

# User Manual

## Application Shield ESP32



version 1.0

date: 02/07/22

1 Overview.....	3
2 Installation.....	5
3 Connect the Application Shield .....	7
5 Upload the Application Shield firmware .....	8
4 A short description of the source code .....	10
5 LED status .....	11
6 Possible Errors .....	12
7 PCB design .....	13
8 Schematic .....	14

# 1 Overview

This Application Shield can be used with an ESP32 development board. It is designed to facilitate the access to our thermopile arrays for the fast way to get the thermal image from the sensor. The full code is provided and completely open source. It includes all required steps from reading the EEPROM to the calculation of the thermal image. The C++ code can be viewed and modified via the Arduino IDE. The PCB is designed as an ESP32-DevkitC-32D extension.

Supported Sensor types:

	TO46	TO39	TO8
I2C	8x8d	16x16d	/
		32x32d	
SPI	/	60x40d	80x64d
			84x60d*
			120x84d
* software will be available soon			

The source code includes two ways to interact with the sensor:

- via WIFI you can stream thermal images in our GUI
- via the serial monitor you can observe the sensor data as text output

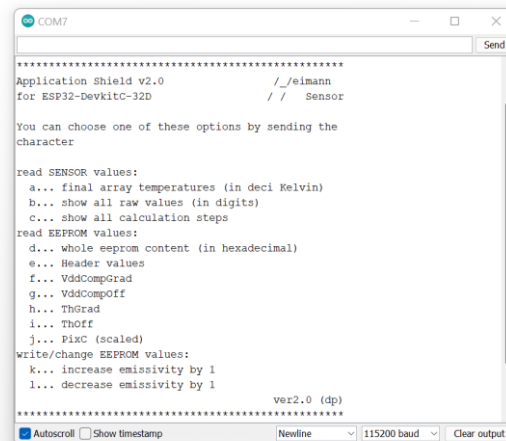
Both modes are containing in the same code and you can activate one or both by activate the matching define.

## Serial

This mode prints all results in the serial monitor of the Arduino IDE. Here the EEPROM/Flash content and sensor voltages can be visualized. It's easy to interact with the sensor by sending the characters depending on the menu function you want.

Benefits:

- shows EEPROM/Flash content in hexadecimal or associated data type (float, short, long, ...)
- prints result after each calculation steps
  - understand the way from raw pixel voltages to the final compensated thermal image

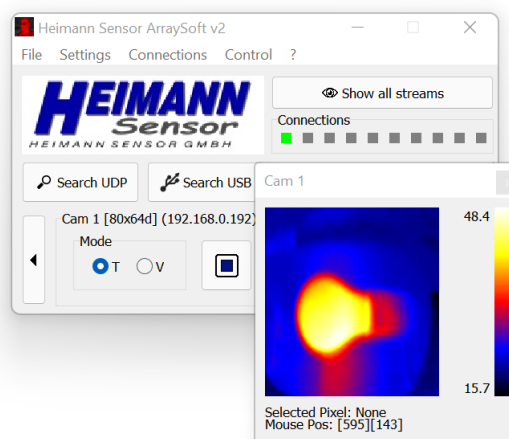


## WIFI

Via WIFI you can connect your thermopile with the Heimann Sensor GUI to stream continuously. With the GUI streaming the sensor images in temperature or voltage mode is possible. Also, you can change user settings, like clock, ADC resolution and emissivity factor.

Benefits:

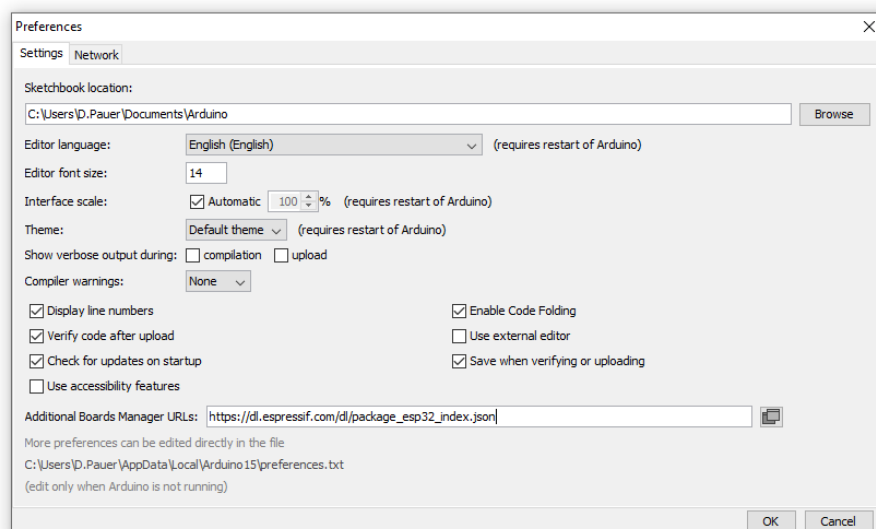
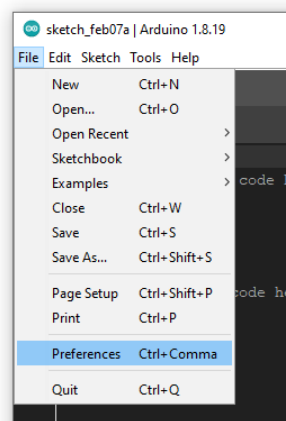
- false color visualization of images
- stream continuously
- switch between temperature and voltage mode
- record/replay
- change user settings



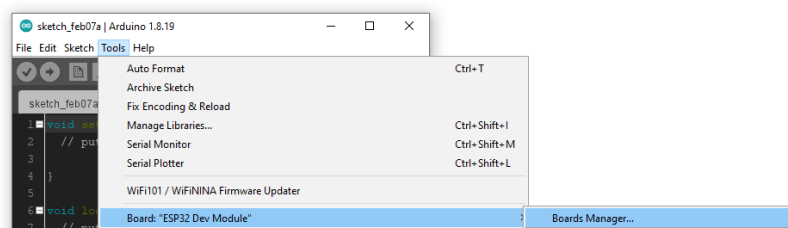
## 2 Installation

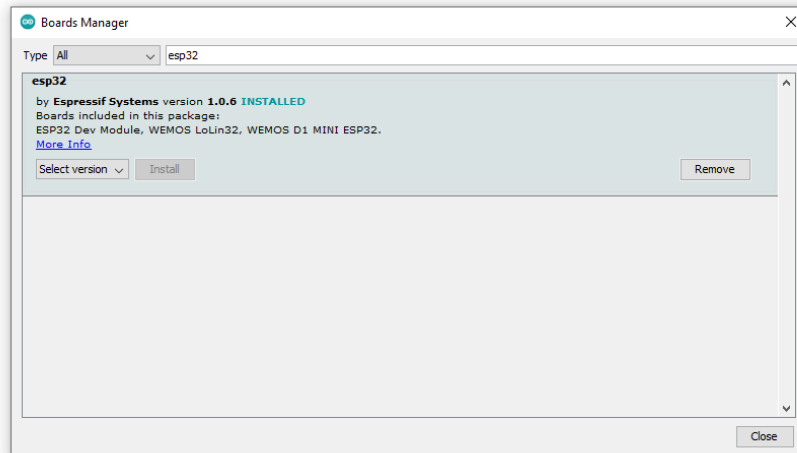
Follow these steps to start with ESP32-DevkitC-32D in the Arduino IDE:

1. Download and install the latest Arduino IDE version from:  
<https://www.arduino.cc/en/Main/Software>
2. In File>Preferences you will find an input box called *Additional Boards Manager URLs*:  
Please copy the following link into this text box:  
**[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)**

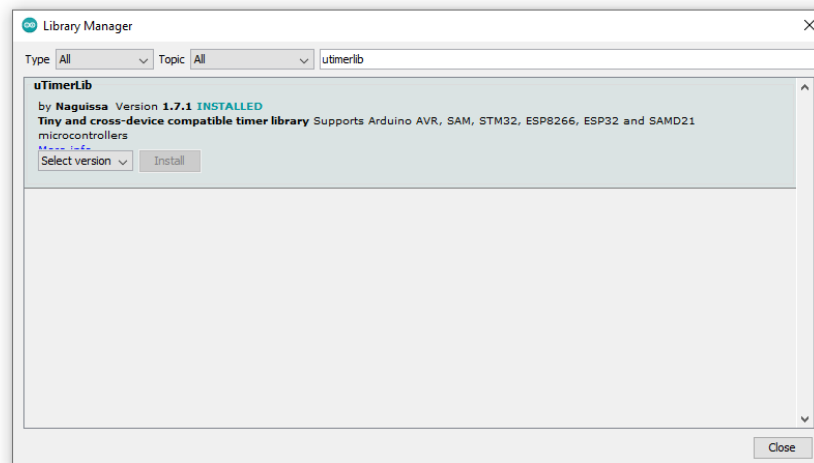
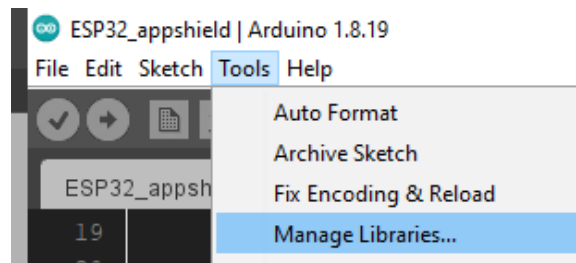


3. In **Tools > Board > Boards Manager** you have to search for **esp32** and install the board library from **Espressif systems**

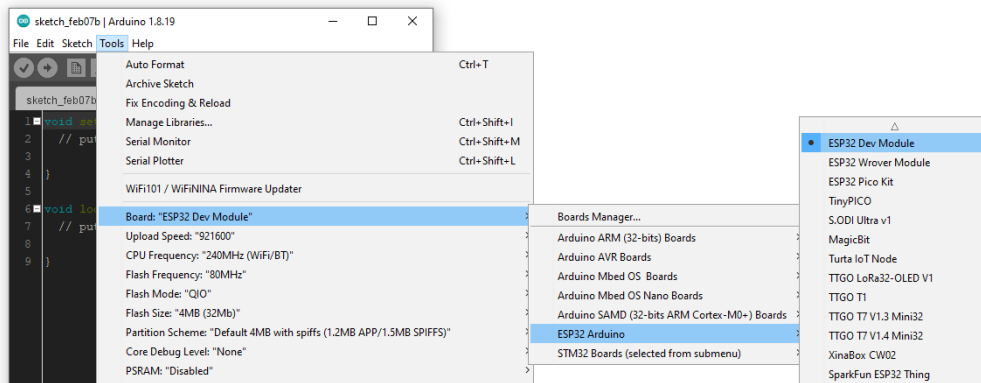




4. Open the Library Manager in **Tools > Manage Libraries...** and search/install the library called **uTimerLib**

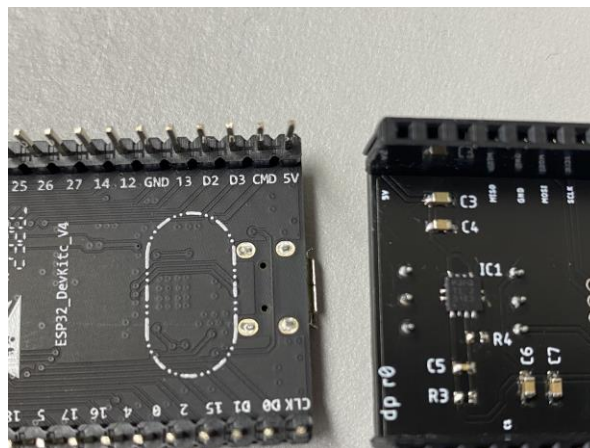


From now you are able to program the ESP32 board with the Arduino IDE. You will find it as **ESP Dev Module**. Please don't change the default setting like Upload Speed, CPU Frequency, ...



### 3 Connect the Application Shield

To get the correct orientation between ESP32 board and Application Shield, have a look on the bottom side of both PCBs. There is an **5V** label on each PCB.



## 5 Upload the Application Shield firmware

The program folder contains two files

- **ESP32\_appshield.ino** includes the complete source code
- **def.h** includes all required settings and defines for each individual sensor. All def.h files are sorted in a folder structure called all\_sensor\_defs. You have to choose the correct def.h for your sensor.

Open the program by double-clicking the .ino-file.

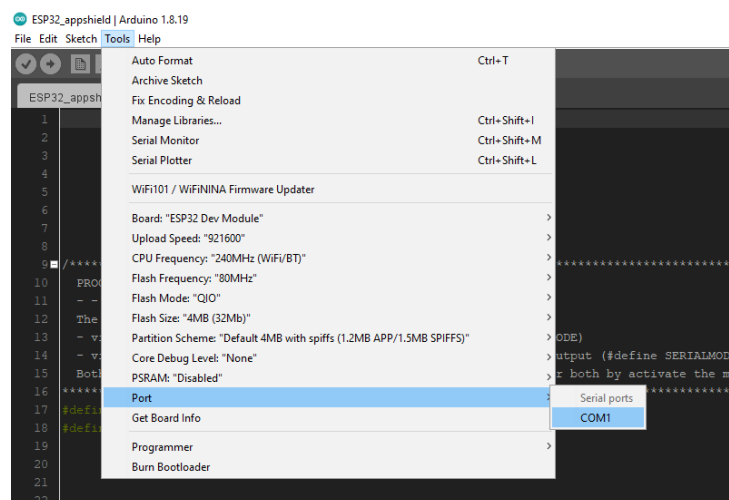
At the top of the source code you will find the two defines to enable/disable the **WIFI** and/or the **Serial** mode. You can choose one or both modes.

```
/******  
PROGRAM BEHAVIOUR  
- - - - -  
The source code includes two ways to interact with the sensor:  
- via WIFI you can stream thermal images in our GUI (#define WIFIMODE)  
- via the serial monitor you can observe the sensor data as text output (#define SERIALMODE)  
Both modes are contain in the same code and you can activate one or both by activate the matching define.  
*****/  
#define WIFIMODE  
#define SERIALMODE
```

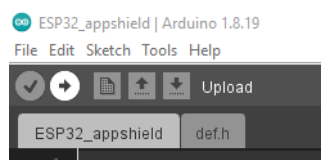
If you want to use the WIFI part, you have to change the variables **ssid** and **pass**.

```
/******  
NETWORK INFORMATION  
- - - - -  
If you want to use the WIFI function, you have to change ssid and pass.  
*****/  
char ssid[] = "yourSSID";  
char pass[] = "yourPassword";
```

Select the matching COM number...



... and upload the program.





If the upload is done, you are able to interact with the sensor with the chosen communication form (serial and/or WIFI).

### **Serial communication**

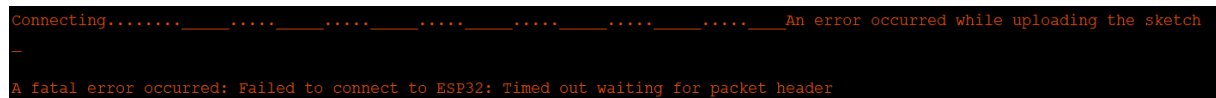
Open the serial monitor and send the character “m” to see the menu. The menu will show you what options are possible with the serial monitor.

### **WIFI communication**

To connect the ESP32: press on “Search UDP”. Now you find a PLAY button to start the stream. For more information: please have a look into the user manual of our GUI “Heimann Sensor ArraySoft v2”.

### **Possible Errors during uploading**

If you get this error message:

A screenshot of a serial monitor window with a black background and orange text. The text shows a connection attempt that failed. The first line is "Connecting....." followed by a series of dots and then "An error occurred while uploading the sketch". The second line is a single hyphen "-". The third line is "A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header".

```
Connecting..... An error occurred while uploading the sketch
-
A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header
```

You only have to press the Boot-button on the ESP32-board during the this ....\_\_ part.

## 4 A short description of the source code

The program will first check if there is a sensor plugged in the board and if the WIFI connection is possible. If both are true, the ESP32 reads the complete EEPROM/Flash and starts polling sensor data without any output. The sensor will permanently “run” in the background to reach a state of thermal stability. A software timer interrupts all other functions if there are new sensor data available. There are two bigger arrays to handle all pixel data. One saves new raw pixel data from the single blocks in 8bit format and the second array (nxm size; depends on the chosen sensor) contains the sorted pixel data in 16bit format. The second one will be used for the pixel data compensation like explain in the datasheet. If there is a request for an output (via GUI and/or serial monitor) the ESP32 will handle that after the next fully compensated image.

Besides the declaration part at the top of the source code, the complete code can be divided in four parts for better understanding:

### **part 1: Arduino function**

This part include the both Arduino specific functions: setup() and loop(). Both are required for each Arduino code. The setup() function is only called once at the beginning and the loop() works as the main-loop of this code. Also in part 1 are functions to handle the LED on the board and the interrupt service routine.

### **part 2: HTPAd functions**

This part contains all functions to handle the data from thermopile array. Here you will find:

- the routine to read the whole EEPROM/Flash
- the read routine to get new raw pixel voltages from the sensor
- the sort routine to convert the 8bit block data to the 16bit nxm array (size depends on the array size)
- all calculation steps to get the final pixel temperatures in dK (deci Kelvin)
- pixel masking routine (if the sensor has defect pixel)
- write examples to write in the EEPROM (not for devices with flash)

### **part 3: WIFI functions**

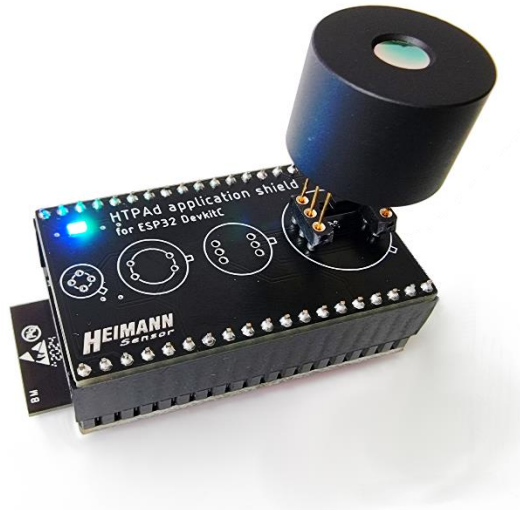
In this part the WIFI will be initialized. The code checks incoming UDP packets and chats with our GUIv2.

### **Part 4: Serial functions**

This part handles the whole serial communication between ESP32 and Serial Monitor of the Arduino IDE. This part contains functions to print the menu, all EEPROM/Flash values and the sensor data in mxn format.

## 5 LED status

There is a three color LED in the upper left corner on the PCB. The following table lists the meaning for each LED state



LED state	Meaning
<b>BLUE</b> (constant)	ESP32 is trying to connect with the WIFI network. If this state doesn't change, please check your WIFI user settings in the code and/or the WIFI setup in your environment. A normal duration for this state is 10 – 60 seconds.
<b>BLUE</b> (blinking)	ESP32 is connected to the WIFI network. Now you can chat with the ESP32 via GUIv2 and/or Serial Monitor.
<b>Light BLUE</b> (blinking)	The GUIv2 called for an UDP device and bound this ESP32. The ESP32 only communicates to this GUI from now. You cannot find this device with other GUIs yet. Closing the GUI will release the device.
<b>GREEN</b> (blinking)	ESP32 sends the thermal images via UDP packets to the GUI and the WIFI signal strength is very good. This is the perfect state to stream continuously.
<b>RED</b> (blinking)	ESP32 sends the thermal images via UDP packets to the GUI and the WIFI signal strength is very poor. It's not guaranteed that the UDP packets will receive correctly from the GUI. Please have a look on your WIFI network and increase the signal strength (for example by using a shorter distance to the router).
<b>RED</b> (constant)	The def.h file doesn't match with the connected HTPAd. Please choose the correct def.h.

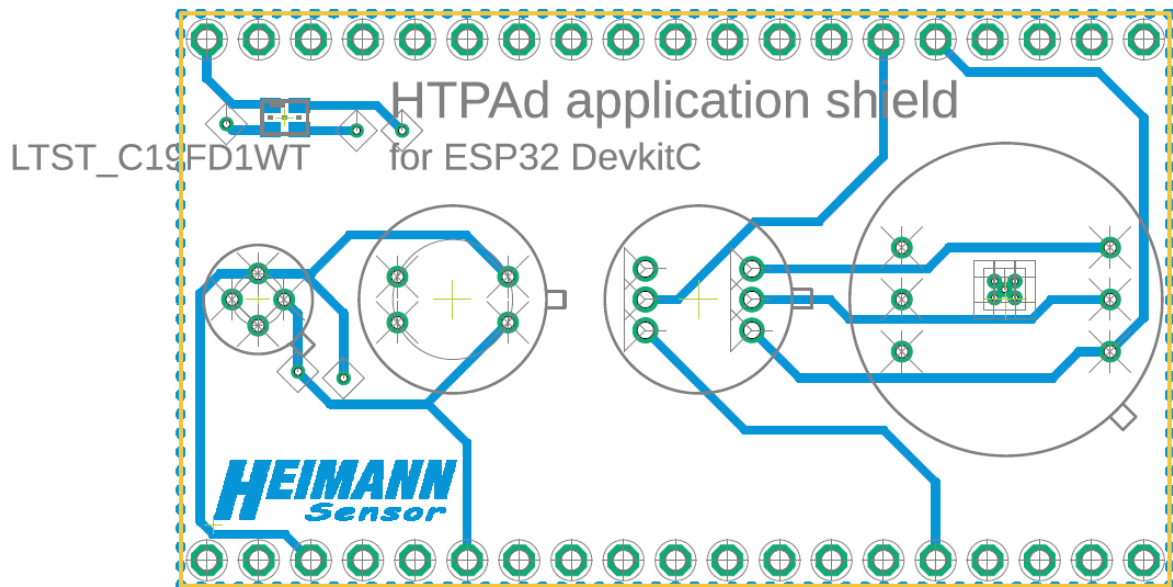
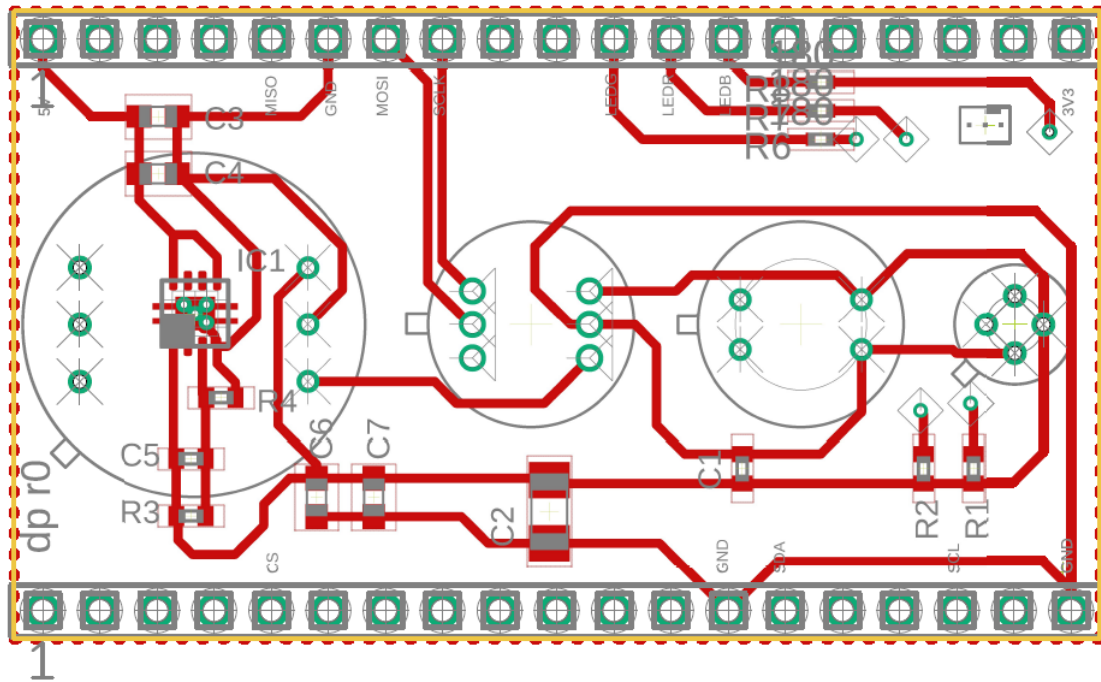
## 6 Possible Errors

### Wrong Lookup table

If you see a little LUT sign in the right upper corner of the thermal image, you included the wrong def.h file for your sensor. Please choose the correct def.h for your sensor and replace it with the current one in you program folder.



## 7 PCB design



## 8 Schematic

