

AUTOMATIC MONITORING MODULE

JEEDOM WATCH SYSTEM (JWS)



Version v1.3

Updated on January 20, 2025

Table of Contents

Versions history.....	3
V0.1 – September 06, 2024.....	3
V1.0 – October 04, 2024	3
v1.1 – October 11, 2024.....	3
v1.11 - November 2nd, 2024.....	3
V1.2 – November 9, 2024.....	3
V1.3 – January 20, 2025	3
Overview	4
Hardware.....	4
Working principle	5
First launch	6
Error Detection.....	7
OLED display.....	7
LED Display.....	9
Error coding.....	10
Self-monitoring of the WiFi network connection.....	10
Cessation of monitoring	10
Installation and updates.....	11
Initial installation.....	11
Flashing an ESP32 with web.esphome.io.....	12
OTA Updates	13
Procedure with the web server	13
Procedure with the Arduino IDE	14
Display during updates	14
Resetting the module.....	15
Jeedom configuration.....	16
Probable Causes of Resulting Errors in Testing.....	18
Links for the purchase of the necessary equipment	19
Practical implementation	20
USB sockets.....	20
RGB LED mounting	22
Mounting the ESP with its bracket and the relay board in the box	23
Finishes.....	25
Electronic diagram.....	27
Table des figures	28

Versions history

V0.1 – September 06, 2024

Initial Release

This version is minimalist and does not manage any status feedback device (OLED or LED screen). All Wifi and MQTT settings (server, passwords, @IP, ports) are directly encoded in the source file.

V1.0 – October 04, 2024

Version 1.0 is the first release and release, and will support:

- An OLED screen of type SH1106 or SSD1306 (optional),
- A status indicator per RGB LED diode (or 3 separate red, green and blue LEDs),
- Wifi and MQTT parameters recorded in EEPROM memory,
- Connection management to the Wi-Fi network with the defined parameters,
- Updates with OTA procedure, either by dedicated web server or the Arduino IDE.

As well as the correction of some minor residual bugs.

v1.1 – October 11, 2024

Changes and corrections brought by version 1.1 are as follows:

- Modification of the LED management to switch to PWM mode, allowing a gradual on/off, and adjustment of the brightness (via the source code).
- Multi-tasking management for initialization step with the dual cores of the ESP32,
- Adding a progress bar and labels of initialization step performed,
- Fixed the following bugs:
 - Behavior when the Wifi credentials entered in AP mode are wrong,
 - Removing the 30-second timeout after initialization.

v1.11 - November 2nd, 2024

Time cycle for tests is longer from 30 seconds to 1 minute, in order to allow updating of system if takes more time than expected in some cases, especially for MQTT. A stop/restart will occur now 5 minutes after an error rises up instead of 2 minutes and half.

V1.2 – November 9, 2024

This version brings the following changes:

- After initializing the module, a permanent WiFi connectivity check (as defined in the settings) is performed. If any network disconnection is detected, JWS will attempt to reconnect and if is unsuccessful, perform a JWS resetting.
- Setting the hostname 'JWS_ESP32' in order to facilitate module identification on WiFi network.

V1.3 – January 20, 2025

Added a maintenance/operational mode function to suspend the monitoring of the Jeedom box.

Improved web interfaces.

Overview

The JWS monitoring module is based on an ESP32 microcontroller and allows, in complete autonomy, to detect potential failures of a home automation system (Jeedom, Home Assistant, or other) hosted on a stand-alone box (such as Atlas, Luna, RPi, mini-PC, etc.).

Since resetting is performed by a power restart (off/on), **this module is not suitable** for virtual home automation systems hosted on NAS or other servers that perform other functions.

Faults are detected at the application level (Jeedom/HA via MQTT protocol) and network (IP), and the module will electrically restart the home automation box up to twice in an attempt to recover normal operations.

It connects on the one hand to a 5v power supply by a male USB-C cable (15W max), and on the other hand to the home automation box also with a male USB-C cable. It is self-powered directly via the USB-C socket it controls, so it does not require any external power supply.

Hardware



Figure 1 - Materials used

All the equipment is based on:

- A standard 30-pin ESP32 4MB/240Mhz/Wifi, with its support,
- A relay module with NO (Normally Open) and NC (Normally Closed) contacts,
- An OLED screen type SH1106 or SSD1306 offering a resolution of 128 (W) x 64 (H) pixels with an I2C interface (4 pins: GND, VCC, SCL, SDA),
- An RGB LED (or 3 red, green, and blue LEDs) with 3 limiting resistors from 180 to 270 ohms,
- Two USB-C chassis-type sockets with a minimum of 6 contacts (VBUS, GND, D+, D-, CC1, CC2),
- A USB-C to USB-C male cable (USB-C OUT socket to the box),
- A case (minimum internal dimensions: L120 x W75 x H27 mm),
- A 75x35x1mm smoked translucent acrylic sheet (cover for the OLED screen),

- Standard hardware (1.6, 2.5 and 3mm screws, washers, and nuts), wiring (24 AWG gauge), heat shrink tubing.

Working principle

At startup, the ESP32 will check for the presence of an OLED screen and display a picture if equipped, then start the initialization procedure:

- Self-test of the LEDs (gradual and successive switching on and off for one second of the red, green and blue LEDs),
- Initialization of the EEPROM,
- Resetting the EEPROM if required,
- Reading EEPROM parameters,
- Connection to the Wi-Fi network (except for the first launch, see next chapter '[First launch](#)').
- Initialization of update routines with OTA,
- Initialization of MQTT exchanges,

These phases are materialized on the OLED screen by a progress bar and displaying the labels of the completed stages.

During the monitoring phase (also called 'Operational' mode), it will perform the following tests every 60 seconds:

- Network ping to the home automation box,

This ping ensures that the home automation box is still connected to the network, and that it is not frozen or crashed.

- Maintaining the MQTT connection,

An MQTT connection must be activated and managed by an MQTT server (Mosquitto broker,...), which reports itself via the associated messaging system. Failure to report this may be the result of a software shutdown of the server. Generally, home automation boxes rely on this protocol to manage the modules present on the Zigbee network or the modules connected via Wifi. It is therefore not necessary in most cases to plan installation and specific activation of a plugin for managing this protocol.

- Sending and waiting an answer to a MQTT message.

In addition to checking for the presence of a broker, the JWS module will emit a specific MQTT message (ping) and wait in return for the response of the home automation box (pong). If there is no response, it is possible that the plugin daemon has stopped.

In any case, any error(s) detections will be indicated by a display of a summary screen on the OLED screen and the lighting of the corresponding LEDs.

In 'maintenance' mode, monitoring of the home automation box is suspended for an unlimited period of time. This mode allows, for example, to proceed for any updates required to stop MQTT and/or IP services, and or which can exceed the limits beyond which the JWS module would proceed to a shutdown/correct operation of the box.

See chapter '[Stopping monitoring](#)' for more details.

First launch

At the first start (after uploading or resetting the EEPROM), ESP32 must memorize the Wifi network connection parameters used by the home automation box (SSID and password), IP address of the MQTT server and the home automation box to be monitored (it should be the same IP address if the MQTT server has been installed in local mode), the username, password, and MQTT port used (by default, the port is 1883).

The ESP will therefore initialize a web server and a Wifi network in point-to-point (AP) mode, showing the SSID `JWS_ESP32-AP` at the address 192.168.4.1, on which you will have to connect with any Wifi terminal and Internet browser, i.e. such as a smartphone.

In this mode, OLED display leaves the startup logo displayed, and the red LED lights up. All you have to do is to fill the various fields displayed, before validating the configuration parameters with the 'Connect' button.

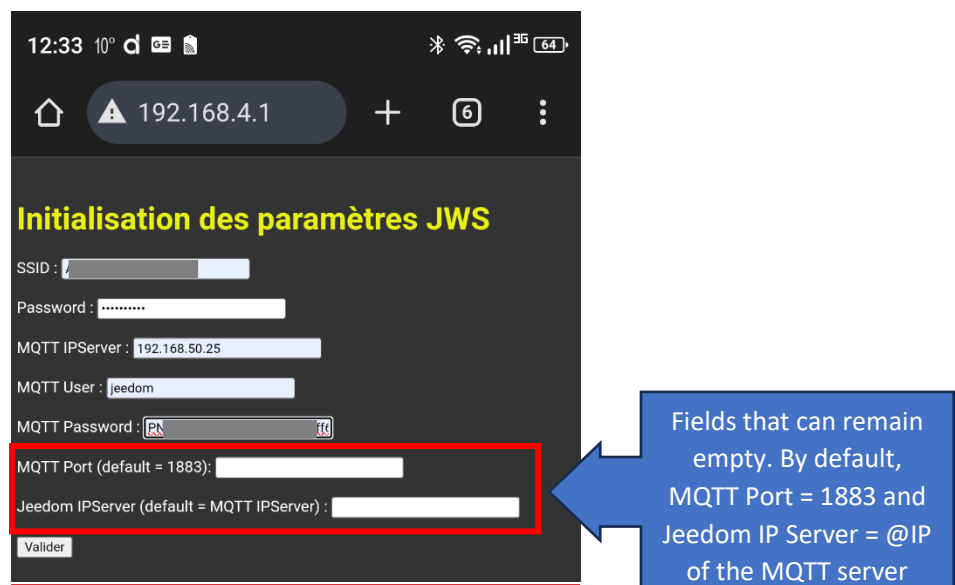


Figure 2 - Screenshot on a smartphone of the web server in AP mode

Once these parameters have been validated, ESP will exit of the AP mode and will reset itself to try to connect to the Wifi network thus defined and initialize the links on the MQTT network.

The OLED screen will display the boot logo and the initialization progress bar,



Figure 3 - Startup logo

Then after a few seconds, the main page.

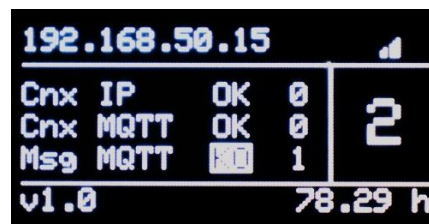


Figure 4 - Main page

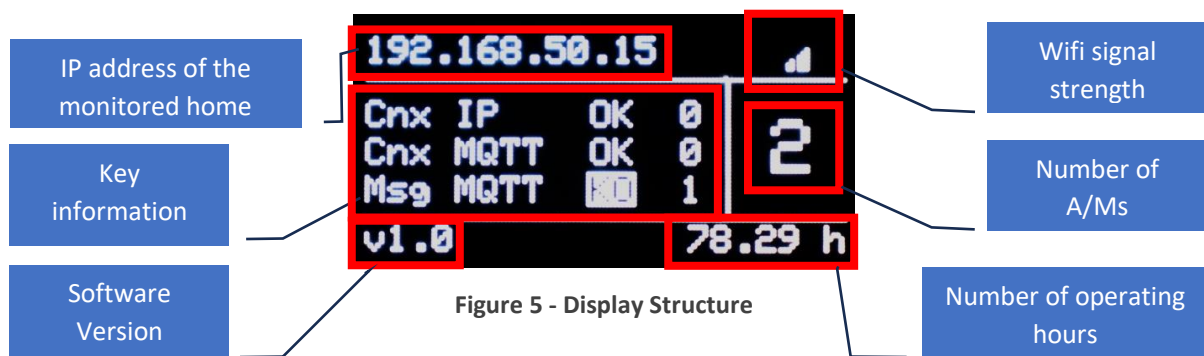
At this step, the module will start to monitor MQTT and Wifi connections with the home automation box in loop, and the OLED screen will turn off.

Error Detection

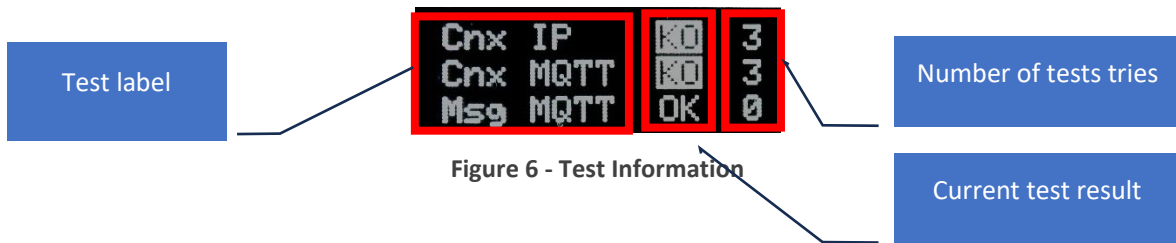
If an error is detected on one of the tests, the corresponding LED(s) will light up and the OLED screen will be activated, as long as the error is not resolved.

OLED display








The display is structured in different distinct areas:



- **Module IP Address:** Indicates the IP address assigned to the JWS module by the Wifi network router.
- **Wifi signal strength:** indicates the received strength of the Wifi signal,
- **Software Version:** Indicates the version of the software,
- **Number of hours of operation:** indicates the number of hours of operation since the start or the last RESET performed, in hours + 1/10 of an hour,
- **Main information:** indicates the result of the tests performed (test, result, number of occurrences in error),



- **Number of Stop/On Remaining:** Indicates the total number of stop/on that can be performed, decreasing from 2 to 0. At 0, there will be no more stops caused afterwards.

RGB LEDs	Fixed	Flashing	Pulsed
Red 	IP Network Ping Error At startup, the ESP is in AP mode (waiting for parameters)	OTA update in progress	3x = End of OTA update with error
Green 	MQTT connection error	-	3x = error-free OTA update end
Blue 	MQTT Message Exchange Error	-	1x every 60 seconds = JWS in service Permanently = maintenance mode
Cyan 	MQTT Message Exchange and MQTT Connection Errors	-	-
Magenta 	IP network ping errors and MQTT message exchange	-	-
Yellow 	IP network ping errors and MQTT connection	-	-
White 	IP network ping, MQTT connection, and MQTT message exchange errors	-	-

After 5 unsuccessful tests, the module will cause the home automation box to stop/start by cutting off the power to the USB socket for 0.5 seconds to restart it.



Figure 7 – Resetting of box

After two successive unsuccessful restart attempts, the JWS module no longer causes a shutdown and is limited to reporting the occurrences of the tests in error, as long as the detected errors are not resolved.



Figure 8 - No more restarts are possible (counter at 0)

Error coding

The internal error coding is as follows:

MQTT Messages Error	MQTT Connection Error	IP Connection Error	#
-	-	-	0
-	-	X	1
-	X	-	2
-	X	X	3
X	-	-	4
X	-	X	5
X	X	-	6
X	X	X	7

Self-monitoring of the WiFi network connection

After initialization, the JWS module will perform a permanent monitoring of the WiFi network connection.

In the event of a loss of this connection, it is first necessary to make sure that the JWS module is not involved in the loss of this connection, before launching the planned reactions for an out of order or unreachable home automation box.

In this case, JWS module will make a reconnection attempt with a 10-second time-out, and if unsuccessful, the JWS module will be resetting.

Cessation of monitoring

The JWS module can be switch in two different modes:

- Operational mode (Normal Mode)

In this mode, the JWS module will ensure permanent monitoring of the Jeedom box to which it is associated.

- Maintenance mode

When this mode is selected, the JWS module will stop all monitoring of the Jeedom box for an unlimited period of time, and will no longer take into account the possible absence of responses to MQTT messages or IP pings, or the loss of the MQTT link.

This mode makes it possible to isolate and secure the Jeedom box to allow it to carry out any long maintenance operations (heavy updates, etc.), and whose foreseeable duration would be longer than the maximum period beyond which the JWS module would react by stopping/starting (in principle after 5 minutes).

Activating this mode results in a continuous flashing of the blue LED in pulsed mode, and the permanent display on the OLED screen of the message "JEEDOM MTN".



Figure 9 - Maintenance mode

The mode change is done from the 'JWS Settings' web page available on the @IP assigned to the JWS module:



Figure 10 - Mode selection from the Web Server

The `Maintenance` and `Operational` buttons allows to switch between these two modes, with the activated mode being recalled by this line : `Mode: Maintenance` or `Mode: Operational`.

Installation and updates

Initial installation

The initial firmware installation (.bin binary) can be done with ESP32 tools website here: <https://web.esphome.io>. But it can only be accessed with Google Chrome or Microsoft Edge browsers at this time (WebSerial component is required).

Failing that, it is possible to install and use a specific .bin file upload tool such as ESPHome Flasher (available for download on Github here: <https://github.com/esphome/esphome-flasher>).

Flashing an ESP32 with web.esphome.io

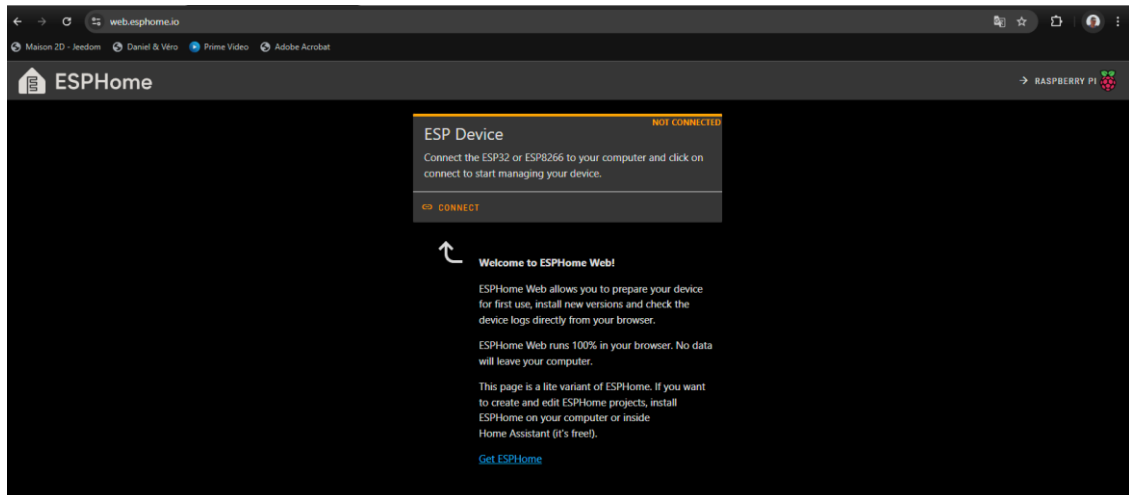


Figure 11 - Home page of the web.esphome.io site

Connect the ESP32 module via USB to the computer, then go to the <https://web.esphome.io> site (with Chrome or Edge). Click on **CONNECT**. Select the corresponding USB port, and then click **Connect**.

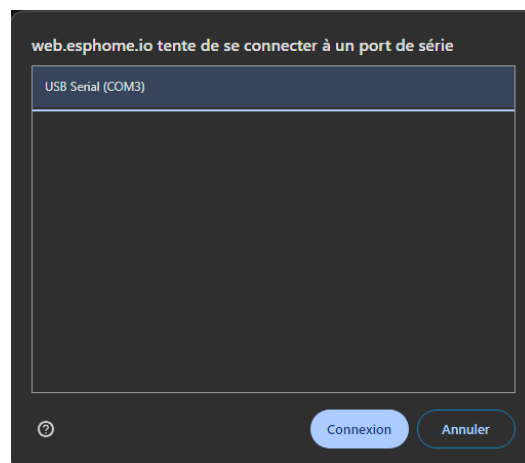


Figure 12 - Connecting the ESP32

Click on **INSTALL**.

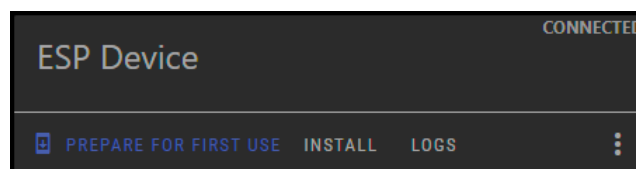


Figure 13 – ESP Connected and Ready

Then select the file .bin on the computer (**Choose File**) and click **INSTALL**.

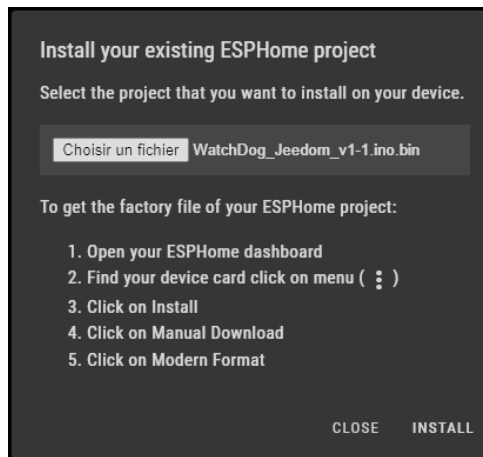


Figure 14 - File .bin selected and ready for upload

The upload to the ESP32 begins.

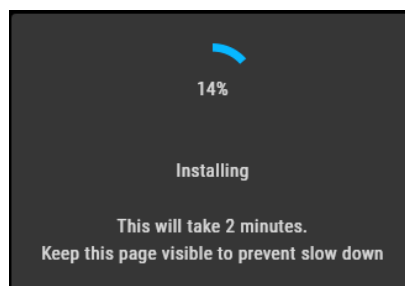


Figure 15 - Upload in Progress

And if there were no errors, it ends with this message:

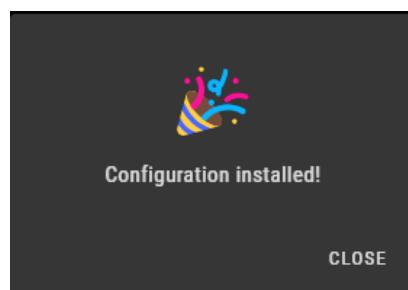


Figure 16 - Successful completion

OTA Updates

After a first initial upload via the serial socket (USB), the module can be updated with the OTA method ('Other The Air', via the Wifi network) either with a dedicated web server, accessible in current operation at all times on the IP address dedicated to the JWS module, or with the Arduino IDE by recompiling the source code and sending it via the Wifi network.

Procedure with the web server

Log in to the web page @IP the JWS module:

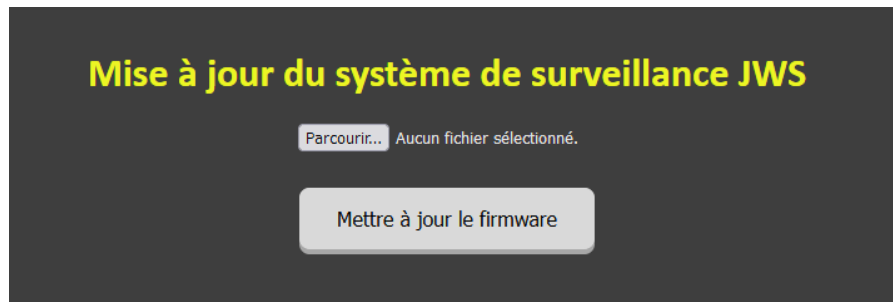


Figure 17 - Web server for firmware upload

Click `Browse...` and select the firmware binary.bin

Then click Update `Firmware` to begin the update procedure.

After a successful upload, this message should appear in the browser.

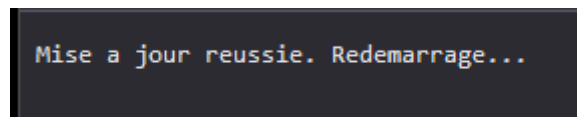


Figure 18 - Update Successfully Completed

Procedure with the Arduino IDE

Launch the Arduino IDE, and select the ESP32 dev module and the Wifi port designated by the name `JWS_ESP32`.

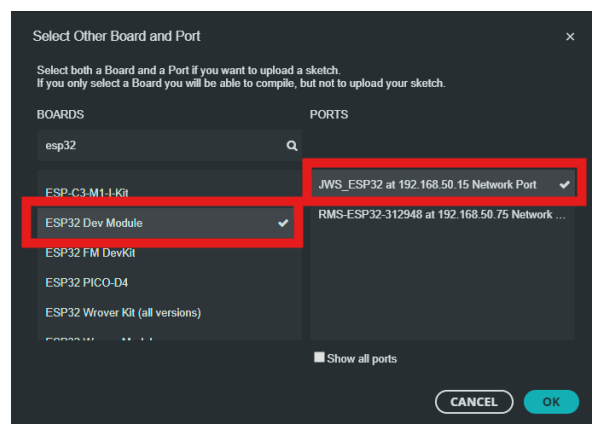


Figure 19 – OTA Update with the Arduino IDE

To start a compilation, the upload will be done automatically to the ESP32 at the end provided that there have been no errors during the process.

On the first upload, the Arduino IDE will ask for a password. There is no password set, just enter any string of any length.

Display during updates

Regardless of the method used, during the upload the OLED screen will display a specific message and the red LED will flash quickly according to the data received, indicating that is in progress.



Figure 20 - Updating

At the end of the update, the green LED will light up and gradually turn off three times in a row to indicate that there was no error during the upload.

Otherwise, the red LED will light up and gradually turn off three times to signal an error that occurred during the upload, with a specific error message displayed on the OLED screen.

The module will restart at the end.

Parameters stored in EEPROM are not affected by updates.

Resetting the module

The module can be returned to its initial configuration by performing a resetting of all the parameters stored in the internal EEPROM memory.

To do this, you need to connect pin 13 to ground (they are located side by side) when starting the module.

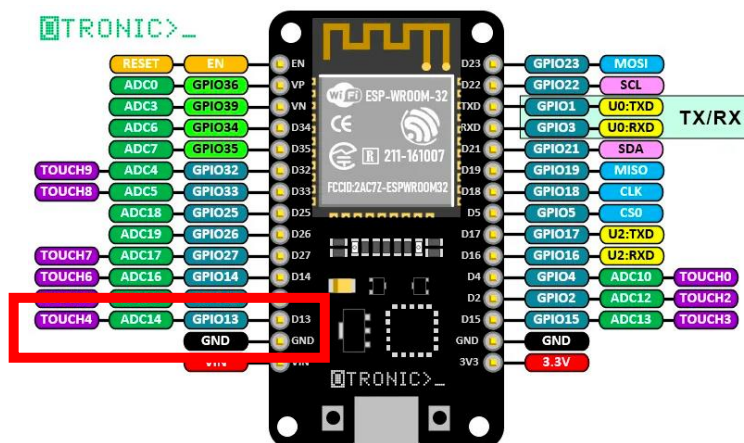


Figure 21 - The GPIO13 and GND pins on the ESP32

After the erasure, the ESP will automatically restart the establishment of a network in AP mode for the definition of the new connection parameters.

Note

Since this function is not normally frequently used, there is no provision for a specific push button to perform this function.

Jeedom configuration

To ensure dialogue on the MQTT network with the JWS module, an MQTT broker (i.e. Mosquito) must be installed, configured and started to activate the network and send a response to the ping messages received periodically.

You have to create a `Self-Monitoring device`, with the plugin that manages the MQTT protocol (i.e. MQTT Manager), by setting the root topic to the ping value.

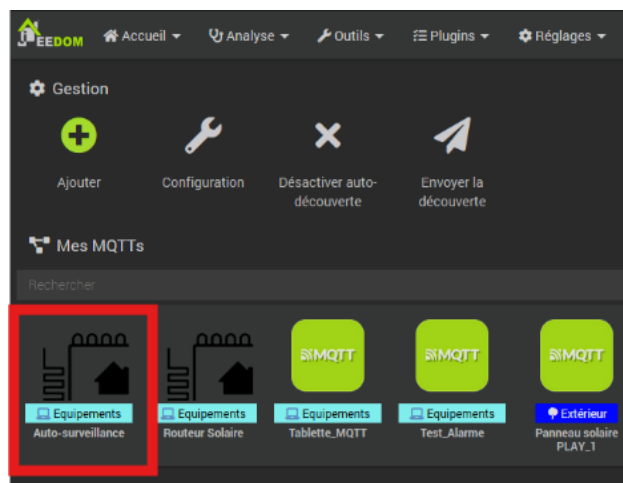


Figure 22 - Creating Self-Monitoring Equipment

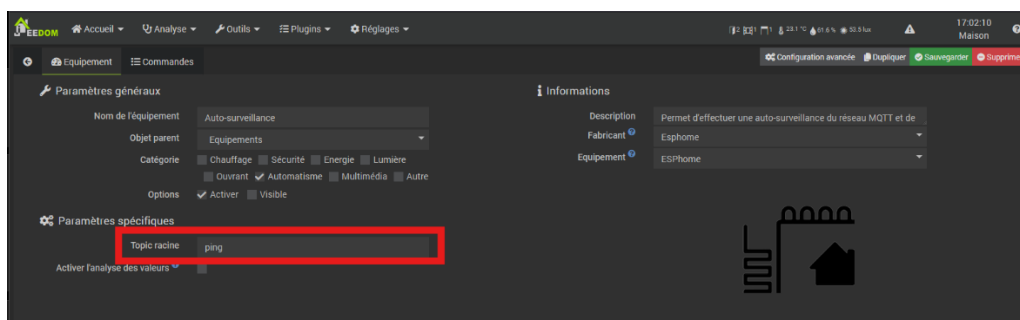


Figure 23 - Definition of the root topic

In the Commands tab, create two commands: `Ex_Pong` (action type, Rx topic, value = pong) and `Rx_Ping` (info type, Ex topic).

These commands will be configured as follows:

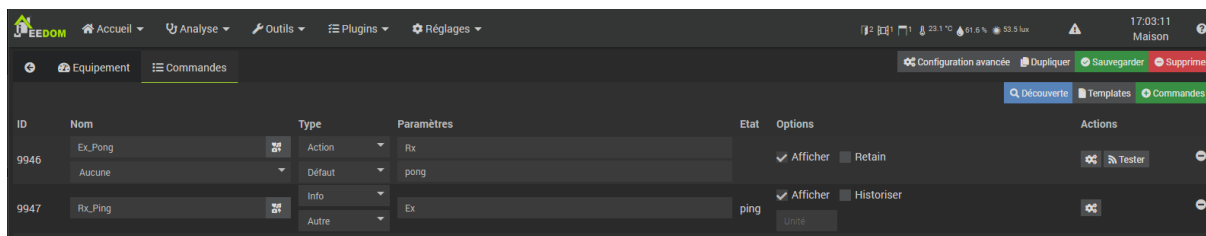


Figure 24 - Defining Commands

In the advanced configuration for `Rx_Ping` information, configure an action on the received ping value to emit the `pong` message.

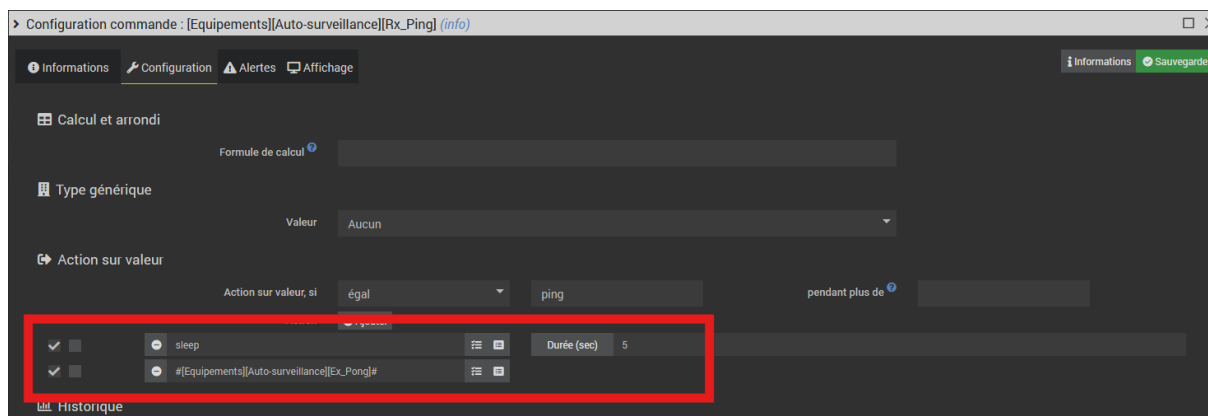


Figure 25 - Action to Perform on Receiving a Ping

Astuteness

The sleep timeout allows the `pong` response to be returned to the JWS module only after the timeout has expired. The latter will therefore detect and report the faulty test by the blue LED diode and the OLED screen display during this interval, to turn them off as soon as the response is received, causing a more or less stealthy flash.

This flash makes it possible to confirm the activity of the JWS module every 60 seconds. A delay of 5 seconds causes the blue LED to flash for about 2 seconds.

Probable Causes of Resulting Errors in Testing

Errors identified during the test cycle	Probably because...
No ping	The home automation box is disconnected from the home network. The home automation box is not started.
No MQTT connection	The MQTT server (Mosquito or other) is stopped. The home automation box is not started.
No response to MQTT messages	The daemon of the plugin handling MQTT connections (MQTT Manager, ZigbeeLinker,...) is stopped, The 'Self-Monitoring' equipment is not activated, The 'pong' message is not activated. The home automation box is not started.

Links for the purchase of the necessary equipment

Hardware	Nb	Link	Price
ESP32 standard CH340C USB-C with its 30-pin support	1	AliExpress	7,49€
1 channel 5v relay module	1	AliExpress	1,39€
OLED display 128x64 pixels, I2c bus	1	AliExpress	2,54€
RGB LED 5mm common anode	1	AliExpress	1,99€ (x50)
Resistor 220 ohms 1/4W	3	AliExpress	0,94€ (x220)
Chassis USB-C socket 6 contacts CR-19	2	AliExpress	2.11€ (x5)
Angled USB-C cable type C2C 0.25m	1	AliExpress	2.05€
Box (ext/int dimensions: 125(120) x80(75) x32(27) mm)	1	AliExpress	2,55€
Hardware (Phillips screws + nuts black steel M1.6 / 2.5 / 3)	14	AliExpress	15,23€ (x1500)
Heat shrink tubing 1.5mm	-	AliExpress	1,89€
Wires 22-24 AWG	-	AliExpress	€1.27 (x5m)
Translucent acrylic sheet 10x10x1mm 'Light black' color	1	Ali Express	3,91€ (x2)
Total			17.58€ to 43,36€

(Note: the equipment presented on the lines with is either optional or can be picked up from stock)



Figure 26 - JWS Module

Practical implementation

USB sockets

USB-C chassis sockets used must have a minimum of 6 pins:

- VBUS (+5v),
- GND (mass),
- CC1, CC2 (configuration channel),
- D+, D- (data).

The CC1 and CC2 signals (Channel 1&2 Configuration) are used to indicate to the power supply the presence of a charge on the USB-C socket. These two pins are in principle connected to ground by a resistance of 5.1Kohms, in accordance with the specifications of the USB standard. Without these signals, the outlet will not deliver any current.

The D+ and D- signals, normally used in data transfer, will be used here by the home automation box for the negotiation with the power supply of the delivery of a current up to 3A max. Without these pins, the power supply will only provide a voltage of 5v under 1A maximum by default, which is insufficient to allow the power supply of a home automation box.



Figure 27 - Chassis USB-C socket pinout

The CC1, CC2, D+, D- signals are not used by the JWS module, and must be connected face-to-face. The JWS module will therefore not be powered if there is no home automation box plugged into the USB-C OUT socket.

As a result, USB sockets require the soldering of 6 wires, the two power wires (GND and VBUS) of which must comply with at least the 24 AWG standard or more (i.e. a minimum diameter of 0.5 mm for a section of 0.205 mm²), to be able to carry the maximum current of 3A under 5V necessary for the home automation box.

The JWS module's own power supply is provided by the same power supply as the home automation box, via the VBUS and GND pins. Only the VBUS voltage (+5v) will be cut off by the relay when the home automation box is restarted.

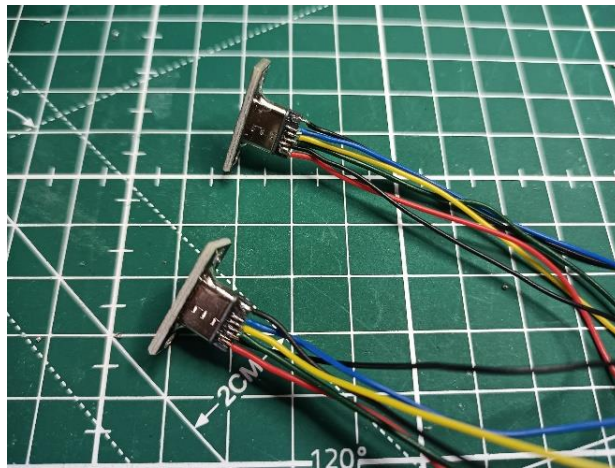
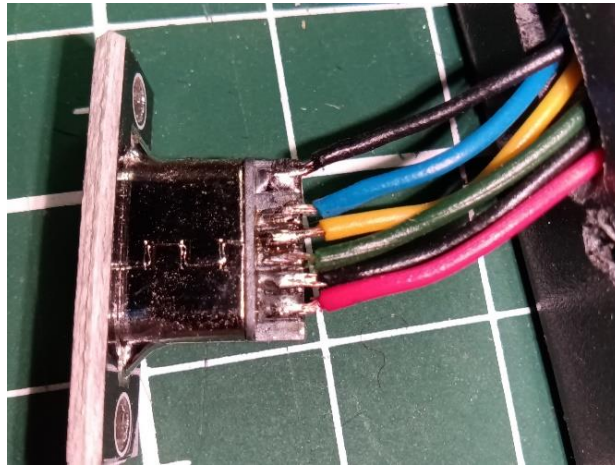


Figure 28 - USB-C Socket wiring details

Final mounting on the case, with a fastening on the bottom half shell of the case to facilitate opening operations.



Figure 29 - Installing USB-C Outlets in their final locations

RGB LED mounting

The RGB LED will be soldered with its three limiting resistors in series on the R, G and B pins. It is of the Common Anode (AC) type, and therefore driven by a low signal on its RGB pins. It can be either clear (for a high directional luminosity) or opaque (with a more diffused light).

Nevertheless, type of LED used (AC or DC) can still be set in the source code if necessary by commenting on the ad-hoc line:

```
#define Type_LED_AC 1 // Définition du type de LED Anode Commune (AC)
// #define Type_LED_CC 1 // Définition du type de LED Cathode Commune (CC)
```

Note

The binaries files available on Github are compiled by default for a Common Anode LED type as shown here.

The RGB LED will be soldered with its three limiting resistors on the R, G and B pins. It is of the Common Anode (AC) type, and therefore driven by a low signal on its RGB pins. It can be either clear (for a high directional luminosity) or opaque (with a more diffused light).

Limiting resistors can have a value between 180 and 270 ohms (the LED being powered by 3.3v), with 220 ohms being the recommended value. The use of heat-shrink tubing on each pin will allow mounting without the risk of untimely short circuits.

Just like the USB sockets, to facilitate opening, the LED will be mounted on the lower part of the case.

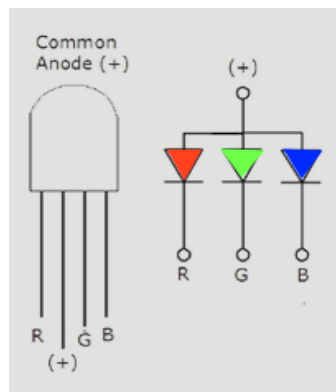


Figure 30 - Pinout of a Common Anode RGB LED

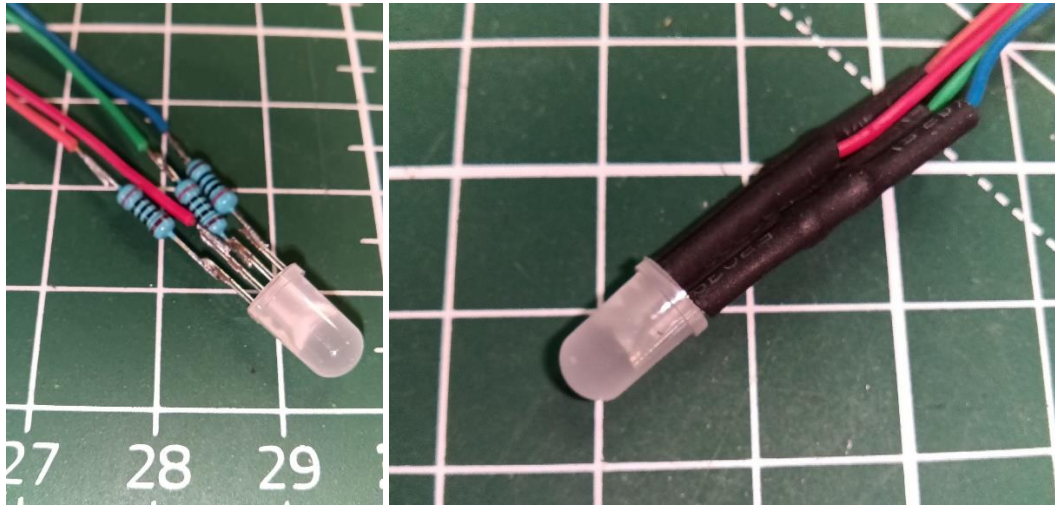


Figure 31 - Details of the installation of resistors and heat shrink tubing

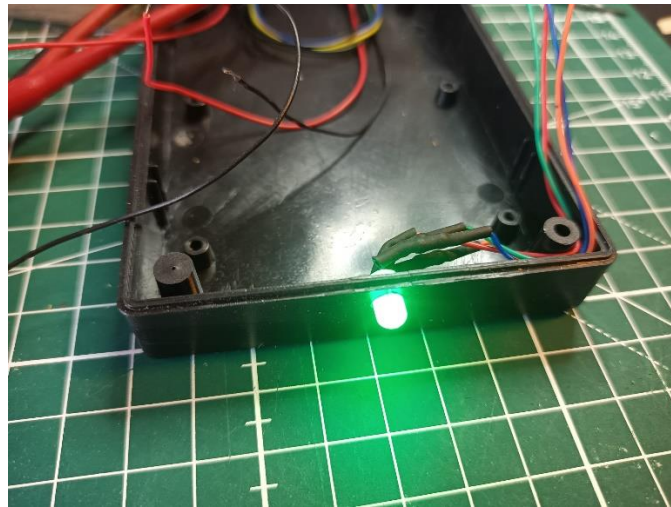


Figure 32 - Mounting the RGB LED in the case

Mounting the ESP with its bracket and the relay board in the box

For mounting, the ESP with its support and the relay board will be fixed using the lugs provided for the fixings at the bottom of this housing model. At this point, the wiring between the elements can be carried out.

The OLED screen will be fixed with 4 bolts (2.5 mm screws) on the upper half shell of the case after opening a window of sufficient size.

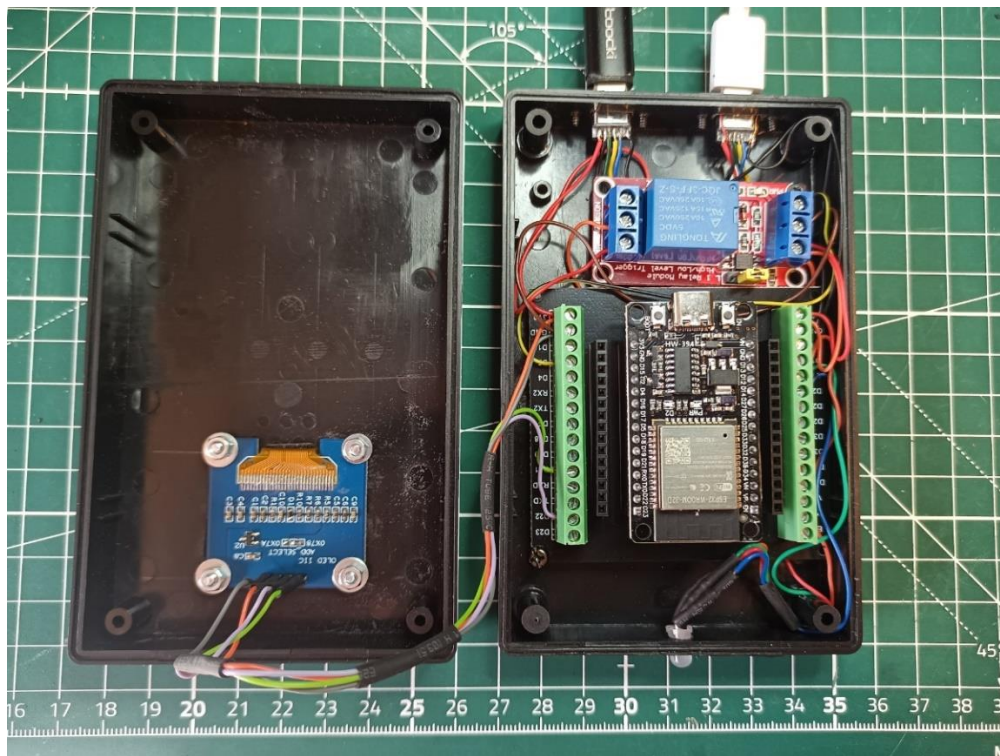


Figure 33 - Mounting the ESP, Relay Board, and OLED Display

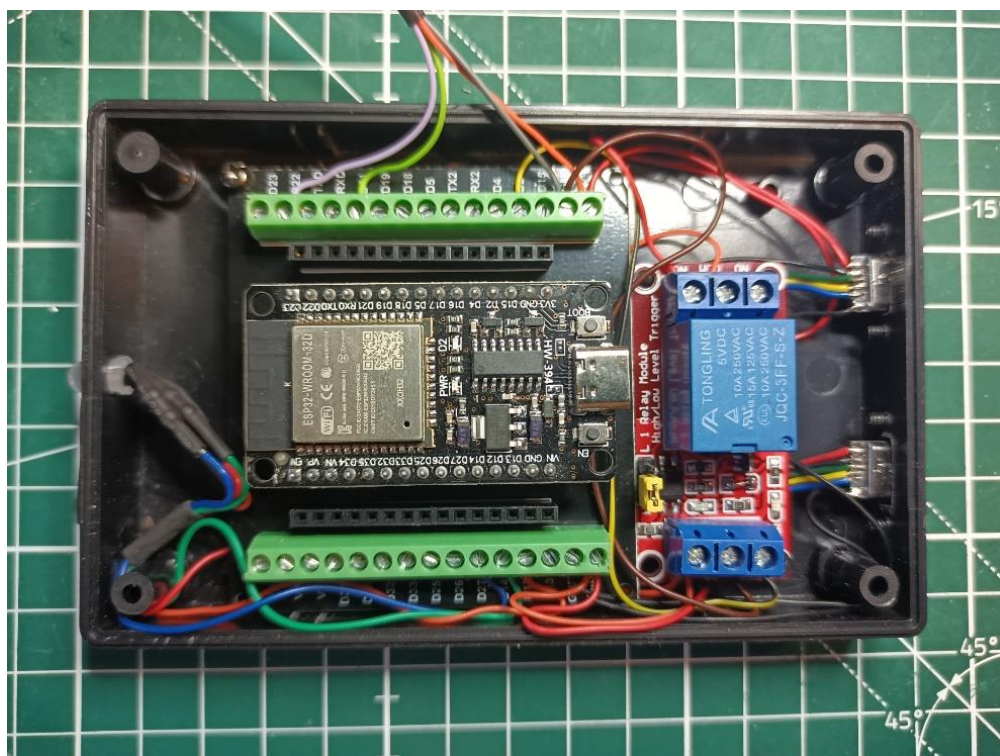


Figure 34 - Mounting details of the ESP, relay board, RGB LED, USB-C sockets, and wiring



Figure 35 - Cutting out a window for the OLED display on the top of the case

Finishes

The case can be finished by attaching a 1mm thick plate in translucent acrylic in smoke black, fixed by black screws and 1.6mm nuts. This plate will effectively hide the OLED screen with its fixings (2.5 mm black flat screws) as well as the opening made on the case, while maintaining optimal readability.

This will also be an opportunity to place an identification label, and to locate the USB IN and OUT sockets (decals). A light varnish will be applied to fix and protect the indications, and obtain a glossy finish.



Figure 36 - The finalized case

Version 1.3



Table des figures

Figure 1 - Materials used	4
Figure 2 - Screenshot on a smartphone of the web server in AP mode	6
Figure 3 - Startup logo	6
Figure 4 - Main page	7
Figure 5 - Display Structure	7
Figure 6 - Test Information	8
Figure 7 – Resetting of box	9
Figure 8 - No more restarts are possible (counter at 0)	10
Figure 9 - Maintenance mode	11
Figure 10 - Mode selection from the Web Server	11
Figure 11 - Home page of the web.esphome.io site	12
Figure 12 - Connecting the ESP32	12
Figure 13 – ESP Connected and Ready	12
Figure 14 - File .bin selected and ready for upload	13
Figure 15 - Upload in Progress	13
Figure 16 - Successful completion	13
Figure 17 - Web server for firmware upload	14
Figure 18 - Update Successfully Completed	14
Figure 19 – OTA Update with the Arduino IDE	14
Figure 20 - Updating	15
Figure 21 - The GPIO13 and GND pins on the ESP32	15
Figure 22 - Creating Self-Monitoring Equipment	16
Figure 23 - Definition of the root topic	16
Figure 24 - Defining Commands	17
Figure 25 - Action to Perform on Receiving a Ping	17
Figure 26 - JWS Module	19
Figure 27 - Chassis USB-C socket pinout	20
Figure 28 - USB-C Socket wiring details	21
Figure 29 - Installing USB-C Outlets in their final locations	21
Figure 30 - Pinout of a Common Anode RGB LED	22
Figure 31 - Details of the installation of resistors and heat shrink tubing	23
Figure 32 - Mounting the RGB LED in the case	23
Figure 33 - Mounting the ESP, Relay Board, and OLED Display	24
Figure 34 - Mounting details of the ESP, relay board, RGB LED, USB-C sockets, and wiring	24
Figure 35 - Cutting out a window for the OLED display on the top of the case	25
Figure 36 - The finalized case	26

Project designed and produced by J. Daniel ©

October 11, 2024

The sources, binaries, and documentation are available for free download or viewing on GitHub: <https://github.com/jngdan/JWS>