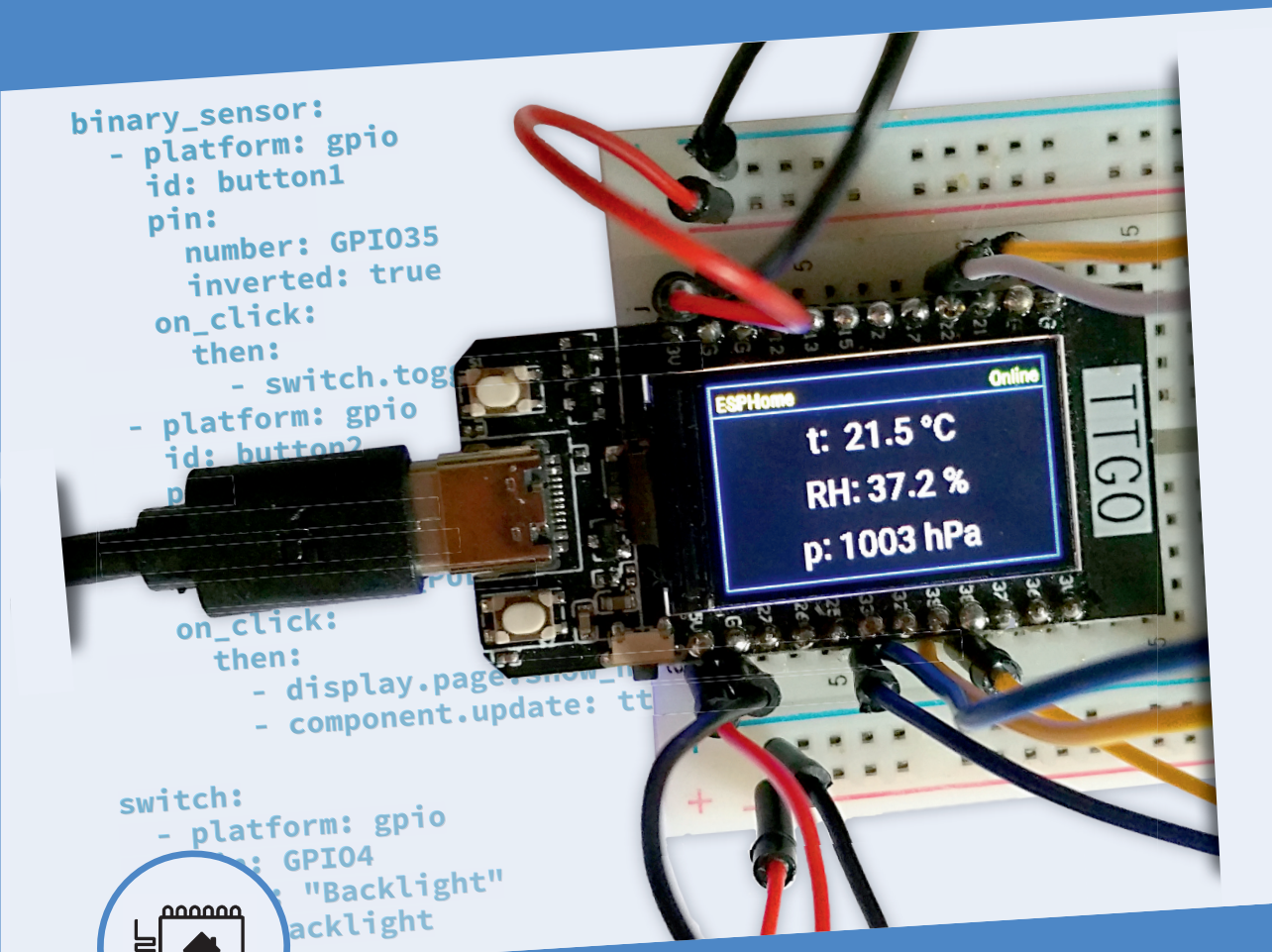
elektor books

Getting Started with ESPHome

Develop your own custom home automation devices



Koen Vervloesem

elektor
design > share > sell

Getting Started with ESPHome

Koen Vervloesem

design > share > sell

an Elektor Publication

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street
London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

First published in the United Kingdom 2021

● British Library Cataloguing in Publication Data
Catalogue record for this book is available from the British Library

● ISBN: 978-3-89576-441-7

● EISBN: 978-3-89576-442-4

Prepress production: DMC ¦ daverid.com
Printed in the Netherlands by Ipskamp

● Preface

Espressif's ESP8266 and ESP32 microcontrollers have become popular for do-it-yourself home automation enthusiasts. If you want to add Wi-Fi connectivity to electronics projects, both microcontrollers are the first that come to mind. They have brought do-it-yourself home automation to the masses.

Not everyone can program these microcontrollers using Espressif's C/C++ SDK, Arduino core, or MicroPython. This is where ESPHome comes in: with this project you don't program your microcontroller, but configure it. Under the hood, ESPHome translates your configuration to C++ code to run on the microcontroller.

In this book, I'll show you how to create your own home automation devices with ESPHome on an ESP32 microcontroller. You'll learn how to use buttons, LEDs, how to sound a buzzer and play melodies, and how to measure various types of sensors. You'll also learn how to communicate over a short distance with NFC, infrared and Bluetooth Low Energy, as well as how to show information on various types of displays.

Most of all, you'll learn how to combine these components and automate everything they do. This way you create completely autonomous home automation devices that you can connect over Wi-Fi to your home automation gateway such as Home Assistant or an MQTT broker.

No home is the same, so no home automation device should be the same. At the end of this book, you'll be able to create your own custom home automation devices the way you like them. Thanks to ESPHome and the ESP32, this is within everyone's grasp.

Koen Vervloesem, 2021

Table of Contents

Chapter 1 ● Introduction

In this book, you'll learn how to create:

- A motion sensor that sounds an alarm on detecting movement.
- A night light that lights up your hallway at night.
- A soil moisture sensor that warns you when your plants lack water.
- An infrared transmitter that can power off or mute your TV.
- A receiver for Bluetooth Low Energy sensor measurements.
- A network clock that always shows the correct time.
- An NFC card scanner that identifies you.
- A distance sensor that shows the distance to an object or water level in a tank on an LED bar.
- A dashboard showing measurements from other sensors in your home automation system.

Before you start working with ESPHome, it's important to take a step back and have a look at what ESPHome is and why you should use it. This chapter also lists some requirements: the hardware you need to install ESPHome on, and the software to work with ESPHome. Before you continue with the rest of the book, you should make sure that these requirements are met.

1.1 ● Configuring instead of programming

The ESP8266 and its successor, the ESP32, are a series of low-cost microcontrollers with integrated Wi-Fi (for both series) and Bluetooth (for the ESP32), produced by Espressif Systems. The maker community quickly adopted these microcontrollers for tasks where an Arduino didn't suffice.[1]

You can program the ESP8266 and ESP32 using Espressif's SDK, Arduino core, or MicroPython. Arduino and MicroPython lower the bar significantly, but it still takes some programming experience to build solutions with these microcontrollers.

One of the domains in which the ESP8266 and ESP32 have become popular is in the DIY (do-it-yourself) home automation scene. You just have to connect a sensor, switch, LED, or display to a microcontroller board, program it, and there you have it: your customised home automation device.

However, "programming it" isn't that straightforward as it sounds. For instance, if you're using the Arduino environment, which has a lot of easy-to-use libraries, you still have to know your way around C++.

Luckily there are a couple of projects to make it easier to create firmware for ESP8266 or

1 Basic Arduino models don't have network connectivity, which limits their use for home automation and IoT applications.

ESP32 devices for home automation. One of these is ESPHome.[2]

On its homepage, the ESPHome developers describe it as:

'ESPHome is a system to control your ESP8266/ESP32 by simple yet powerful configuration files and control them remotely through Home Automation systems.'

The fundamental idea of ESPHome is that you don't program your ESP8266 or ESP32 device, but configure it. Often you only have to configure which pins you have connected to a component, such as a sensor. You don't have to initialize the sensor, read its values in a loop, and process them.

Configuration is a completely different mindset than programming. It lowers the bar even more. With ESPHome, everyone can make home automation devices.[3]

Essentially ESPHome creates C++ code based on your configuration. The process looks like this:

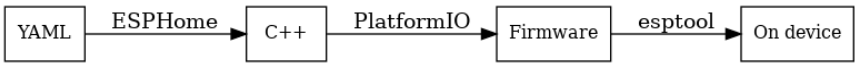


Figure 1.1 ESPhome turns YAML code into firmware on your device

So when you write a YAML file with your device's configuration, ESPHome generates C++ code from it. More specifically, ESPHome creates a PlatformIO project using the Arduino framework. PlatformIO then builds the C++ code, and esptool uploads the resulting firmware to your device.

You don't have to know anything about what's happening under the hood. You just have to write the configuration of your device in a YAML file and memorise a small number of commands to let ESPHome do the rest.

1.2 ● The advantages of ESPHome

Why use ESPHome? The first reason is clear from the project's description: because you don't need to be able to program. Even if you're a programmer, ESPHome offers many advantages:

Works completely locally

Many commercial Wi-Fi-based home automation devices need a connection to a cloud service of the manufacturer. In contrast, ESPHome devices work locally and can communicate with a local home automation system such as Home Assistant or an MQTT-based home automation system.

2 Some well-known alternatives to ESPHome are Tasmota, ESPEasy and Espurna.

3 Note that you can still add your own C++ code to program ESPHome devices if you like.

Offers on-device automations

Many home automation systems use a central gateway that contains all the logic, with automations like "if the sun goes down, close the blinds." In contrast, ESPHome offers powerful on-device automations. Your devices can work independently from a home automation gateway, so they keep working if they lose Wi-Fi access or if your home automation gateway crashes.

Offers over-the-air updates

ESPHome includes out-of-the-box over-the-air (OTA) update functionality. This makes it easy to centrally manage your ESPHome devices and update the firmware. This means you don't have to go around your house with your laptop to connect a serial cable to each device and flash the firmware.

Supports a lot of components

ESPHome supports many components out-of-the-box: several types of sensors, switches, and displays (even e-paper displays) are available with just a couple of configuration lines. The list of supported components is growing with every release.

Has extensive documentation

The developers have documented every component in ESPHome, and this documentation (found on https://esphome.io) is quite good.

Is customisable

Although you create ESPHome firmware by writing a configuration file, ESPHome doesn't hide anything from you. It's still possible to add custom components that you write in C++. You can even look at the C++ code that ESPHome generates and change it.

1.3 ● Requirements

To make the most of ESPHome, you need a few things:

- an ESP8266 or ESP32 device
- some electronic components
- Home Assistant or an MQTT-based home automation system
- a "development" environment

1.3.1 ● ESP8266 or ESP32 device

ESPHome creates custom firmware for the ESP8266 and ESP32 microcontrollers, so you need one of these. There are many types of boards for both microcontrollers, varying in the amount of flash memory, RAM, and available pins. Some of them even come with extras

such as a built-in display (OLED, TFT, or e-paper), battery, or camera.

ESPHome doesn't support all features of all boards out-of-the-box. Technically, all ESP8266/ESP32 devices should be able to run ESPHome. Some features just aren't supported yet.

Your first choice is between the ESP8266 or ESP32. If you're buying a device at present, the choice is simple: the ESP32. It is much more capable than its predecessor and has a faster processor, more memory, more peripherals, and adds Bluetooth.

Note:
The examples in this book use ESP32. If you still have some ESP8266 boards lying around, by all means, use them in your ESPHome projects if they're suitable for the hardware.

Then comes the choice of board. Espressif has some development boards. Many other companies are making them too. There are even complete kits such as the M5Stack series (https://m5stack.com). These are ESP32 development boards ready to use in your living room in a case with a display, buttons, MicroSD card slot, and speaker.

Other interesting devices to run ESPHome on are devices from manufacturers such as Sonoff (https://sonoff.tech) and Shelly (https://shelly.cloud). These come with firmware that works with the manufacturer's cloud services. You can however replace the firmware with ESPHome. This unlocks the full potential of the devices and lets you use them in your local home automation system without any link to a cloud system.

All examples in this book use the TTGO T-Display ESP32 made by LilyGO. You should be able to follow along with most of the examples using any ESP32 or ESP8266 device. The built-in display is only used in the last chapter.

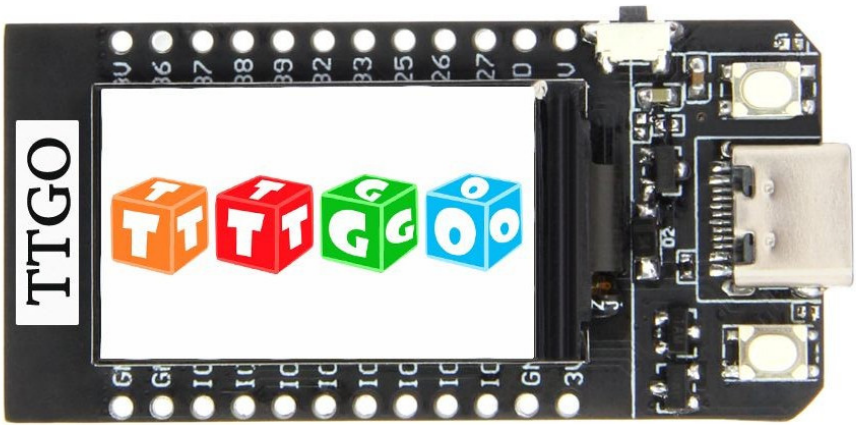


Figure 1.2 The TTGO T-Display ESP32 by LilyGO is an ESP32 board with an integrated 1.14 inch TFT display

Warning:
If you try the examples from this book with another board, make sure you know the pin-out of your device and adapt the pin numbers when required in your ESPHome configurations.

1.3.2 ● Electronic components

The electronic components you need depend on your case for usage. The examples in this book use a small collection of inexpensive, widely available components:

- PN532 NFC/RFID reader
- BME280 3.3 V temperature and humidity sensor
- HC-SR501 PIR sensor
- DS18B20 temperature sensor
- passive buzzer
- TEMT6000 ambient light sensor
- MAX7219 LED dot matrix 8x8
- resistive soil moisture sensor
- NTC MF5A-3 10K thermistor
- TM1637 LED display
- WS2812 stick with 8 RGB LEDs
- 0.96 inch SSD1306 I²C OLED display
- HC-SR04 ultrasonic distance sensor
- TSOP38238 infrared receiver
- 940 nm infrared LED
- BC547 transistor

Added to this, you need some general components which you probably already have if you have completed some electronic projects in the past:

- 830-point breadboard
- 16x m/m jumper wires
- 6x f/m jumper wires
- USB-C to USB-A cable
- Breadboard push-button
- Resistors 100 Ω, 220 Ω, 510 Ω, 1 kΩ, 4.7 kΩ, 5.1 kΩ, 10 kΩ
- Red LED (633 nm)
- DIP switch
- 1 µF 6.3 V electrolytic capacitor

ESPHome supports many more components, including some costly ones. Have a look at the project's homepage for a full list.

1.3.3 ● Home automation system

You can use ESPHome to create a fully autonomous microcontroller project - for example, a plant monitor that turns on an LED if the plant's soil is too dry. However, if you don't publish the plant's status over the network, this would be a waste of the ESP32's capabilities. The main usage cases of ESPHome are:

- To send a device's sensor measurements to a home automation gateway.
- To remotely control a device's lights or switches from a home automation gateway.

ESPHome supports two ways of communication between your device and the home automation gateway:

Native API

The ESPHome native API is a highly optimised network protocol using Google's protocol buffers. It's meant to be used with Home Assistant (https://www.home-assistant.io) - an open-source home automation system.

MQTT

MQTT (Message Queuing Telemetry Transport) is an OASIS standard messaging protocol designed with a lightweight publish/subscribe approach for messages. All your ESPHome devices then communicate with an MQTT broker such as Eclipse Mosquitto (https://mosquitto.org).

From the beginning (when it was still called esphomeyaml), the ESPHome project has been tightly integrated with Home Assistant, so the ESPHome developers prefer the native API. However, MQTT is fully supported, allowing your devices to communicate with many other home automation gateways, as MQTT is a popular standard.

I don't want to force you to use a specific home automation gateway to use the examples in this book, as I'm a big believer in choice. Therefore, the examples in this book use MQTT, but they're also usable if you choose the native API. The differences in configuration are minimal.

Note:
As this book focuses on the ESPHome devices and not on the gateway, it doesn't cover the installation and configuration of Home Assistant or another home automation gateway. You can find installation instructions on Home Assistant's home page (https://www.home-assistant.io/installation/). If you don't want to tie yourself to Home Assistant, consult the book 'Control Your Home with Raspberry Pi' (https://www.elektor.com/control-your-home-with-raspberry-pi) published by Elektor. It covers the installation of the MQTT broker Mosquitto and how to integrate it with various home automation services, including Home Assistant in detail.

1.3.4 ● "Development" environment

With ESPHome you don't program your devices but configure them. However, you still need something that looks like a "development" environment. When your device configurations are simple, you could do without, but the more complex they become, you'll need all the help you can get.

This doesn't mean you have to install a full-blown Integrated Development Environment (IDE). You should only need a couple of programs:

An editor

You could make do with a simple text editor such as Notepad (Windows), TextEdit (macOS), or the default text editor on your Linux distribution. However, having an editor with syntax highlighting for YAML files is easier. Some examples are Notepad++ and Sublime Text. If you're a command-line user on Linux, both vim and Emacs work fine. Use whatever you like, because your editor is an important tool in this book.

A YAML linter

A linter is a program that checks your file for the correct syntax. An editor with syntax highlighting has this linter built-in, but you can also run this standalone. A good YAML linter is the Python program `yamllint` (https://yamllint.readthedocs.io). Not only does it check for syntax validity, but also weird things like key repetitions, as well as cosmetic problems such as line length, trailing spaces, and inconsistent indentation. ESPHome includes its own linter, specifically targeted at finding errors in ESPHome configurations. Both linters are complementary.

If you're used to developing in an IDE, an interesting alternative is the ESPHome plugin for Visual Studio Code.(https://marketplace.visualstudio.com/items?itemName=ESPHome.esphome-vscode). This plugin provides validation and completion of what you type in an ESPHome YAML file. It also shows tooltips with help when you hover over keywords in the configuration.

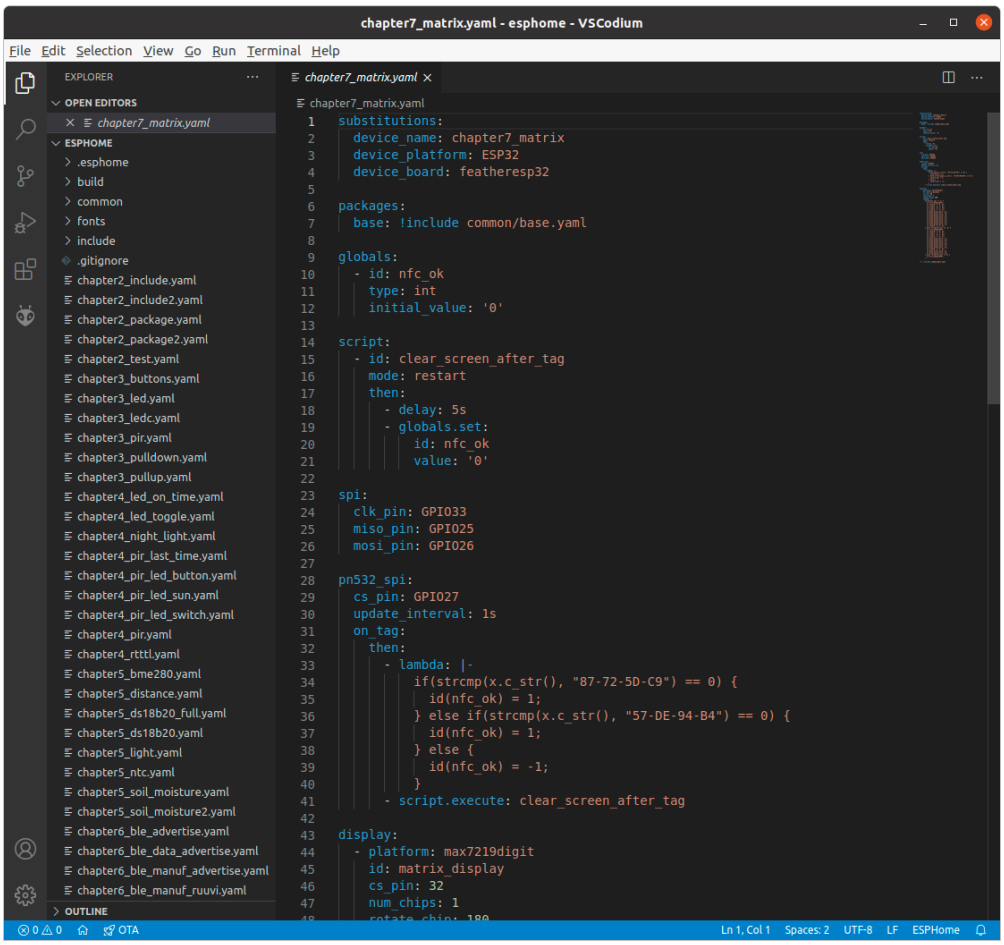


Figure 1.3 The ESPHome plugin for Visual Studio Code has some useful features like tooltips and validation and completion of the YAML code.

The next chapter shows you how to install ESPHome standalone. You can also install ESPHome as a dashboard (in a Docker container or as a Home Assistant add-on). The latter also lets you edit your ESPHome configuration files from a web interface. Because I don't want to tie you to Docker or Home Assistant, this book uses the standalone installation. Feel free to use the other methods.

1.4 How to use this book

This book describes a basic set of electronic components you can use to create your own home automation devices with ESPHome. This is by no means meant to be complete. I selected these components to be able to explain as many features of ESPHome as possible with a small set of cheap components.

After completing this book, you will have enough experience with ESPHome to start adding

other electronic components. You should be able to create your own personal devices that make your home more comfortable.

Some basic electronic knowledge is recommended to follow the examples in this book. But even without this knowledge, they aren't that difficult, and I've tried to explain most non-trivial electronics. The small electronic circuits in this book don't work with mains electricity and are safe to use. That said, if it's all new to you, I recommend you to buy a basic electronics book.

Here's a short overview of what this book covers:

Chapter 1: Introduction

An introduction to what ESPHome is, why you use it, and what you need to create your own home automation devices.

Chapter 2: Preparing your ESPHome environment

The preparation for this book, where you install ESPHome and its dashboard, flash your first firmware, and learn about over-the-air updates and logs. This chapter also gives some best practices to keep your ESPHome configurations maintainable.

Chapter 3: Simple digital input and output

Your first hardware project with ESPHome, where you learn how to use the power of digital input and output and connect buttons, motion sensors, and LEDs.

Chapter 4: Automations

Automations linking various ESPHome components to each other, which makes your ESP32 device able to do stuff without depending on a home automation gateway.

Chapter 5: Sensors

Various types of sensors, including analog, 1-Wire, I²C, and ultrasonic distance.

Chapter 6: Remote communication

Communication methods over a distance other than Wi-Fi, including NFC, infrared light, and Bluetooth Low Energy.

Chapter 7: Displays

Various types of displays, from simple RGB LED sticks, 4-digit displays to a matrix, OLED, or the TTGO T-Display's built-in display.