to cycle through options, and execution by pressing both buttons simultaneously.

Available Options:

Option	Description	Trigger
Reset WiFi Settings	Clears stored WiFi configuration (WPA/EAP). The gateway restarts and attempts an Ethernet connection or shows the WiFi setup screen.	Press both buttons (option 0)
Factory Reset	Deletes all files from the SD card and resets the gateway to factory settings. The device then restarts.	Press both buttons (option 1)

Navigation:

Action	Button	Description
Back to Overview	A (Back ↓)	Devices Overview
Select option	B (↓ ↑ Select)	Cycles between reset options
Execute action	Both buttons	Executes the selected reset operation

The currently selected option is marked with a right arrow symbol (►).

> Factory reset deletes all data from the SD card, including stored firmware images and log files. Make sure you don’t need any important local backups before executing this option.

14.5 Screen: Help (Help Screen)

The help screen displays a QR code linking to the online documentation for the Siebenschläfer Display.

Displayed Information:

- **QR Code** - Links to `https://www.siebenschläfer-display.eu/` for documentation and support

Navigation:

Action	Button	Target Screen
Back to Overview	B (↓ Back)	Devices Overview

14.6 Screen: Register Gateway (Register Gateway Screen)

This screen appears after a successful network connection when the gateway is not yet registered with the web backend. It displays a QR code for gateway registration.

Displayed Information:

- **QR Code** - Contains the registration link

How it works: Scan the QR code with your smartphone or another device to complete registration on the website

14.7 Screen: WiFi Setup (WiFi Setup Screen)

This screen appears when no Ethernet connection is detected and no WiFi configuration is stored yet.

Navigation:

Action	Button	Description
Start WPS	A (WPS ↓)	Starts the WPS setup mode

> Ethernet is recommended as the preferred connection method, as it is more stable and reliable. Use WiFi only when no Ethernet port is available.

14.8 Screen: WPS Setup (WPS Screen)

This screen appears during an active WPS setup process.

Navigation:

Action	Button	Description
Cancel	B (↓ Cancel)	Exits the WPS setup mode and returns to the WiFi Setup screen

WPS Setup Flow:

1. Press the WPS button (A) on the WiFi Setup screen
2. The gateway switches to the WPS screen and activates WPS client mode
3. Within two minutes, press the WPS button on your router
4. The gateway automatically connects to the router's WiFi network
5. After successful connection, the gateway switches to the registration screen

   The WPS process has a time window of typically two minutes. Make sure to press the WPS button on your router within this window.

14.9 Screen: SD Card Missing (SD Missing Screen)

This error screen appears when the gateway does not detect an SD card at startup.

Displayed Information:

- **Error message** - "No SD card detected"

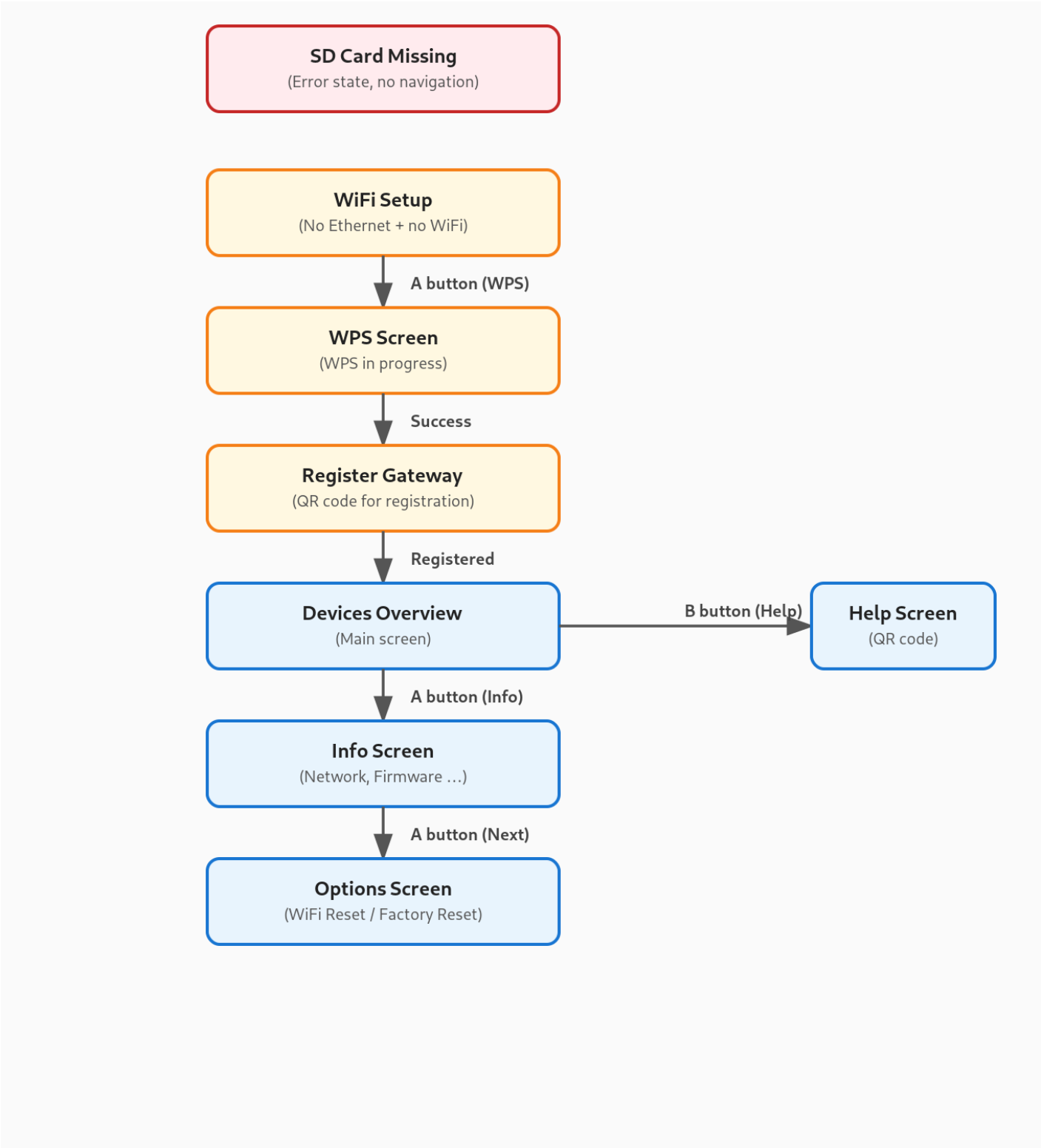
The SD card is required for gateway operation, as it stores firmware images and log files among other data. If this screen appears:

1. Power off the gateway (disconnect USB-C)
2. Check that the SD card is properly inserted
3. Try using a different microSD card
4. Power on the gateway again

   The gateway cannot operate without an SD card. Ensure a compatible microSD card (recommended: 32 GB) is correctly inserted.

14.10 Screen Hierarchy and Navigation

The following diagram shows the relationships between the individual gateway screens:



The physical MCU reset button triggers a direct device restart on every screen. The "↓ Reset" label refers to this hardware reset function.

14.11 Reset Functions in Detail

The gateway offers two levels of reset operations accessible through the options screen:

14.11.1 Reset WiFi Settings This option clears only the stored WiFi configuration data:

- Stored WPA/WPA2 credentials are removed

- EAP enterprise WiFi configuration is deleted
- The gateway restarts and attempts an Ethernet connection
- If no Ethernet is available, the WiFi setup screen appears

This option is useful when the WiFi password has changed or the gateway needs to be connected to a different network.

14.11.2 Factory Reset This option fully resets the gateway to its out-of-box state:

- All files on the SD card are deleted
- Gateway configuration is reset
- The device restarts and goes through the entire setup process from scratch

  Factory reset is an irreversible operation. All local data, including firmware caches and logs, will be lost. Use this option only when a complete reconfiguration is required.

15.1 Cleaning the Display and Solar Panel

To clean your Siebenschläfer™ Display:

1. Use a soft, dry microfiber cloth
2. For stubborn dirt, slightly dampen the cloth with water (no chemicals)
3. Gently wipe the e-paper surface and solar frame

Avoid: - Alcohol-based cleaners or solvents - Can damage the anti-glare coating - Abrasive materials - Can scratch the glass surface - Excessive moisture - Water ingress can damage internal electronics

15.2 Anti-Glare Glass - UV Protection and Care

The display's front glass features an anti-glare (AG) coating that: - Reduces reflections from overhead lighting and windows - Provides UV protection for the e-paper panel behind it

To preserve the AG coating, clean only with a soft microfiber cloth. Avoid pressure washing or high-pressure air cleaning.

16. Open Source Licenses

The Siebenschläfer Display System uses the following open-source components:

Backend (.NET)

- **BouncyCastle.Cryptography** – MIT (https://github.com/bcgit/bc-csharp)
- **Google.Protobuf** – BSD-3-Clause (https://github.com/protocolbuffers/protobuf)
- **Grpc.AspNetCore** – Apache-2.0 (https://github.com/grpc/grpc-dotnet)
- **MailKit** – MIT (https://github.com/jstedfast/MailKit)
- **ManagementPort** – MIT (https://github.com/bernd-herzog/ManagementPort)
- **Microsoft.EntityFrameworkCore** – MIT (https://github.com/dotnet/efcore)
- **Npgsql.EntityFrameworkCore.PostgreSQL** – PostgreSQL License (https://github.com/npgsql/npgsql)
- **OpenTelemetry.Exporter.OpenTelemetryProtocol** – Apache-2.0 (https://github.com/open-telemetry/opentelemetry-dotnet)
- **OpenTelemetry.Extensions.Hosting** – Apache-2.0 (https://github.com/open-telemetry/opentelemetry-dotnet)
- **OpenTelemetry.Instrumentation.AspNetCore** – Apache-2.0 (https://github.com/open-telemetry/opentelem dotnet)
- **OpenTelemetry.Instrumentation.Http** – Apache-2.0 (https://github.com/open-telemetry/opentelemetry-dotnet)
- **Polly** – BSD-3-Clause (https://github.com/App-vNext/Polly)
- **Swashbuckle.AspNetCore** – MIT (https://github.com/domaindrivendev/Swashbuckle.AspNetCore)
- **System.IdentityModel.Tokens.Jwt** – MIT (https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet)
- **TimeZoneConverter** – MIT (https://www.nuget.org/packages/TimeZoneConverter)

Firmware (C/C++)

- **bsdiff** – BSD (https://github.com/mendsley/bsdiff)
- **bzip2** – bzip2 License (https://gitlab.com/federicomenaquintero/bzip2)
- **eLua** – MIT (https://github.com/elua/elua)
- **FatFs** – FatFs License (https://elm-chan.org/fsw/ff/)
- **libjpeg-turbo** – IJG/BSD (https://github.com/libjpeg-turbo/libjpeg-turbo)
- **LittleFS** – BSD-3-Clause (https://github.com/littlefs-project/littlefs)
- **LVGL** – MIT (https://github.com/lvgl/lvgl)
- **luavgl** – MIT (https://github.com/XuNeo/luavgl)
- **mbedTLS** – Apache-2.0 (https://github.com/Mbed-TLS/mbedtls)
- **NanoPB** – BSD-3-Clause (https://github.com/nanopb/nanopb)
- **nlohmann/json** – MIT (https://github.com/nlohmann/json)
- **UMM_malloc** – MIT (https://github.com/rhempel/umm_malloc)

Gateway (ESP-IDF Managed Components)

- **espressif__esp_websocket_client** – Apache-2.0 (https://components.espressif.com/components/espressif/e
- **espressif__nghttp (nghttp2)** – MIT (https://github.com/nghttp2/nghttp2)
- **espressif__sh2lib** – MIT (https://components.espressif.com/components/espressif/sh2lib)

Frontend (JavaScript/TypeScript)

- **@dnd-kit/core** – MIT (https://www.npmjs.com/package/@dnd-kit/core)
- **@fluentui/react-components** – MIT (https://www.npmjs.com/package/@fluentui/react-components)
- **@monaco-editor/react** – MIT (https://www.npmjs.com/package/@monaco-editor/react)
- **crypto-js** – MIT (https://www.npmjs.com/package/crypto-js)
- **i18next** – MIT (https://www.npmjs.com/package/i18next)
- **immer** – MIT (https://www.npmjs.com/package/immer)
- **js-sha512** – MIT (https://www.npmjs.com/package/js-sha512)
- **jwt-decode** – MIT (https://www.npmjs.com/package/jwt-decode)
- **pngjs** – MIT (https://www.npmjs.com/package/pngjs)
- **React** – MIT (https://www.npmjs.com/package/react)
- **react-i18next** – MIT (https://www.npmjs.com/package/react-i18next)
- **react-router-dom** – MIT (https://www.npmjs.com/package/react-router-dom)
- **react-select** – MIT (https://www.npmjs.com/package/react-select)
- **recharts** – MIT (https://www.npmjs.com/package/recharts)
- **swr** – MIT (https://www.npmjs.com/package/swr)
- **underscore** – MIT (https://www.npmjs.com/package/underscore)
- **upng-js** – MIT (https://www.npmjs.com/package/upng-js)