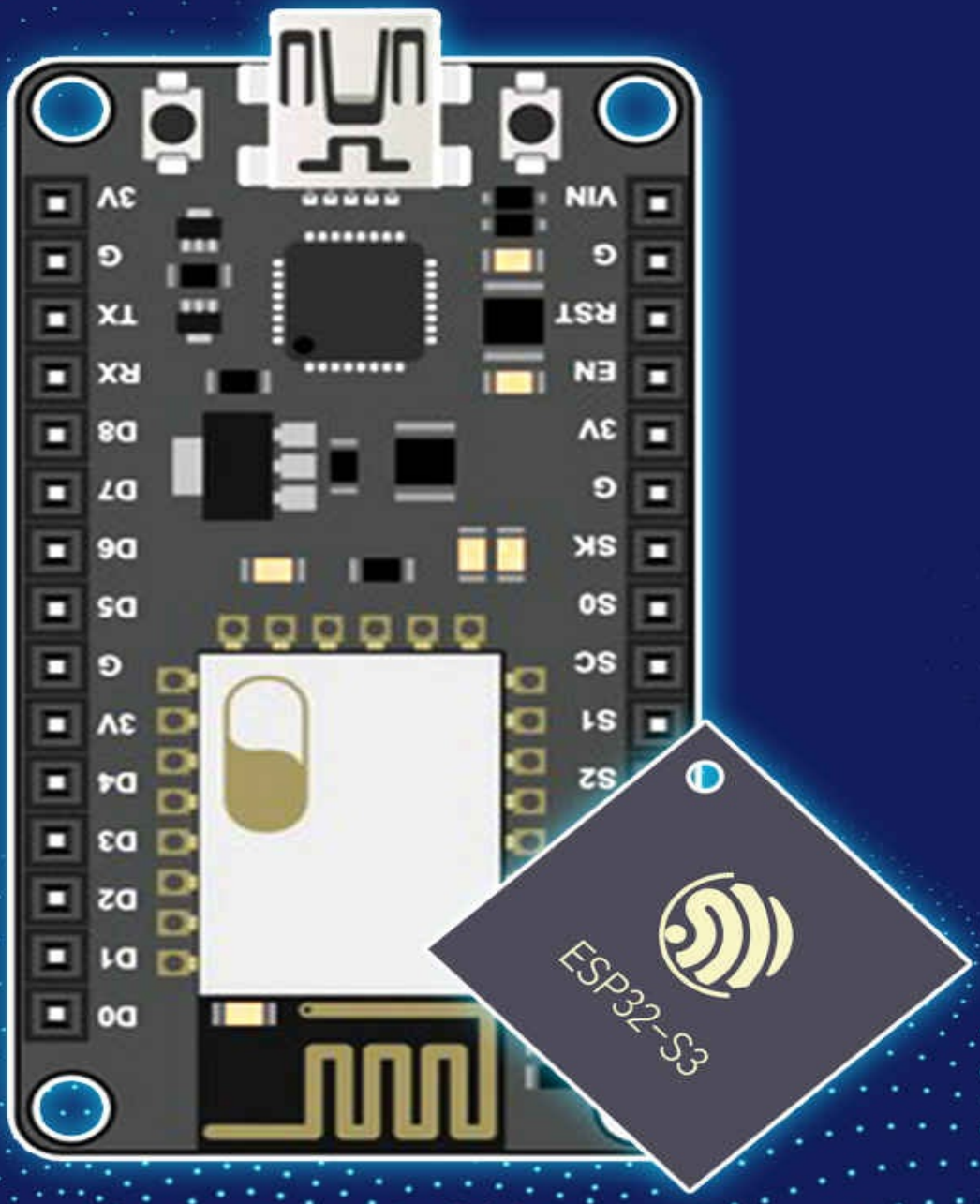
LEARN ESP32 WITH ARDUINO

MQTT, Arduino Coding, ESP32 Coding, Circuit Diagram, IoT Projects



LEARN ESP32 WITH ARDUINO

MQTT, Arduino Coding, ESP32 Coding, Circuit Diagram, IoT Projects

By

Janani Sathish

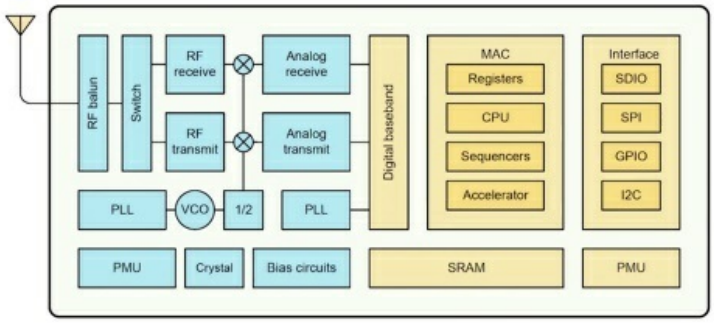
Table of Contents

GETTING TO KNOW YOUR ESP32

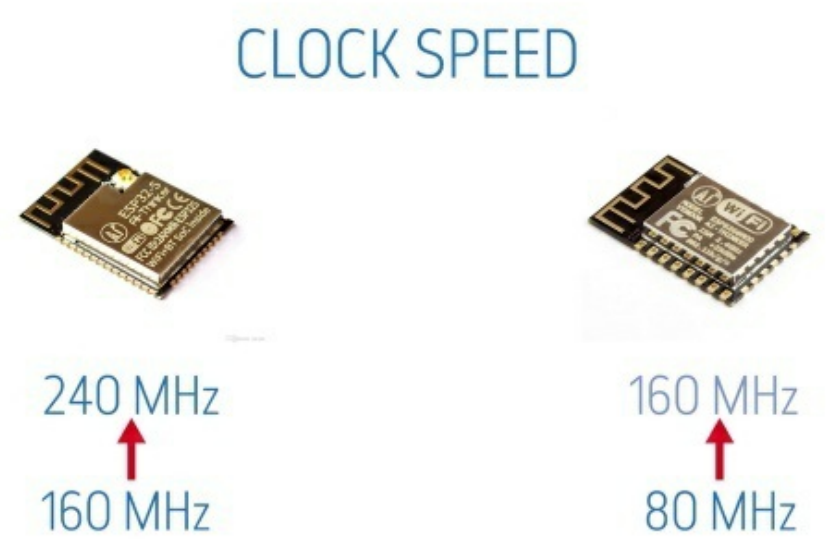
I want to go over some of the reasons why, in my opinion, the ESP 32 is an incredible microcontroller and why you should use it in your IoT projects. For starters, the ESP 32 is very powerful. It contains a dual-core CPU that can be clocked at 8,160 or 240 megahertz. That's quite a lot of computing power in a reasonably small. It also has a ULP or ultra low power coprocessor. And this is a much slower process, or they can be used to perform smaller tasks while the big dual-core CPU is in a night of sleep. Now, besides killer processors, the ESB 32 also has a ton of memory. It includes 512 kilobytes of on-chip SRM memory used for data and programs instructions. Besides this there's also support for external memory and depending on your board, that might be as much as four to eight megabytes. This means that the ESP 32 is also suitable for some heavier tasks, like connecting with cameras, recognizing speech streaming data from the internet. And. But the biggest reason why I think this chip is so good is that it has built-in wifi and Bluetooth. So no need for additional radio modules like you would see on most Arduino boards, the ESP 32 is just one chip with everything in one package. The rest of the IO is pretty impressive as well. There is a 12 bit ADC which can be used to measure external voltages. There are 10 touch sensors for detecting capacitive touches, an led power management chip, a hall effect sensor built-in acceleration for encryption. And again, depending on your board, up to 34 programmable GPI open. So basically the ESP 32 is a very versatile chip that can be used for many IoT projects. And if you use all of this power wisely, you can even let it run on a battery for a very long time. we'll focus on it. And finally, I want to mention that the ESB 32 also supports the Arduino framework. If you know how to program an Arduino board, then you already know how to work with the ESB 32. And it also implies that all of the Arduino libraries that already exist

also work on the ESP 32, a huge advantage. So if you're convinced by the ESB 32.

We will learn the following What is ESP32? The features of ESP32 and how it compares to the wildly popular ESP8266. So let’s dive right into the world of the ESP32. The ESP32 is a SOC or system-on-chip microcontroller. It is manufactured by Espressif, a Shanghai-based company with expertise in Low Power IoT solutions. It was introduced at the end of 2016. The ESP32 can be called the successor to the IoT enthusiasts favorite the ESP8266. It is specifically designed for low-power IoT or the Internet of Things applications. So what makes the ESP32 stand? Let’s take a closer look at its primary features and how the ESP32 compares to the ESP8266 Let us look at the CPO first the ESB 32 contains two low-power dense silica extends out 32-bit microprocessors. Let’s look at the CPU first. The ESP32 contains two low power Tensilica Xtensa 32-bit microprocessors



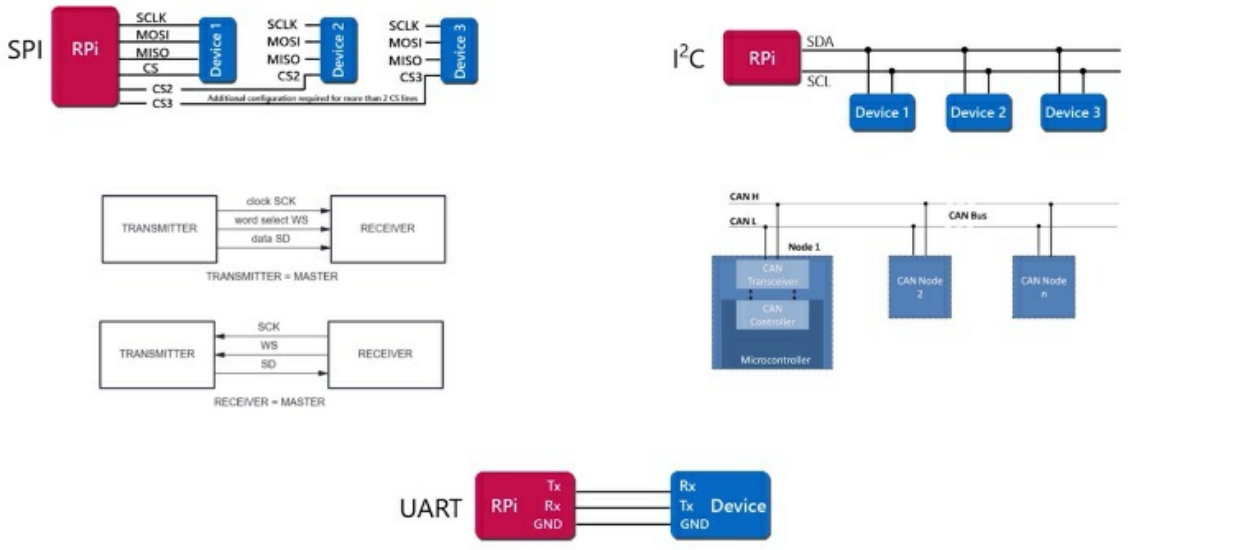
This is an upgrade over its predecessor, the ESP8266, which just had a single-core Tensilica Xtensa 32-bit microprocessor.



The ESP32 is clocked at a faster 160 MHz CPU and can be clocked up to a maximum of 240 MHz. 160 MHz CPU and can be clocked up to a maximum of 240 MHz. All of this means overall better performance and faster computation. The ESP32 has faster Wi-Fi built right into it when compared to the ESP8266.

This is a huge deal because this allows the ESP32 based development boards

to be truly wireless, unlike traditional microcontroller boards like the Arduino. The ESP32 implements full TCP/IP, 802. 11 b/g/n/e/i WLAN MAC protocol. It can connect to most Wi-Fi routers with ease which makes it highly compatible to use. The ESP32 also overcomes what the ESP8266 lacked, inbuilt Bluetooth. Which means integrating it as a standalone unit with Bluetooth. enabled devices like your smartphone are now possible. The ESP32 has dual Bluetooth support, for both Bluetooth version 4. 2 or Bluetooth classic and Bluetooth Bluetooth low energy or Bluetooth Smart. This makes it ideal for low-power applications like wearables, health-based sensors, beacons, etc. It also has built-in sensors like the Hall Effect sensor and temperature sensor. It even has touch-sensitive gpio pins! This makes the ESP32 self-contained without the need for interfacing separate sensors for measuring magnetic fields or temperature. In comparison, the ESP8266 does not have any of these sensors built into it.

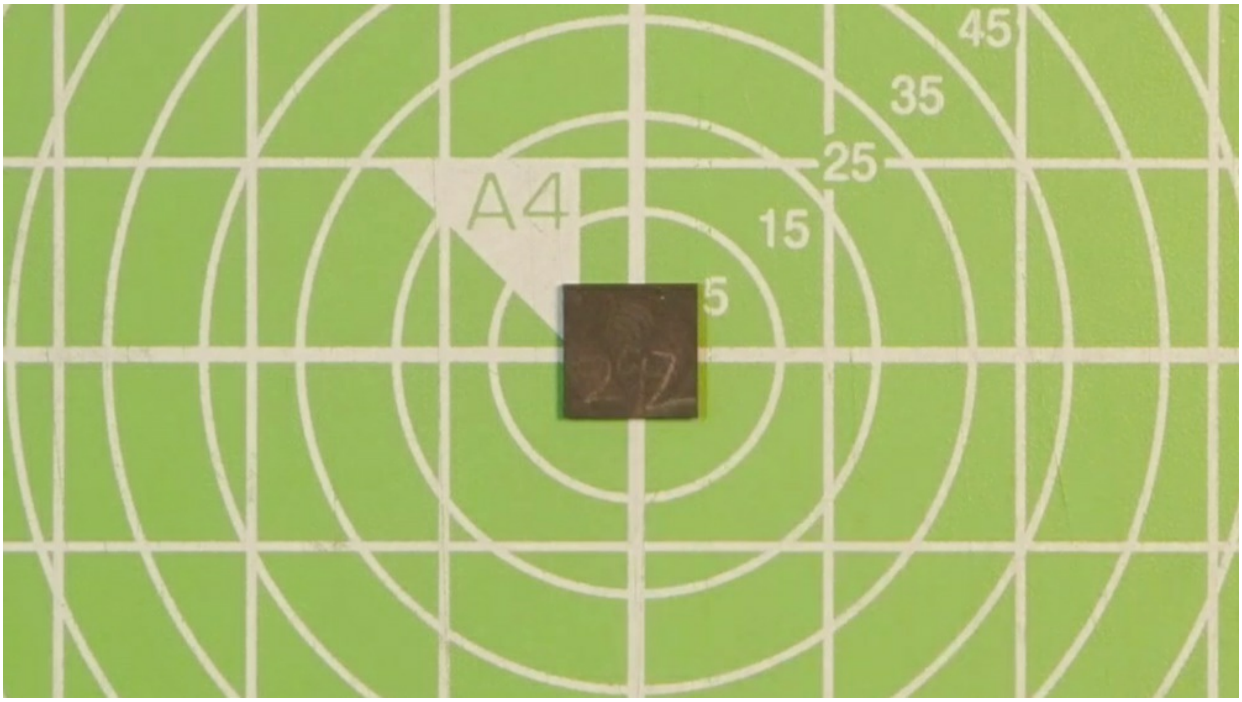


The ESP32 has support for peripheral interfaces like SPI I2C, I2S, CAN, and UART. This allows the ESP32 to interface easily with an extensive range of sensors and actuators which communicate via these protocols One of the more attractive features of the ESP32 is the Deep Sleep mode, which when activated consumes only about 0. 15 uA to 10uA of current.

To put this in perspective, in the normal mode, the ESP32 consumes 240 mA of current, a staggering 24000 times the deep sleep mode current! This translates to your power-hungry projects lasting years on battery power! The ESP32 is powered by 40nm technology, making it highly robust and helping it to satisfy the demands for efficient power usage, security, reliability, and high performance. The ESP32 is jam-packed with features that the ESP8266 lacked It is apparent now that the ESP32 has a significant advantage over the ESP8266 in all aspects All of these features combined make the ESP32 an ideal standalone microcontroller for your mobile wearable and Internet of Things projects. We recommend you to go through the ESP32s datasheet in the resources section to understand the power requirements and the absolute maximum power ratings It would also help you to further explore features in-depth before getting your hands dirty with the actual hardware.

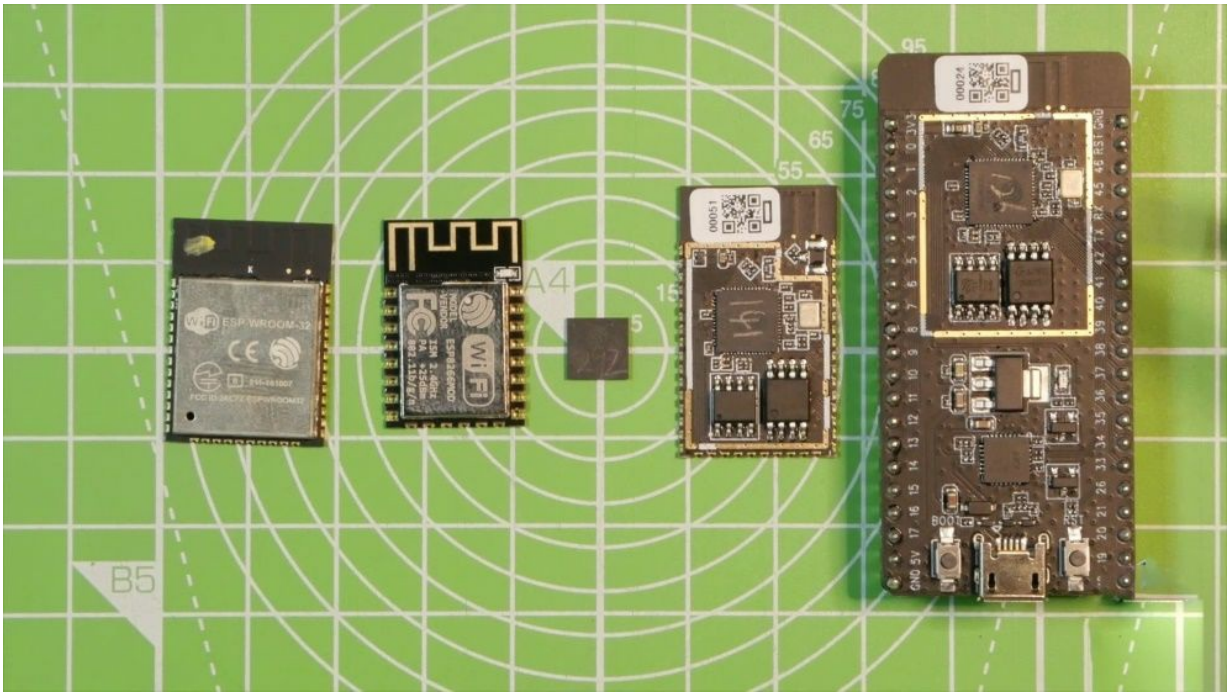
THE NEW ESP32S2 BETA FIRST LOOK

Introduce you to the new ESP 32 S two. And there it is, that's it. That's the new ESP 32 S two. Let's get a little bit closer. It's a little bit better. It's still pretty small. So this is the current beater Silicon of the ESP 32 S two that I received from expressive a few days.



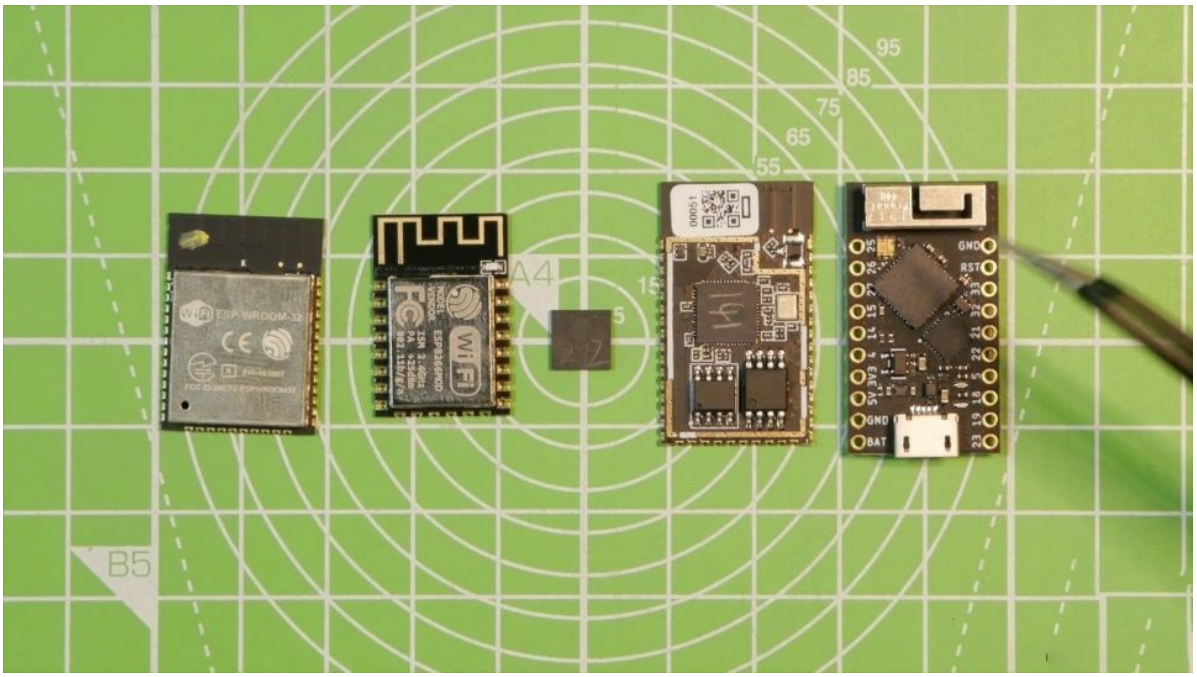
It's not the final Silicon. There are some more changes they're making to it, but that is it. That is the chip. So let's talk about where it sits within their product line. You are an ESP, a 2 6, 6. The chip is designed to be more advanced than the. So it suits after the here is a is P room 32 module. So that's where it sits in the product line. It's way more powerful than an 8, 2, 6, 6, but has less functionality and features in many respects than the ESP 32 that's currently available. But it also has some things that are slightly better than they used to be. We'll talk about what those differences are short, for now, let's discuss how this chip is going to become available to people. This

is the module reference design right now. That's got this chip on it, as you can see right here, all of these chips are numbered, which is pretty cool. So this particular module layout is pretty similar to the Rover. It's probably the same as the Rover. And as you can see, there's some Bram on here. There's some more flat. External crystal and everything else. And there's the room to put the shield over the top. The chances are, even though the chip is still going to go through another revision, at least one revision, that's going to be the shape of the module. And right now there is a reference board that I also got that has that module sitting on here. But obviously, it's not in module format right now. It's just. But that's where the module would be sitting. And one of the features of this particular chip is it will have a built-in USB, but right now that is not currently available in this version of the Silicon. And so the reference board still has a separate CP 2102, to be able to connect the USB to it, to flesh it. But once the final Silicon comes out, there'll be onboard USB. So you won't need to have an external peripheral.



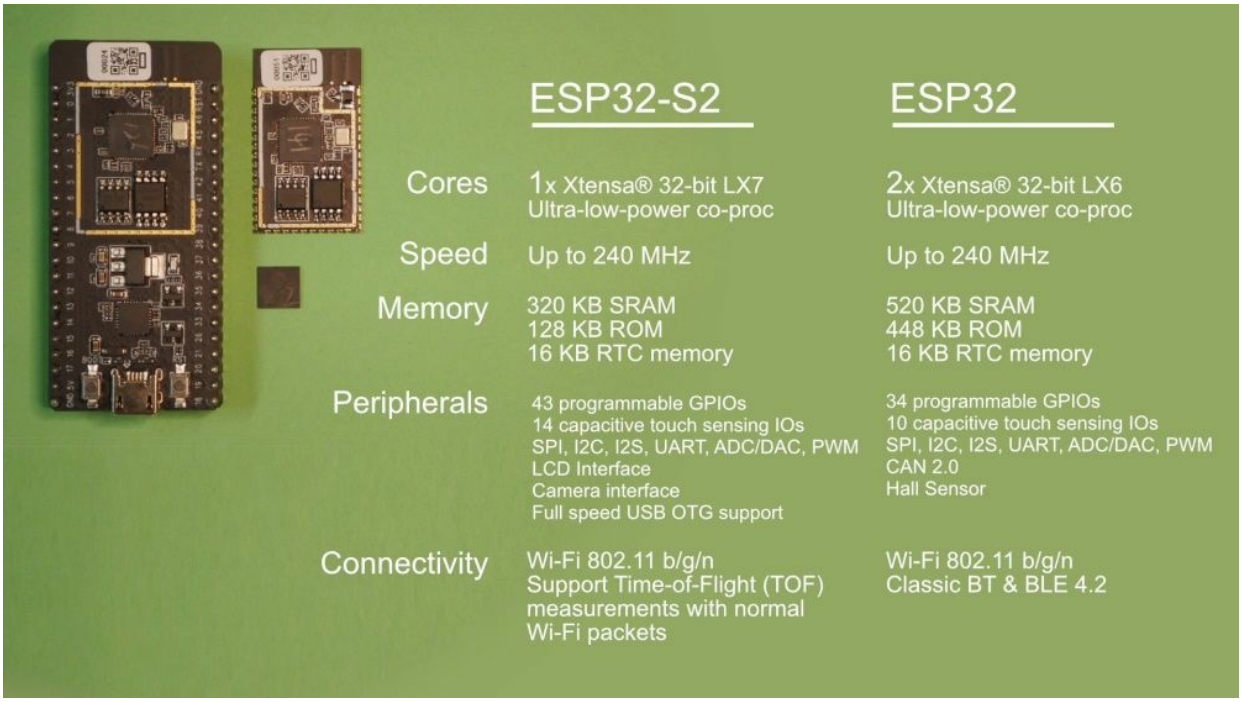
To control and flesh the chip and that onboard USP will also support full-speed USB on the go, which is pretty cool. Now, just to give you a size comparison, here is my tiny Pico. So this is using an ESP 32 Pico D four, which is the same size chip, the same size QFN. But this particular chip has

built-in crystal and flesh. So it's, what's called a sip assistant in a package. Except for one of these chips, except for this chip right here, which is on the bottom of this board, all the rest of everything you see here, except for the antenna is actually inside this chip.



Now I'm hoping in the future, expressive will come out with the new Pico chip. That'll be the S two, but inside a sip package, that'd be great because as you can see, if I look at all these peripherals here, it doesn't save me a lot of space using something like that. On a feature, tiny Pico when I might save by not having the and a couple of transistors, but I'm going to lose that space anyway, by having to put a crystal and a lot of passives there, but that's it. This is the board. Let's now have a look at this. Okay. Let's look at the specs and we're going to do it as a comparison to the ESP 32. So the cause of the STC, it only has one core, not two, but it's using a newer extensor LX, seven architecture. That's supposed to be more performance, both still have the same ultra low power co-present speed. Both go up to 240 megahertz memory, 320 kilobytes of SRAM compared to the five 20 kilobytes of best range. Yeah. Current ASP 32, only 1 28 K of rum compared to 4 48, but both have the 16 K of RTC memory with peripherals. There are more IO on the new S two than you currently find on the ASP 32 there's 43 programmable

GPI owes 14 capacitive touch iOS compared to 10, your usual assortment of spy. I squared C I squared S so forth. The S two has a built-in LCD interface. So you can drive LCDs directly with it. There's a camera interface, so you can drive cameras and there's full speed USB on the go support. I'm not sure about canvas and whole sensors and everything else. I would assume they would still be in there, but I can't find any documentation to support that. Right.



When it comes to connectivity, this is the other big difference between the two chips. There is no more Bluetooth at all. It's just wifi. There is support for the time of flight though, but it's just the straight wifi. 8 0 2 dots 11 BG, N N. Now there are features in the S two that may be sort of different. There are what seems to be new security features, but I'm not going to jump into those right now. So that's it. That's the new ECP, 32 S two. When is everyone are going to be able to get their hands on it? No idea. When is the next revision of the Silicon coming out? I don't know. I can't wait to get my hands on it because although I want to start designing a board using this new chip right now, I can't put one together, even using the speeder Silicon without putting external USB support on which kind of defeats the purpose of the type of

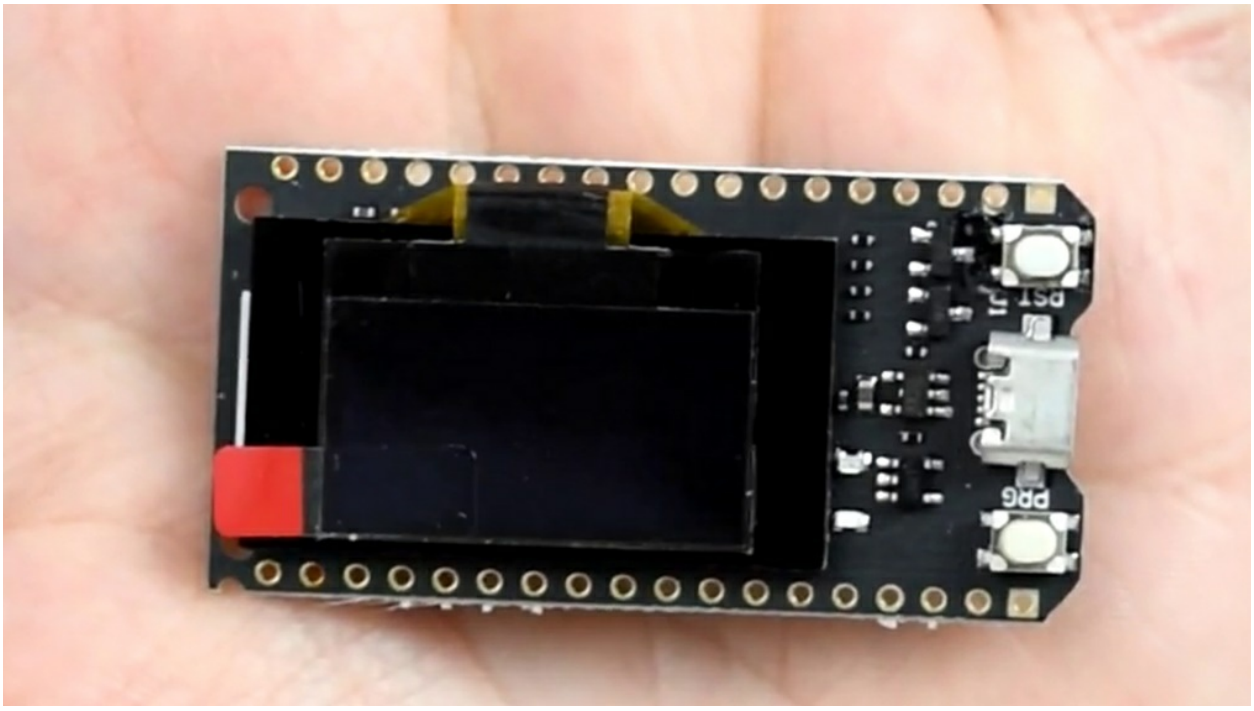
boards I want to make. But it's going to be a killer feature when the final Silicon comes out.

ESP32 IS THE BEST ARDUINO REPLACEMENT

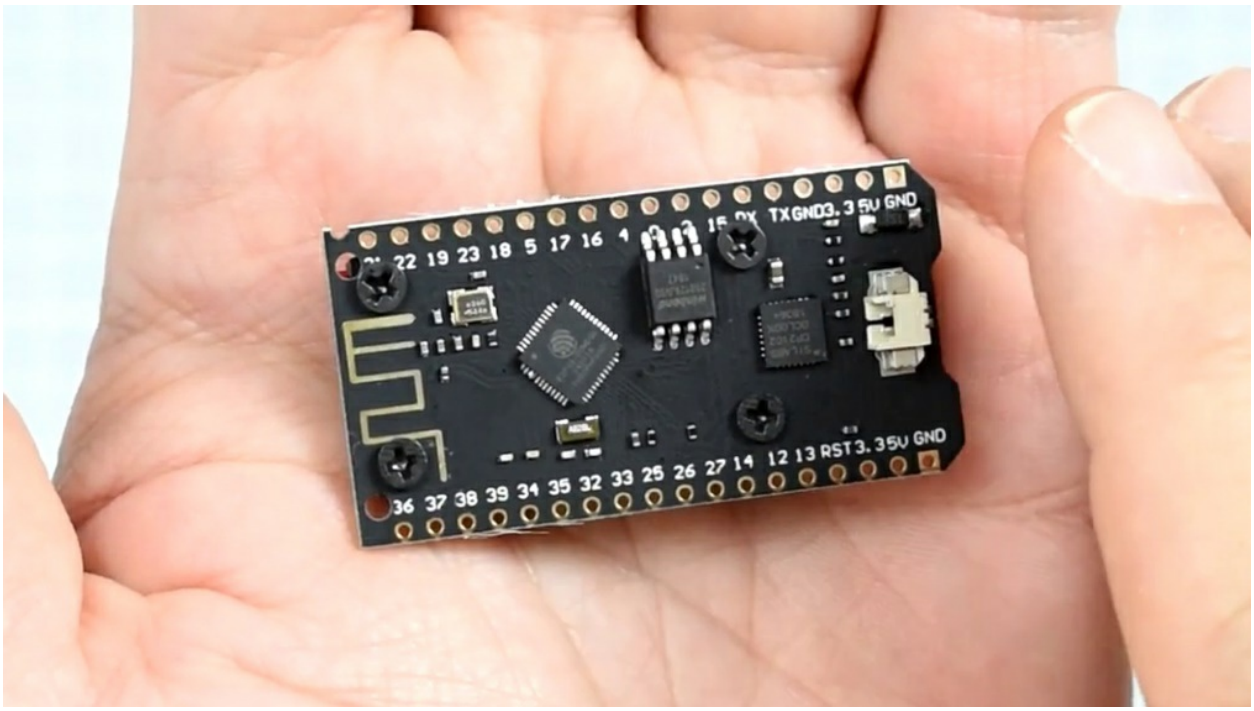
Into building your own electronic devices with the help of the microcontrollers, for example, with the megabase or STM 32, and you still do not know what ESP, ESP 32 east, then this video is definitely for you. Please bear in mind that the topic of the ESP 32 is maybe not the. Topic in the ward because the board itself, the chipset itself appeared on the market four years ago, because somewhere around 2016, but the amount of the possibilities and the developments that were put into this board and how cheap and available they are and which incredible functions they do offer is just like almost mind-blowing, to be honest, I always liked to create me something with the Arduino. n It's there it's easy. It's simply added. There were a lot of libraries, however, well, however, one at one point I discovered that the old ATmega Arduinos clause was okay. Clause of the hardware. Well, they really kind of suck, not too many serial ports, not too many flashes, not too much Ram. Everything is so bloody complicated. When you try to build something more competitive. Requiring from, from the board. And if you, for example, want to connect this to the internet somehow, then the problem starts to amplify ears. I know. Uh, for example, nowadays from the, you know, you can buy those new fancy with the S an M D architecture, but they are kind of like expensive, expensive. And if you compare the expensiveness of those boards with the fact that you can get the ESP 30. Wherever the wifi. Yes. With the wifi, with the Bluetooth. And for example, with the installed, uh, LoRa module running at [00:02:00] 868 megahertz for really long wrench communication for around 10 bucks, maybe 15, if you add all the OAD to it, then it kind of like. Yep. You'll see. There is a difference. And with ESP, you do not only get the internet connectivity out of the box. You also get goodies, like for example, three serial ports out of nowhere for SPI. Am I correct? Um, yes. For SBI buses, you can get, uh, two ITC buses. You can get the, uh, as

the current driver out of. Nowhere are 16 PWM outputs. Yes, they are software PWM, but still PWM outputs and many, many, many other cool features. Like for example, 36 GPA ELLs, not every single one can be used as an output and the input, but this is a completely different, different topic. And all of that for like three bucks lends you the kind of like you start to wonder maybe this is the real.
Yeah, the real way to go forward and to make things even more interesting. All the ESP 30 twos come with a 160 megahertz clock. Yes. More than 500 kilobytes of rum. Yes. And two cores. Yes. Then they are running the free real-time operators. Some have two cores and you can have two things running at once on them and everything is there. And by the way, you can program this thing with everything you want, because it will run or do you know? Yes, you can flush out. We are not in the program with Arduino, not the program is doing about the UN are doing the environment. Uh, you can do that. C just program everything we've seen, just like you wanted, you can use Python and probably no JS and everything. Everything, because the power that lies inside of the ESP chipset is kind of like massive. They are not perfect. They are not perfect because, for example, the majority of them lack any flash memory. So if you want to have a place to store your cult, you have to have the board and the board has to have external. SBA connected flash memory, but. 3 10, 15 bucks. Okay. 20, uh, probably the most expensive ESP 32 board I found it for like 25 bucks, but this one has the power management USP, 32 Bluetooth wifi, of course, all led display and LoRa module, and also a GPS connected and the, uh, liti on 18 650 holders. In there. So I'm like, come on.
It's like almost a full-blown device that you can do whatever you want with this. And that Skype is the link. So what are we going to do later in this video? Later in this video, I will show some of the examples of the boards that I have. Here is not everything because, for example, I do not have on me the example of the simplest one without the only display, I have no idea where I used all of them. Probably every time I just need something running a microcontroller, I just put the ESB 32 over there. And from my stack of probably five or six, I got zero. And I have no idea where the rest is, or

maybe they. Who knows, and then DM and in the future, that also be a video, how to start your adventure with ESP environment. And this is cool. Okay. So, um, let's see what I do have on my workbench today, let's begin a venture with ESP 32 boards.



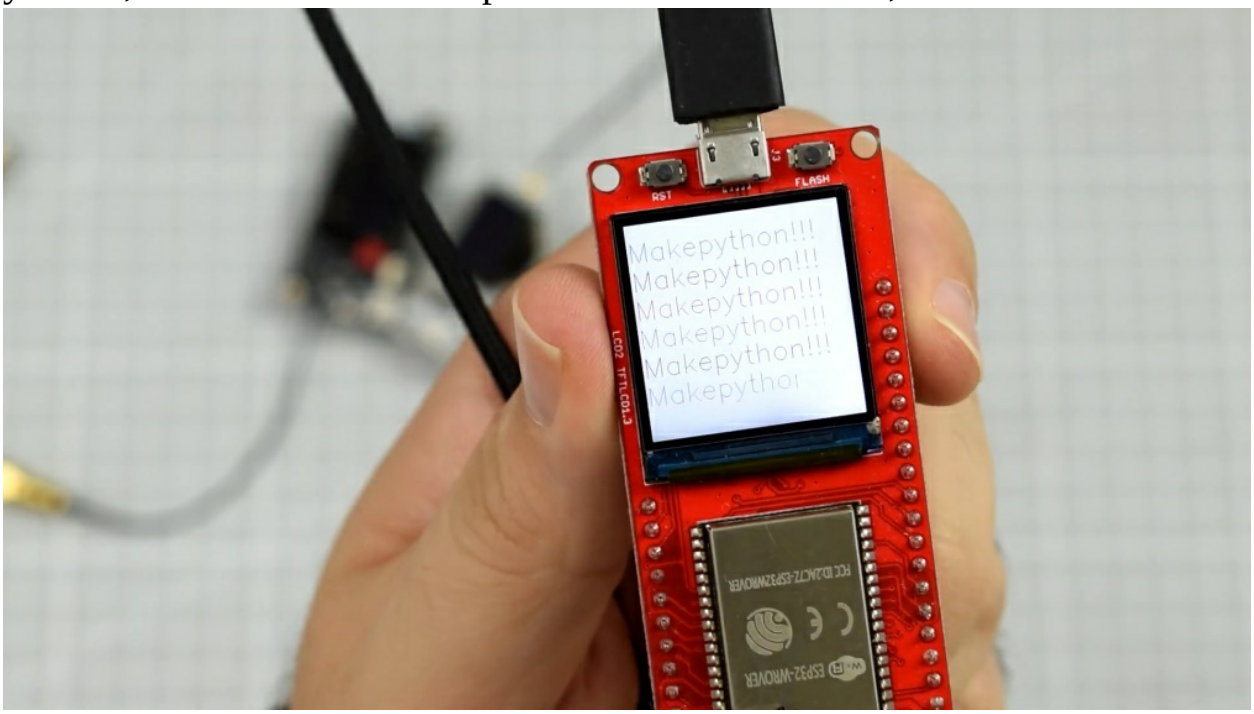
What I have over here is, probably the most interesting. Value for price board that there is right now on the market, which is the ESP 32 in the 30 8:00 PM. Very Shaun, which has USB micro was be connected on my side, two buttons. One of which is, and then the second one can be used just like the button on the board or led display.



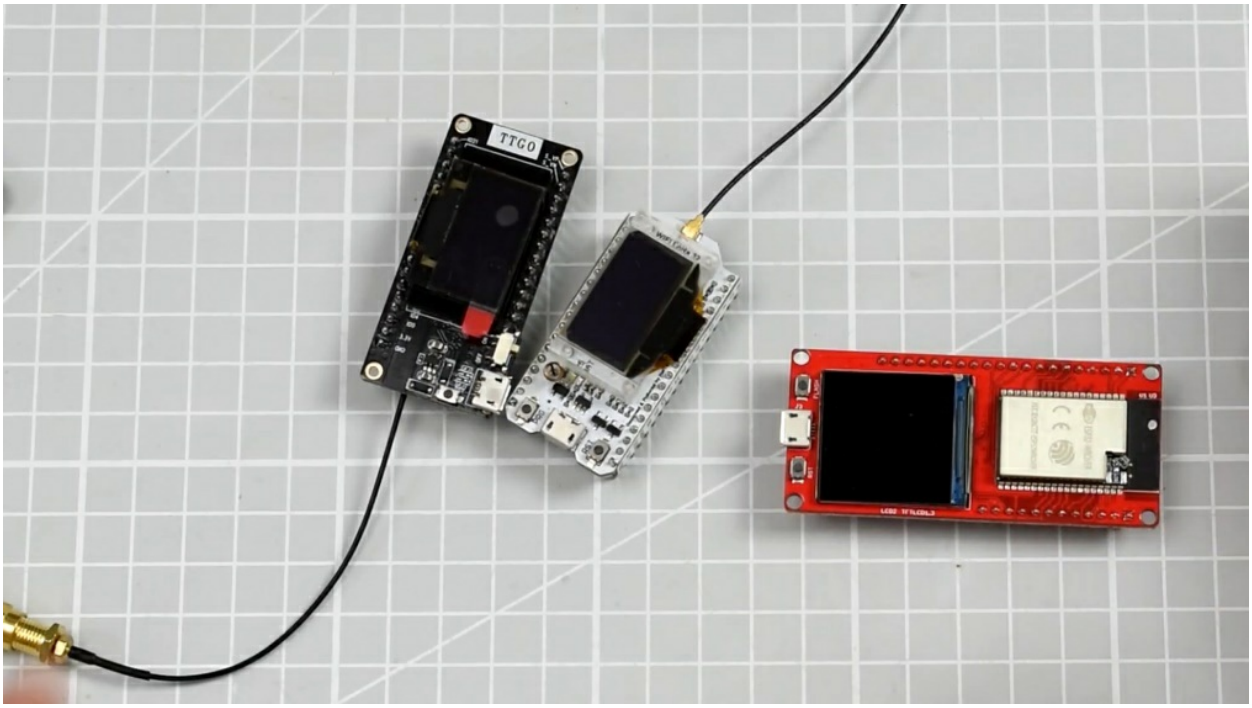
And on the bottom side, we can see there is the wifi and turn up. There is the flash memory. There is a voltage regulation for everything. It accepts five volts and this very interesting connector over here. Thanks to this connector, you can connect any lipo or lithium battery sync one S 3.7 volts that just fit somewhere here and have this thing running as the battery-operated device because this thing has a charging unit. Every time you plug in the USB port, it will just charge the battery and then allow you to use this thing.

Well, without being black for everyone, there are also very similar boards to this one, which is lacking the led display. But if you want to have this thing kind of like interacting with the user, this is rather the way to go. Of course, you can. Not only display to the very Shaun room 32, I think without early D the price difference is not that high, uh, not that big. And if you get yourself feeling. Uh, ESP 32 S and external audits, you can just as well get one with the integrated bearing in mind that the way of working with the early deacon can be different between boards. For example, some of them require you to keep a pin, uh, digital, uh, IO 16 high too, for the, for this to work, some are, uh, flipped and you kind of sometimes need a special version of the library, but. It works. It also has enough flash enough memory and enough computational power to run a small web server without any problems you can connect this thing to the internet. There are certainly plenty of examples, how to use wifi integrated on it, make the request to a server, get the data, or just act as the server to have this thing working together with you or your smartphone. You just either connect to the same network or. Make an access point of the ESP and everything is fine. And then, yeah, this is not enough for you then if you want some kind of the connectivity between two ESPs for a longer render that Ruggles longer ranges than the wifi or Bluetooth allow you

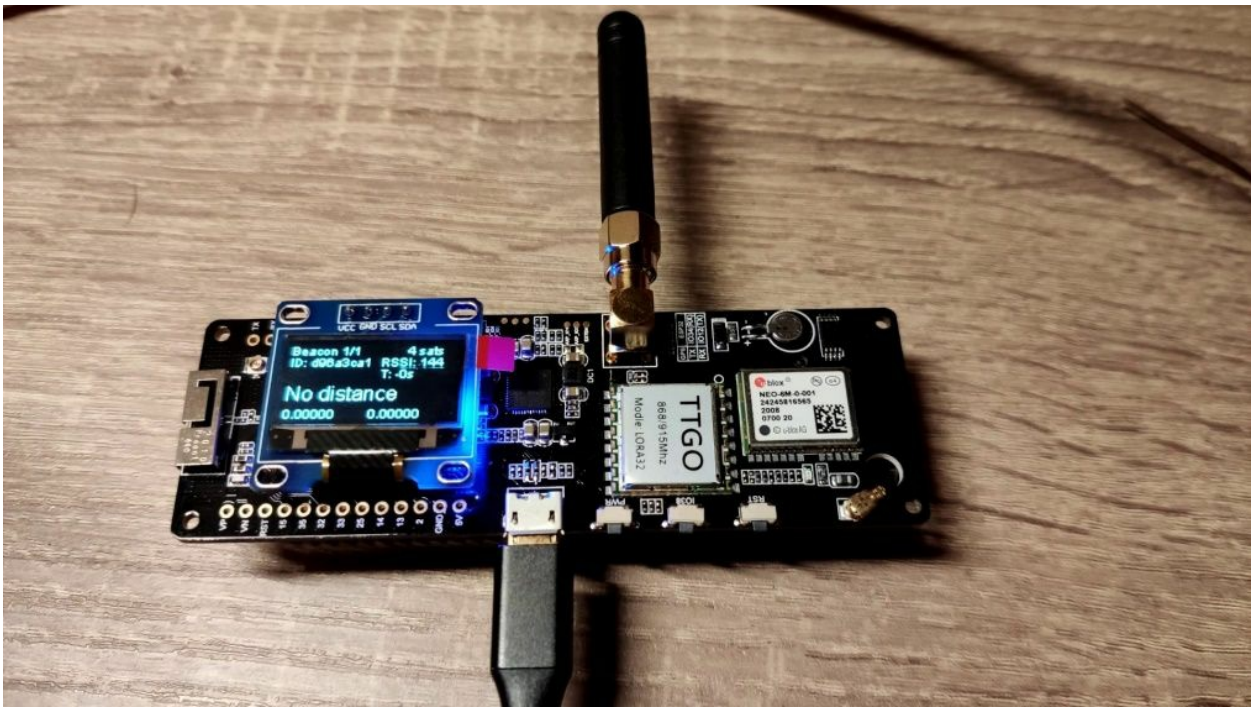
to do, there is this version and something like that. There are of course more variance, but those are the, basically this thing where. Laura integrated here on the bottom site over here, you can find some tech. 1278 or six. I never remember which exactly. And of course the antenna, uh, on, on the pigtail LoRa connectivity. So you can talk really for like incredible distance as few kilometers, at least if there is, um, at least some elevation between the unintended us and have, uh, Spread the network of devices, doing incredible stuff, access to the Laura mash, et cetera, et cetera, et cetera, really the sky is the limit.
Um, I'm currently using a lot of those devices for my project that uses and 32. And Laura, I was using this for the location because I was using this also for the long shred. Your system sky's the limit there. You can also get very shows running on the forehead. 33 megahertz. And I bet you that quite soon, there also will be the versions with the LoRa module running on 2.4 gigahertz that will more or less allow you to build your version of immersion or see a ghost or fly sky FRM three. There are also kind of more advanced versions because for example, what I have over here is the ESP 32, again, but equipped with the color LCD. If I plug the battery in, um, that's what, then you see, it's a color LCD. I hope it will be visible in the, is it visible?



Yeah, it's a color LCD. So if you only want to. Mainly the color and display. And this one, for example, should be out of the box programmed with. But this is not, of course not the only, only choice you have. There, there are also the versions of the ESP 32. I do not have it at hand.



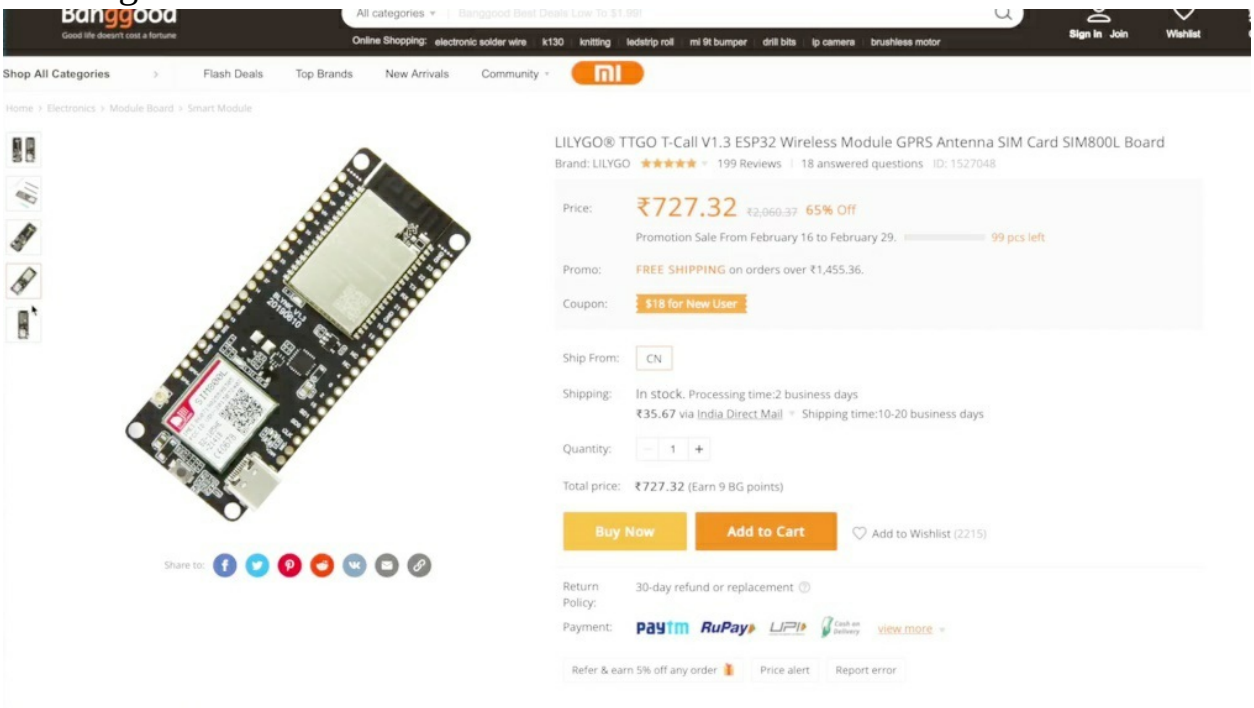
The first one was this form factor about not sorry, this form factor, but without the early display, there is also a much smaller, very strong call to Done or something like that, which has much fewer GPIs and you cannot connect everything. Connections are just not broken out. However, those things like super simple, super cheap, and small, and you can put them everywhere. There are also very shows with the paper displays. There are versions with the Laura or early D GPS power distribution system and a little holder. One more time. Nothing can me, but I do own two of those.



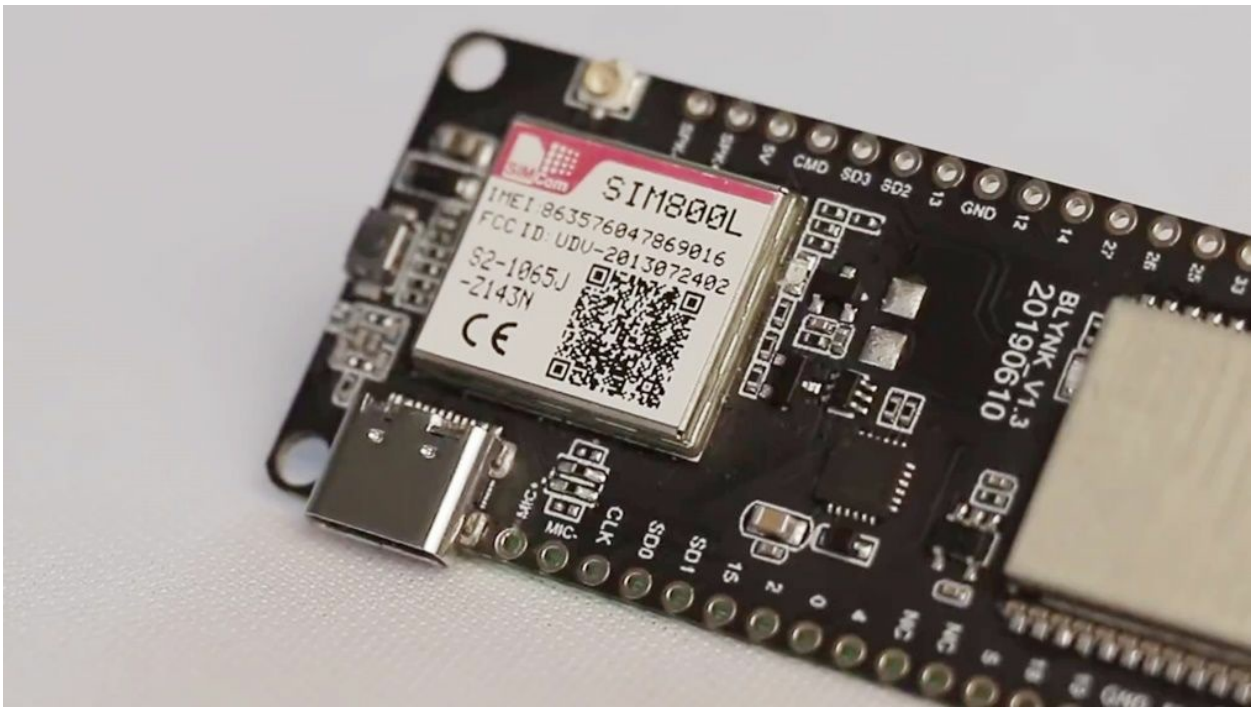
And for 25 bucks, Almost everything, everything you want. If you want to know what you can do with ESP 32 and what kind of incredible hardware you can get, then AliExpress has everything you type ESP 32. And you only, and speaking of which I will probably have to make some purchases right now, because like I said, I'm out of few versions. Those sports bear in mind, the quality of the kind sometimes can be disappointing. Uh, for example, all the versions of the early, uh, installed the early D cracks very easily. And they have like a small box of Thor three without the old one because they just cracked and it was not working. And. I think that's all for today. I hope I got your interest in the ESP still T2. And when you will be thinking about the hardware to use in your next electronic project, you will think about that stuff because it's amazing for the price.

CALL THE BEST IOT BOARD WITH MULTIPLE CONNECTIVITY OPTIONS ESP32 SIM800L

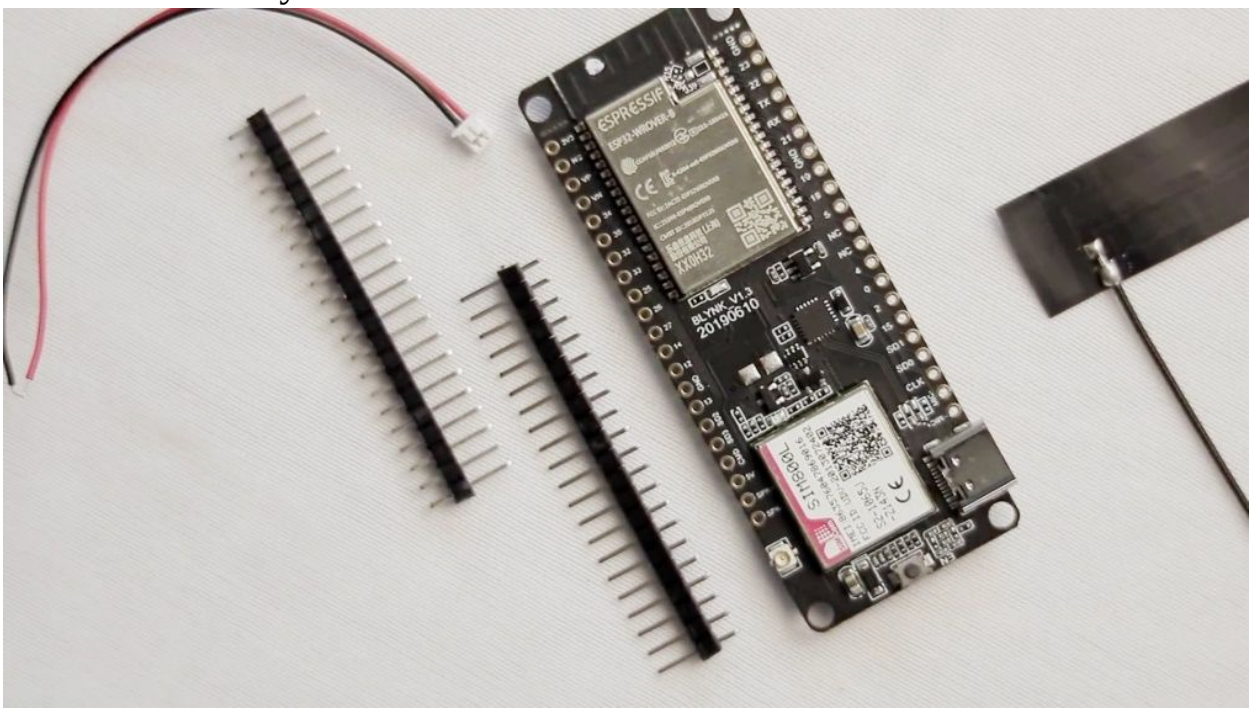
I will let you know how to use the model. And I'll also show you what different kinds of projects you can make with it. So let's get started. This is how you receive this module and talking about how to order. Then I got this morning from bamboo.com.



So along with the hardware, you'll also get one in Dina, some male headers, and one battery connected.

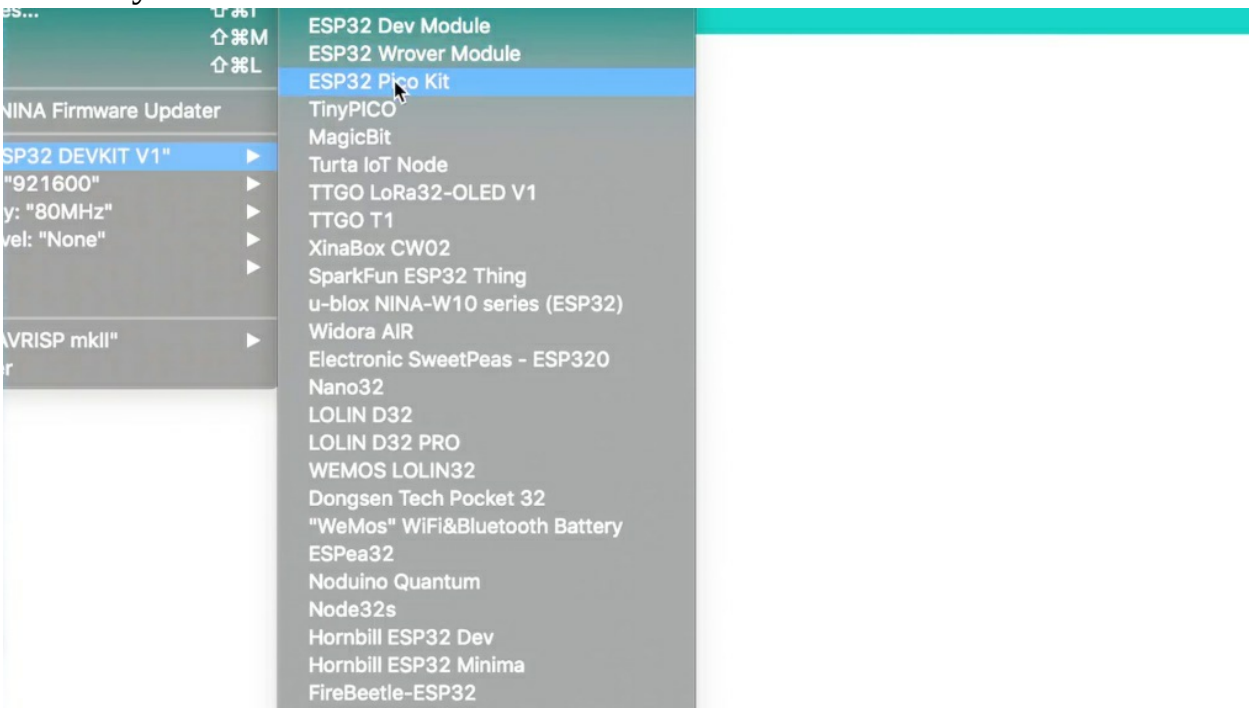


Now talking about the hardware configuration, then this model is based on ESP three to Ruby.



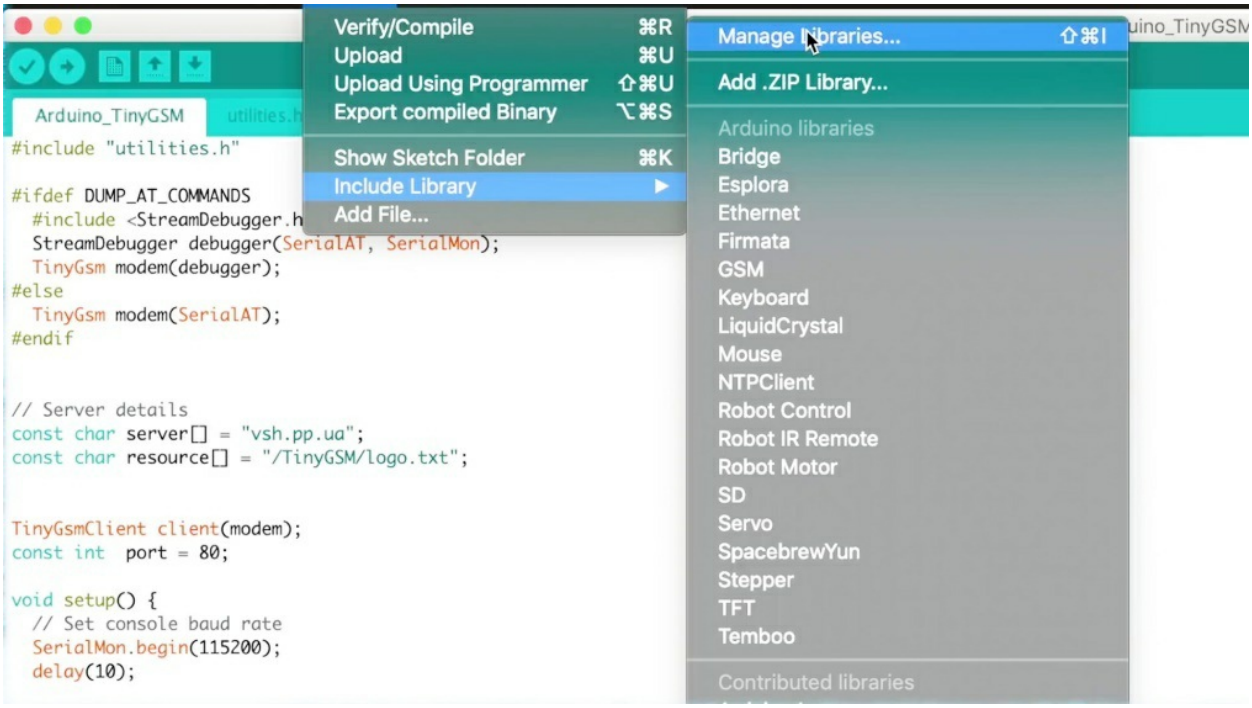
That means you can expect the wifi, Bluetooth, connectivity, or dual cord to extend a 32-bit processor, a capacity that spins, and everything which you expect from an ESPN three, two board. But what makes this model stands out? Is it DSM, GPRS, connectivity? Yes. This model has built-in SIM 800

LGS and DP artists. That means other than wifi and Bluetooth. Now we have, we're connected with the option for communicating between devices. We can communicate via SMS, via phone calls, and GPRS, internet connection. So now no need to worry about the and its range limitation. Now make your internet-connected projects be controlled from anywhere in the world without any rights. So I think this configuration is enough to make anyone fall in love with this model. Now let the street jump through the Arduino ID and let's see, how do you.



So now, before jumping into any court, I expect that everyone has ESP to the award packages already installed on the Arduino IDE. If not, then follow this Project, which will guide you on how to install it in August. Going ahead. Now, this mortar doesn't have any particular library for using it rather than we can use the tiny DSM library for using all the functions of the c-MET 800 modules.

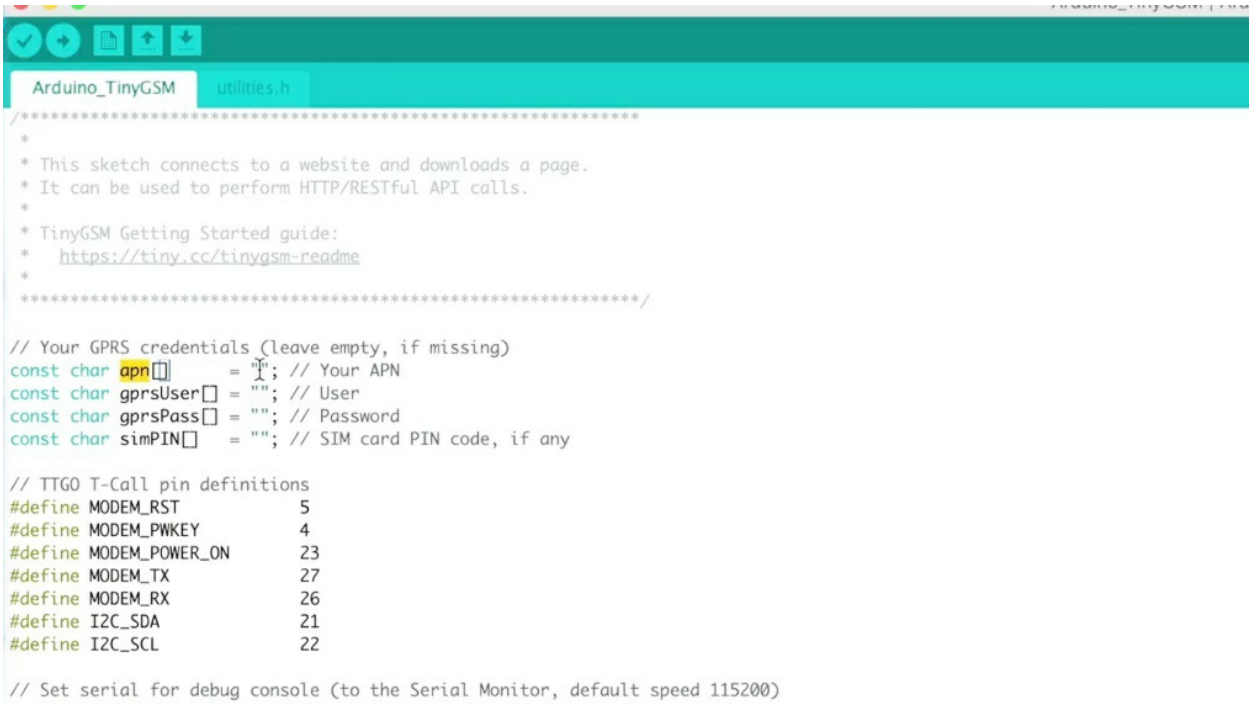
Still use the district, download this example, which we'll make to get started with this module a bit. There's a link for that in the description after downloading just an open example called tiny GSM. Now, before using this score mixture, you have a tiny DSM library already installed.



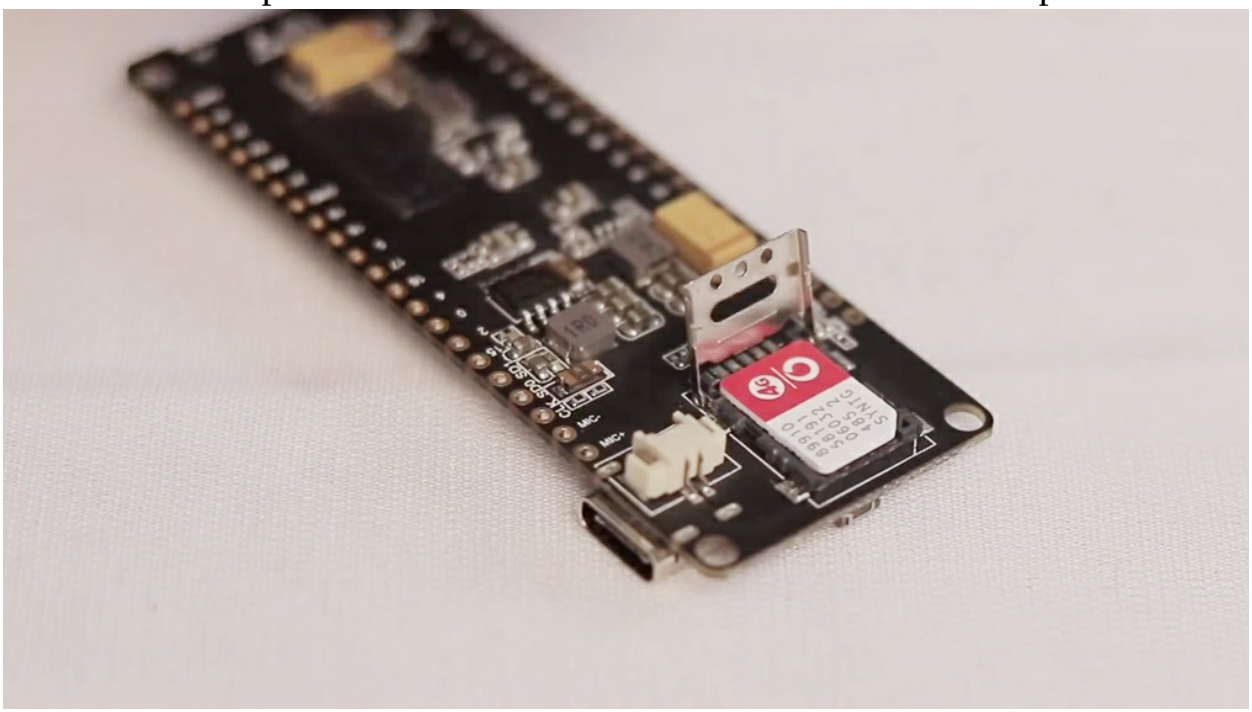
If not, then just go to the sketch include a library. That doesn't matter. There

are libraries here.

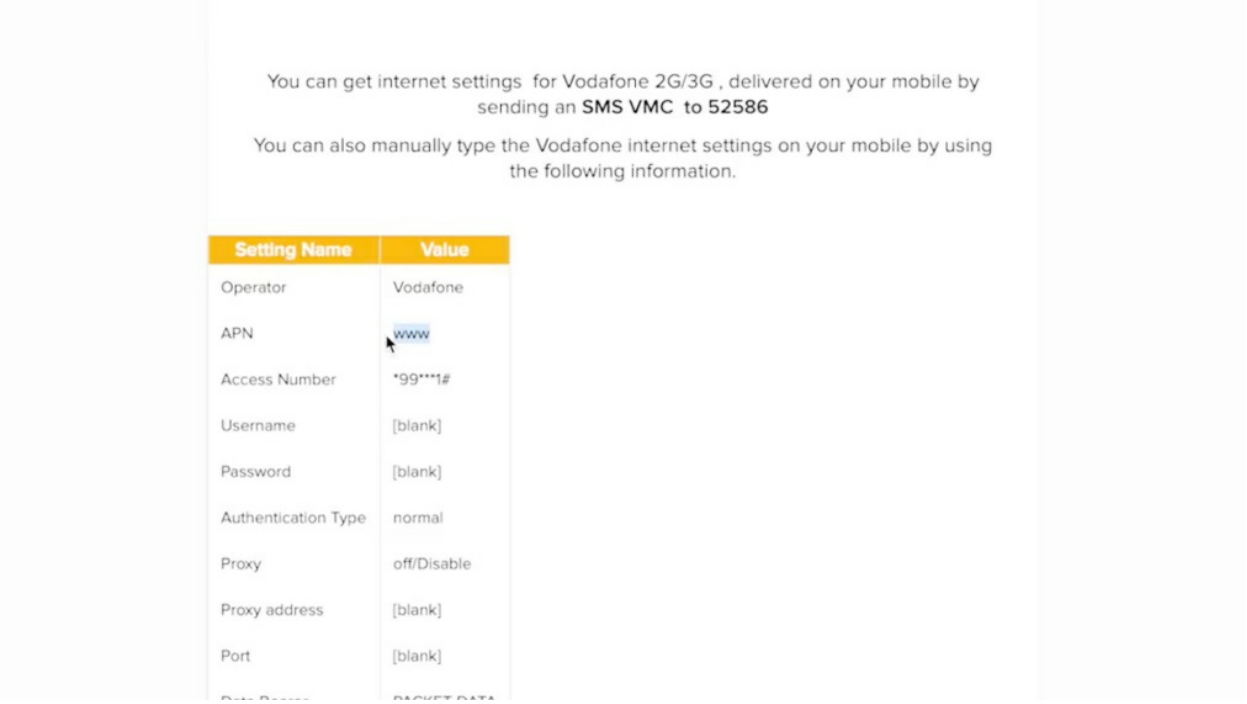
Click install. That's it. Now jumping into court. Then this court is only an example code, which can fetch the data from this website, from this. Okay. So before uploading the core, first, we need to provide the APN. That is the access point need.



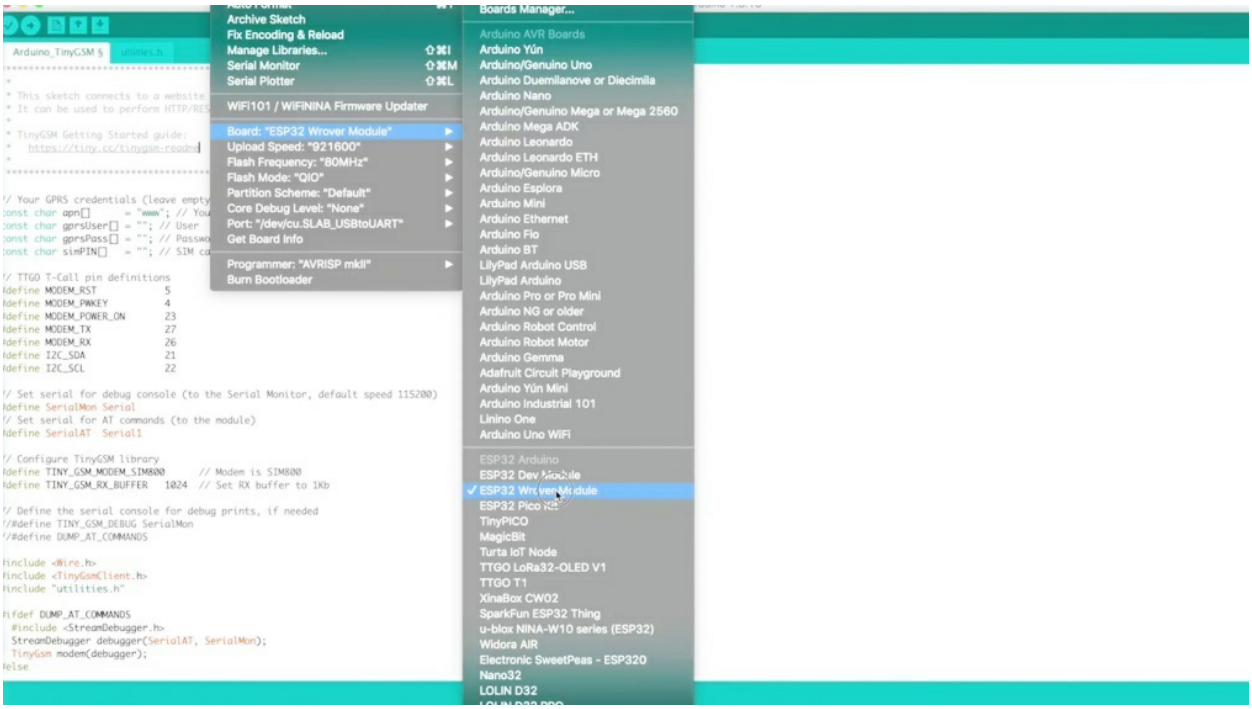
Now the access point name is different for the different network providers.



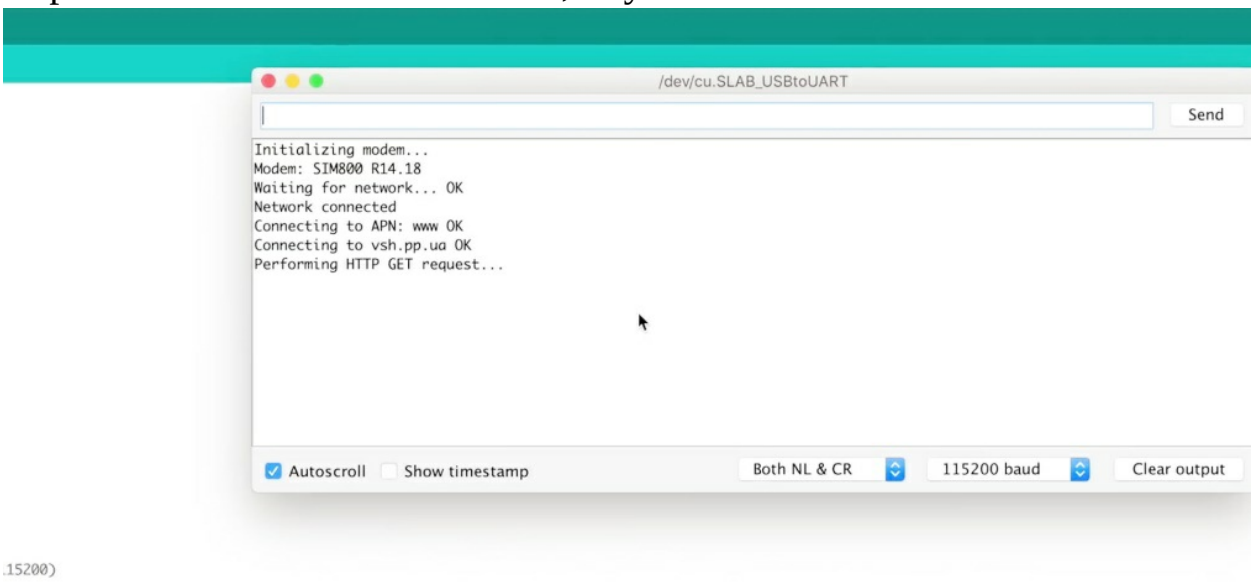
So currently I'm using this word, have one SIM card with an active data plan and the access point name for what a phone is www. I don't know the access point. Name of your Net-a-Porter. Nobody does Google it.



You will get it. Now one thing here, which I need to tell you is this module is a two G module. That means this module will only support SIM card, which has two D connectivity options. For example, if you are using a GeoSyn guard, this module won't support you because you have only Fuji banks, so don't have to the bank. So it won't be able to communicate with this module rather go for some other network provider. In my case, I'm using this, which has a two G. Okay, so make sure you use a SIM guard from the network provider who has two D banks.

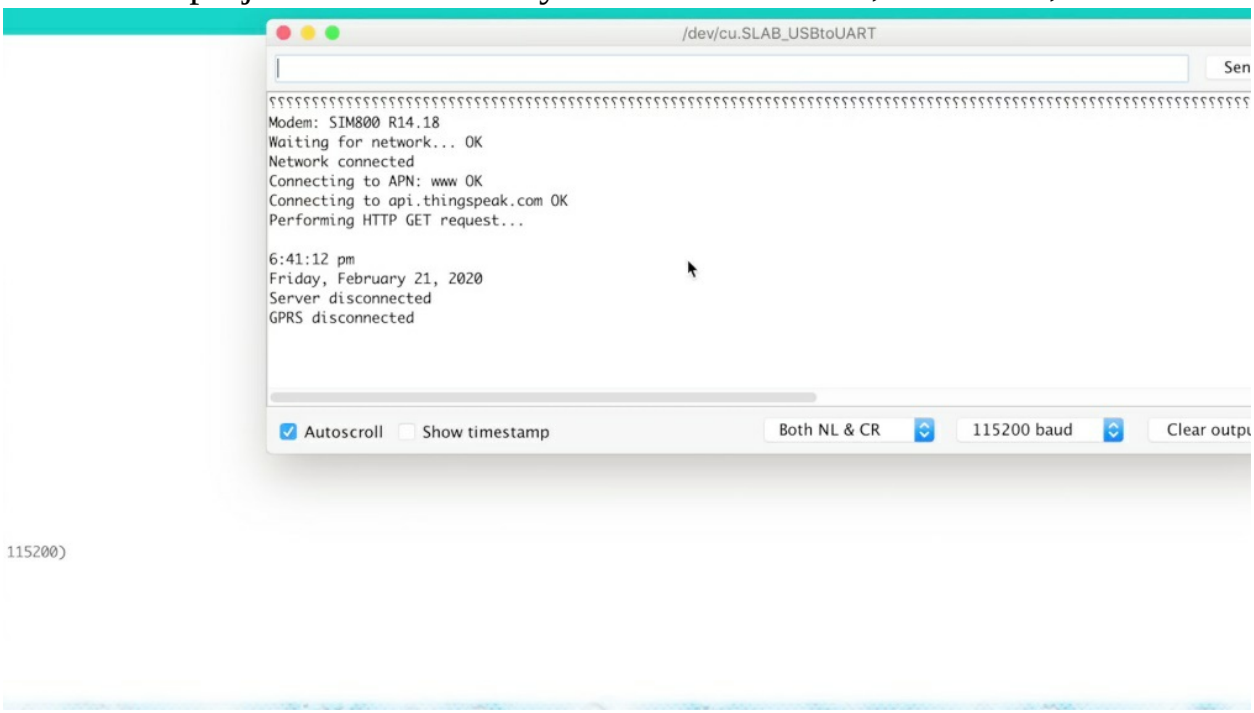


That being said I'll straight away upload this score by selecting the board as ESP three Dover module or cause of the court is successfully uploaded. Now let us open the seat on the monitor. As you can see, I successfully got the response on the website with SAIS, tiny GSM.

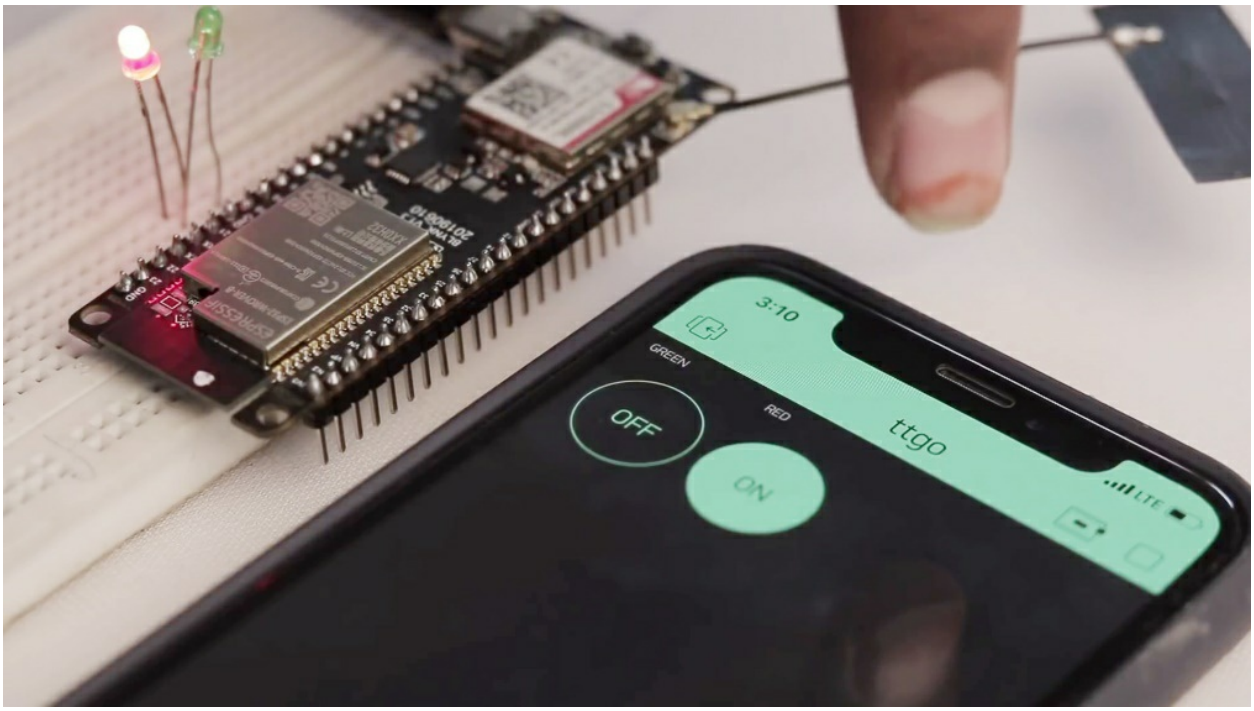


With this, I can easily conclude that the SIM card has an active internet connection and it got successfully connected with the server. So this was a

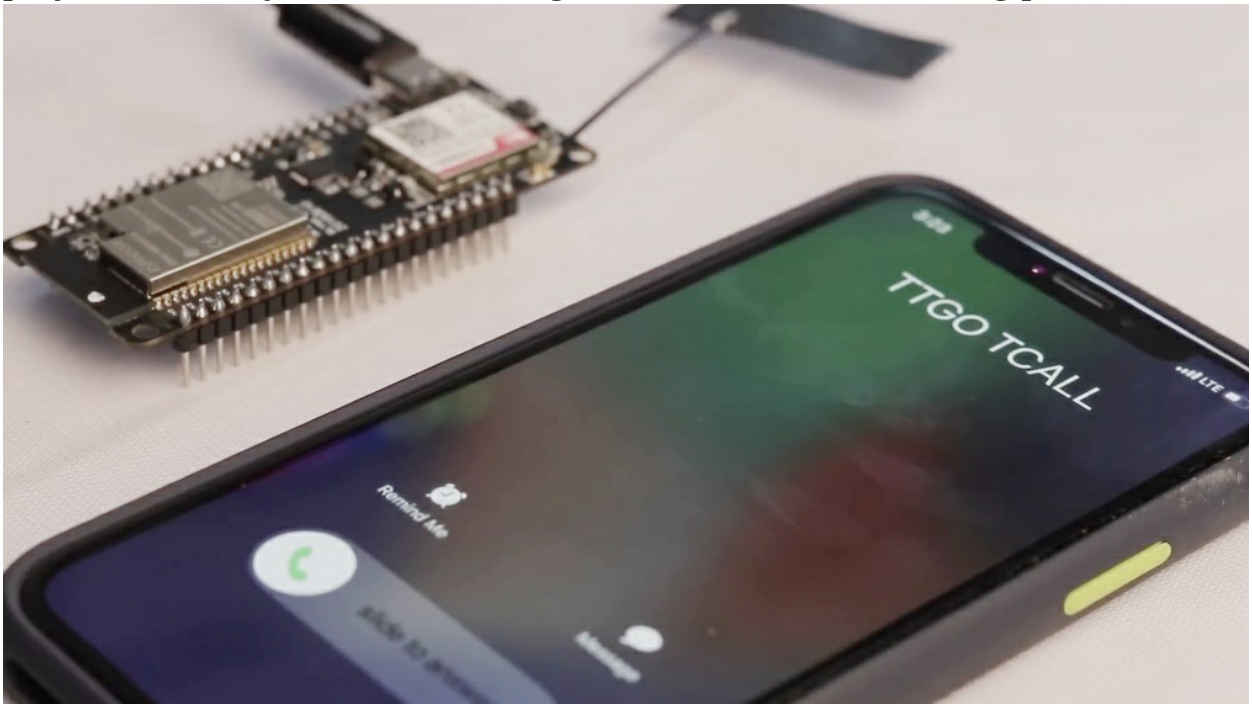
simple example. Now talking about the projects, which you can make using it. There are many projects, uh, you can make with it, but I had experimented some other projects and I will let you know what I. So, first of all,



I made a project called real-time and date in which I'm just vetting the time and date of India from one website by using the API. So with this simple project, we can make a world clog. We can make an internet-connected, uh, weather station without having any limitation of the wifi router. When you heard this morning also works with the blink IoT platform. That means now you can make the blink project will be controlled without having any vitality. Y



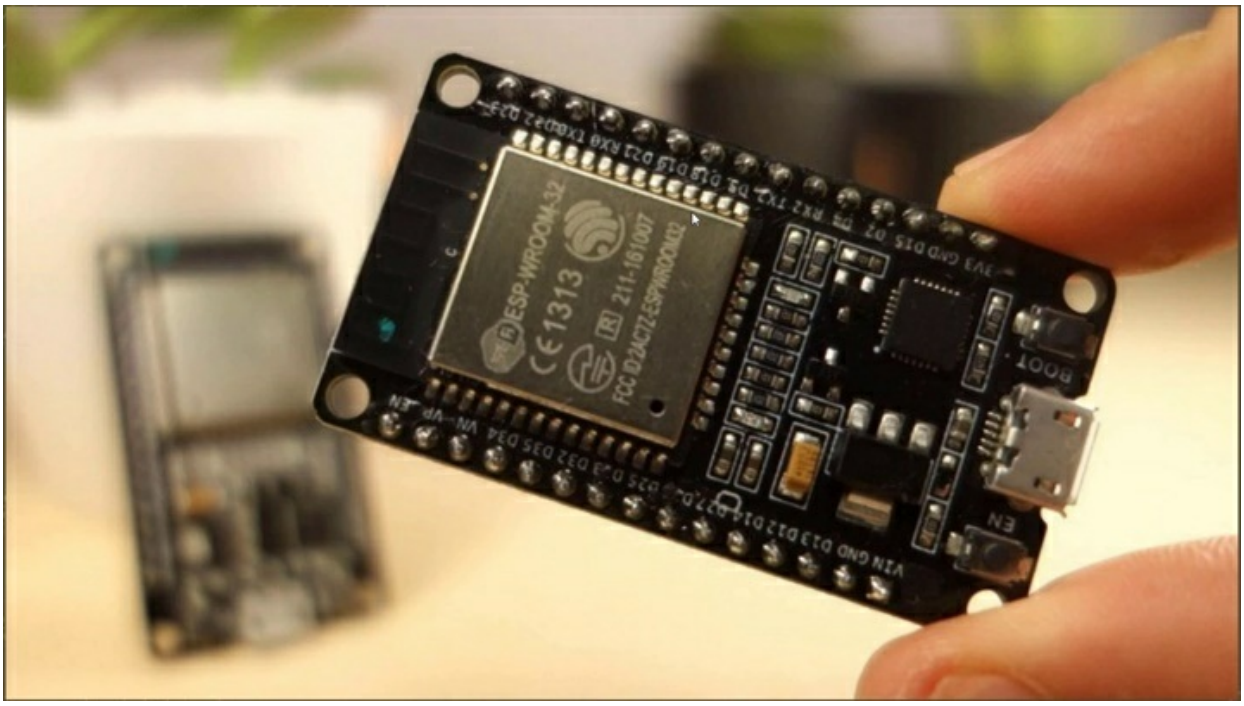
ou gets in the Project, as you can see, I'm controlling the LEDs connected with this module, with the help of the blink on my food. And here, the data is transferred through GPS, no wifi, no Bluetooth. This is an amazing way to communicate with that IoT device. When you added, we can also make a project like, uh, you know, sending SMS to the device, making phone calls.



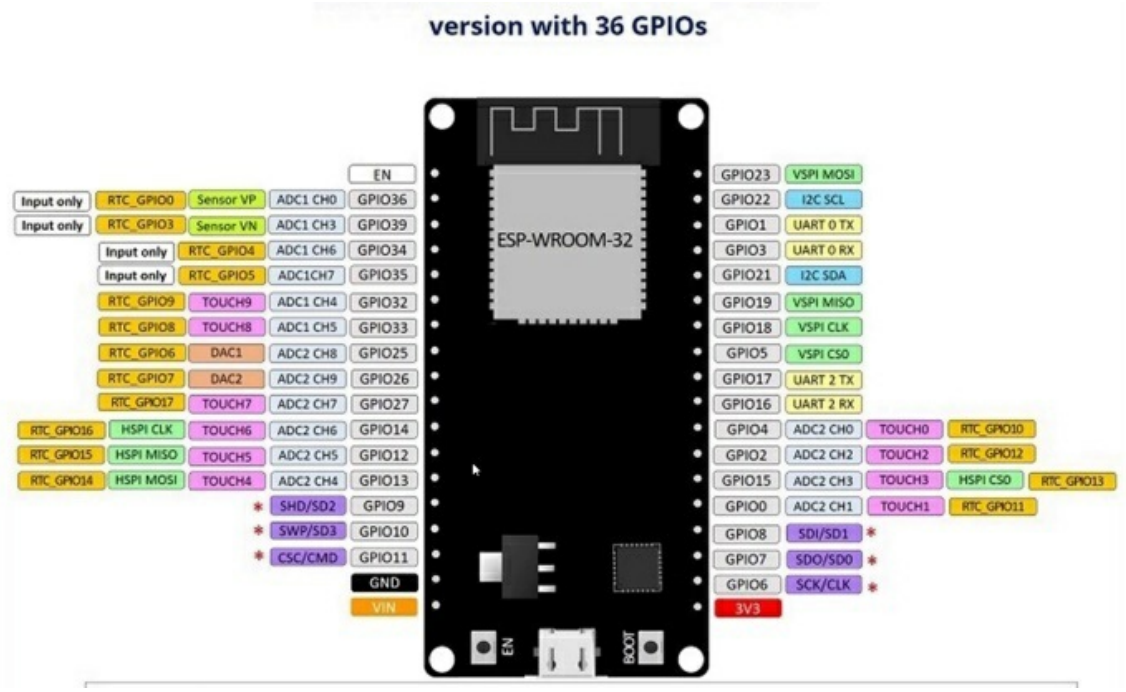
And there are tons of other projects which are possible with this, uh, model.

Do comment about the project, which you want me to make a detailed Project. And I was Charlotte. So, yeah, this was all about getting started with this module.

ESP 32 PINOUT V1 DOIT



You can see, it's very similar to the Arduino Nano, but it has a Wi-Fi built and now they Espey first two per Furlong's include 18 analogs to digital converter or ADC channel, where you can receive analog signals and these signals can be converted to digital internally. It also has three S. P. I interfaces for Çehre communication and three your art interfaces for S. R. O. communication and two eyes-to-see interfaces for Seattle communication. So these lists say eight modules or eight pens can be used to allow cell communication with multiple devices that support S. P. I Eye to see or Isaac Mercy and Your Art. It also has 16 P. W. Arm outward channels which help produce an analog output from the E. S. P pens. It also has two digital to another convertors and two eye-to-US interfaces. It also has then capacitive syncing general-purpose input-output. I provide more data and more details about each of these pens in the resources lecture. But since we have a lot of them, we don't want to get caught in the details.



Now, what we need to know is that pin out itself. As you can see, this is how the board blocks. This is the US B board, and you are going to hook up your Jospeh connector here and the other side will be connected to a computer. As you can see, these are the pens g p I o means around purpose and what output. Now that is more than one job burbles input to output. As you can see here, and each of these pens has more than one function. As you can see, usually Bend's comes with my name. H. S, P I and ADC and digital converter Anjar burbles and put out so you can use it as input-output, pain or to receive the analog signal or for S. P. I or for all to see. So the choice is yours. Now, as you can see, these pins, all are no for easy access so that you can easily know which PIN is connected to which. Now, additionally, there are pins with specific features that make them suitable or not for a specific project. The following demonstration shows you some of these pins. And I'm going to talk about each of these pins in detail and if they can be used as input-output. Now, the pins are highlighted in green here. I'll show you a table to summarize this information.

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	

Again, the pins highlighted in green are OK to use the ones highlighted in yellow are OK tools, but you need to pay attention because they may have unexpected behavior mainly at both times, while the pins highlighted in red are not recommended to use as input or output. Now, the general purpose and what output PIN zero is, as you can see. OK to use, but you need to pay extra attention because it may have an expected behavior at full time. So it can be used as pull-up input or as output. It outputs PWI signal output. No one can be used as t expen force here. Communication or output when it debugs output PIN two is OK to be used as input or output. And usually, it is connected to unboarded led. So you can use it to test a code or to test our Basch function because you don't have to connect extra components at all that we have built and led. PIN three is okay to use as input, but you can't use it as output. It's a high output, so it will read one output. Benham's Binz, numbers four and five, are OK to use as input or output, and PIN five also outputs P. W. AM signal output pens from six to 11 are connected to the integrated S. P. I Flash. So you can't use them as input or output. But No. 12 is okay to use. But food will fail if Bould High. So it's OK to use as input. But you need to pay extra attention to this not. And if you are connecting it as output, it's OK. You don't have any problems. Pens from thirteen to sixteen are OK to be used as input or output without paying extra attention. Now as

you can see. Same for pens from 17 to 33, while 34, 35, 36, and 39 are input-only pens. You can you can't use them as output. Now, our examples are in our practice tests. We are going to use two, which have the built-in lid on board, built and led, and using that on board built and the lid will make it easier for us to test out or to try different things. If we are making Auberge with a baton to control it via Internet, now that's it for the pin out. Now I will add extra information as articles to this section of the project. So let you know more information about the job payables and what outward pens.

17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

But for me, what I need you to know at this point is that we have pens that can be used easily without any extra attention as input-output, which are these spin's two, four, five, 13 to 16, and 17 to 33. And if we want to input only pens, we can use 34, 35, 36, or 39. Now, if we need extra features if we need B. W., ADC, or DHC, if we need a combusts or enabled bin or BW humpin, we can go on and check this schematic. And from these pens, you can see that in our case, SHA Purple sPIN36 can be used as ADC or do some converter. As you can see, and it has in but the only state, you can use this schematic printed out to refer to it whenever you need to do something. Same for here. You can see from this image that PIN25 can be used as a digital to analog converter and PIN26 can also be used digital to and converter or just

have a converter. So depending on what you need or what's your end goal, you are going to check this schematic and make sure that you choose the bin that fits your need. So if you are going to use a pen as output, you can't use PIN34 because as you can see here, it's only input, only pen. And if you want to use, let's say, an analog-digital converter, you can't use this pen. PIN17 because it only supports Syria. Communication and input-output. Regular input-output or digital input-output. So before using any pen. Take a minute or two to make sure that it supports what you are going to do. And the same sorts or the stuff that you are going to connected to without they are analog or digital. They are input like pushbutton or output like LED. You need to connect the element to the right pen before start coding to avoid having problems in the future. Once you start testing your code. If you have an inquest soon or if you have a project that you don't exactly know which pins might fit for that project.

ESP32 VS ARDUINO SERVO MOTOR CONTROL

You've got to understand if you're programming, Arduino. Um, that'd be things like digital, right? Digital read analog, right? The analog read will tell those four functions down. There is so, so much. And I would also add to that list, controlling DC motors, using a motor driver, and controlling servos. Once you have those six things down, you can add them together in so many different ways. Now there are tons of awesome tutorials out there for controlling servos with Arduino. But today I wanted to focus on the differences between controlling servos with Arduino versus controlling servos with the ESP 32. Now, this can seem a little intimidating at first and there are a few key differences. Once you understand those differences, there's so much you can do with them. Let's take a look.

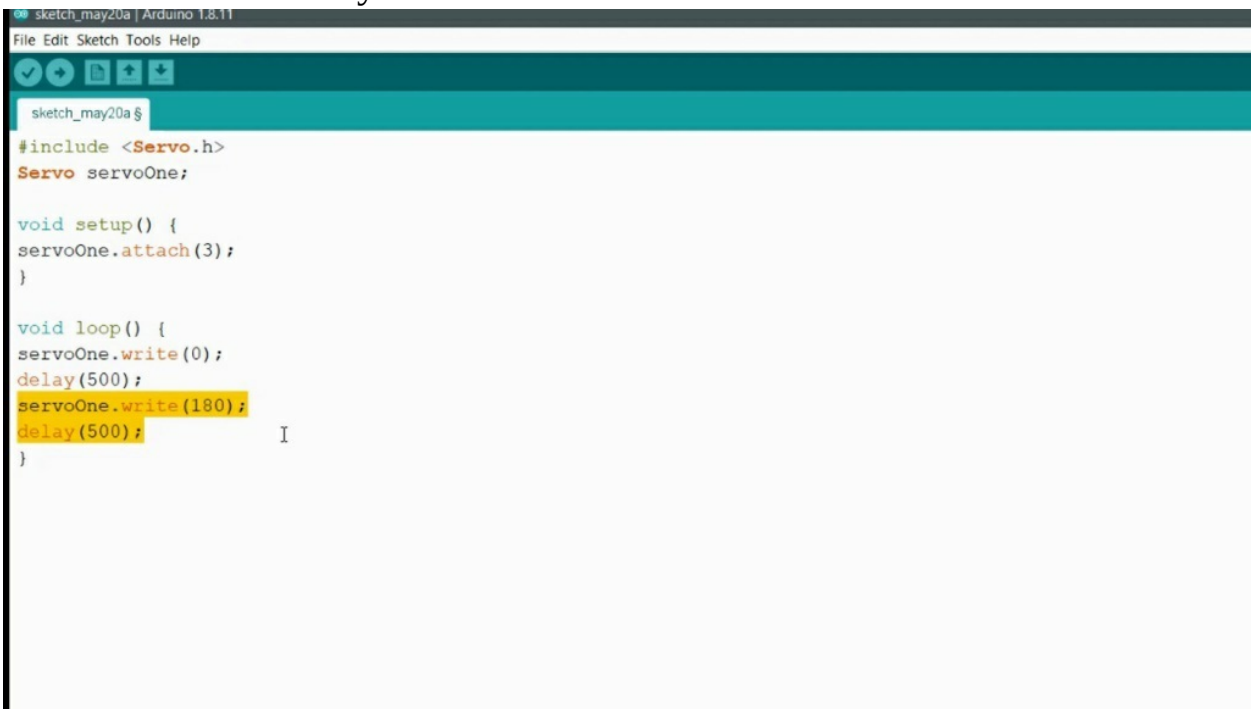
So here we have in our, do we know nano and ESP? Now I love this kind of Arduino nano here. So this is the RF nano. It has the NRF 24 L zero one radio built-in, which is awesome for creating the wireless index. Now the

ESP 30 SU also there's a ton of wireless communication built-in, right. We have Bluetooth, BLE. We have wifi. Of course, we didn't have ESP. Now. I love love, love of that stuff. So. The Arduino has been around for a lot longer. And so when you think of the number of projects and all of the amazing things that people have done with Arduino over the years, it's, it's just extraordinary, you know? Um, I think they deserve so much credit in helping shape this community, of makers and young engineers. Um, and there are Arduinos that have more PWM pins, than the UNO or the nano. You know, in this case, we're gonna be talking about the Arduino nano. When, you know, you can use something like a 25 60, if you wanted to, if you needed to use Arduino and you needed more PWM pins, but that being said, if you are using an Arduino UNO or an Arduino nano in this case, you're stuck to six PWM pins. That's 3, 5, 6, 9, 10, and 11. So let's see, I don't think this, this model even labels them for you. They're labeled with a scullery mark, um, on a UNO. So we have pins. D five D six D nine, D 10, and D 11. Um, now one downside though, is that the radio here uses SPI communication. And so the, uh, the pins are hardwired 13 through nine. And so in reality, we just have three, five. Six and that's pretty much it. And so if you only have three PWM pins and say, you want to use some of them to drive a motor driver, you don't have as many options. And so if you need a lot of pins, you could use an I squared C motor servo driver. Those are awesome. Um, if, if you want it to, but if you want to do this all in one chip, you may want to think about using an ESB 30. So when ESP 32 comes in a lot of different shapes and sizes, I love this model right here. Um, with the external antenna, I use these for all of our crazy robots and it's a ton of fun. Um, and so obviously the ESP 32 has way more GPIO pins. And so what's so cool is except for a few input-only pins you can use almost any of these pins. As a PWM pin for servo control. Now the biggest difference, and I think this contributable up sometimes is that there are not specifically PWM pins, there are PWM channels. And so even though there are 30 plus GPIO pins on here, um, you can't use three. Different servers if you want him to, but you can use 16. And so what you do is you set up your PWM channels and then you attach pins to that channel. And I know that can seem kind of confusing, but

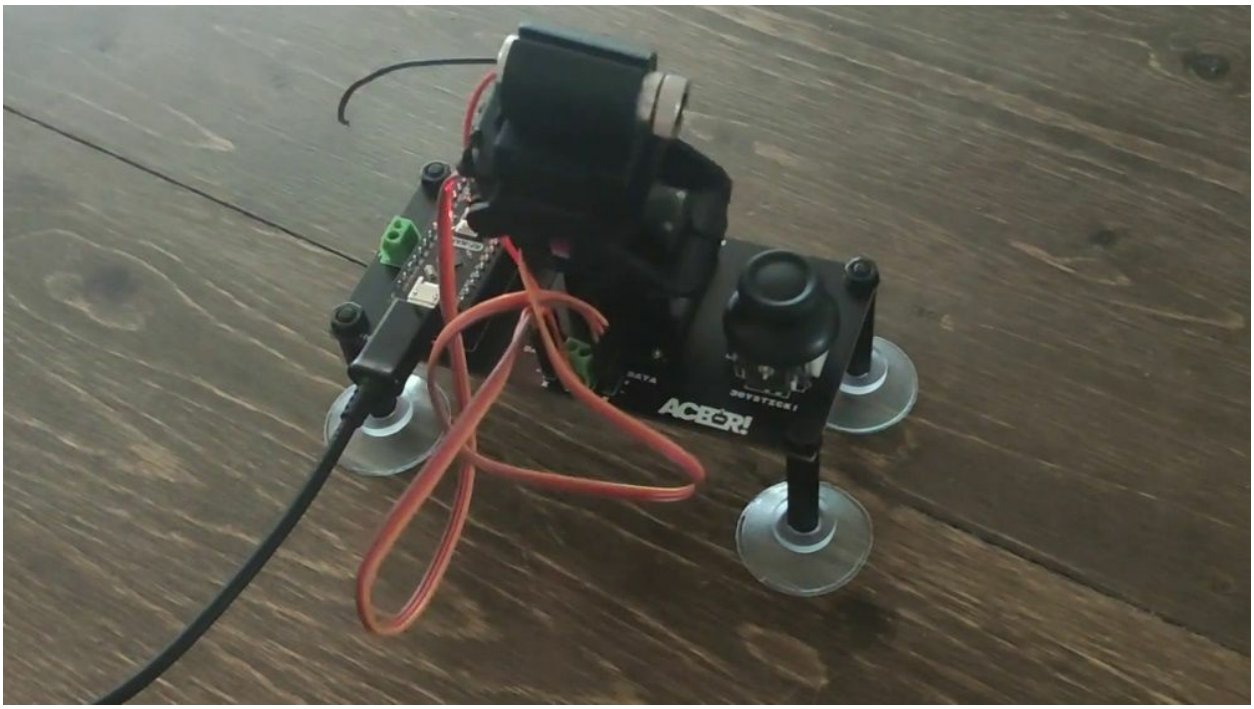
that's why I'm making this project. So I want to make sure that by the end of this project, this difference is clear. And you have no problems controlling, either a servo with an Arduino nano or an ESP 32. Let's go ahead and take a look at the Arduino. So to demonstrate how to program a survey using an article, you know, I thought it would bust out the ACPR robotic laser cannon kit.

This kit is a ton of fun. We just finished a little bit of a redesign on this guy. Um, we added the, uh, uh, screw terminal here for the laser and a screw block terminal for some external power. Uh, but this is a simple example here of how we can take some joystick data and we can correlate that to the servo position. Now for us to get a better understanding of how this all works, let's go and look at the code so we can break things down and talk about how to program. Okay. so just like the ESP 32, there is more than one way to program a servo using the Arduino. Uh, some people like to get in there and control the PWM directly using the, uh, digital right microseconds command. Um, but the most popular way by far is to use the servo library. Now the server library for Arduino has been around almost as long as Arduino has, and it is useful for controlling servers. Um, it makes it straightforward. And so, um, the first thing we need to do is to add the servo library. Um, so the servile library is built-in whenever you download or do we know? So oftentimes whenever you're using a crazier library, you have to download it and install it. Pre-installed in Arduino. So you don't have to add any other files. All you have to do is call it. Okay. So then we need to name our servo. We're going to be super boring here and call it servo one. All right. Now we need to attach our servo to a pin. In this case, we're going to start with pin three. So we're going to start with the name dot attach. And then the PIN, pin three. So I have Serbo one dot attached to pin three. Now we mentioned this a little bit ago, but if you want to control a servo using the Arduino, you've got to use pins 3, 5, 6, 9, 10, or 11. Um, and in some cases you need, uh, those higher-level pins, the, uh, you know, 11, 12, 13 for our communication. Um, luckily if you're using, I squared C some, like I squared C sensor, that's using. A four and five. So that gets out of the way, but there are those options. So 3, 5, 6, 9, 10, and 11. Okay. The first thing we're going to do, it's going to be

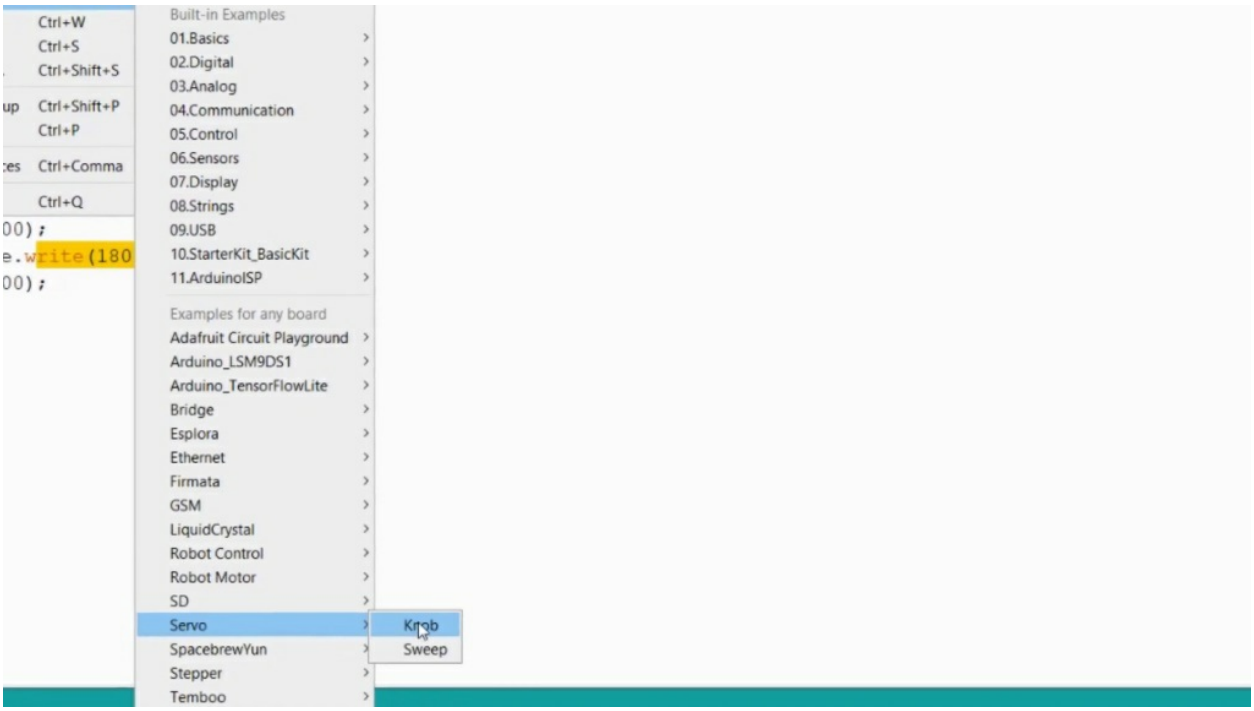
crazy. All we're going to do is we're going to tell it to go to zero degrees. Oh my gosh. So crazy. Just kidding. Okay. So we're telling this to go to zero degrees. That being said, if we send our servo to zero degrees and it was already at zero. There's no way for us to tell if it's working. And so we're going to get a little bit crazier here. We're going to go ahead and we'll say for half a second, then we want you to go to 180 degrees. Stay there for half a second. And so now this is super simple. All we've done is we. Imported the serval library we've named our servo. We've told it to attach to pin three. We've told it to go to zero degrees for half a second, then go to 180 degrees for half a second. Okay.



So let's go ahead and we will plug this guy in, upload this code and see the result. Sure enough, our servo is moving from zero to 180 degrees, albeit a little bit frantically here.

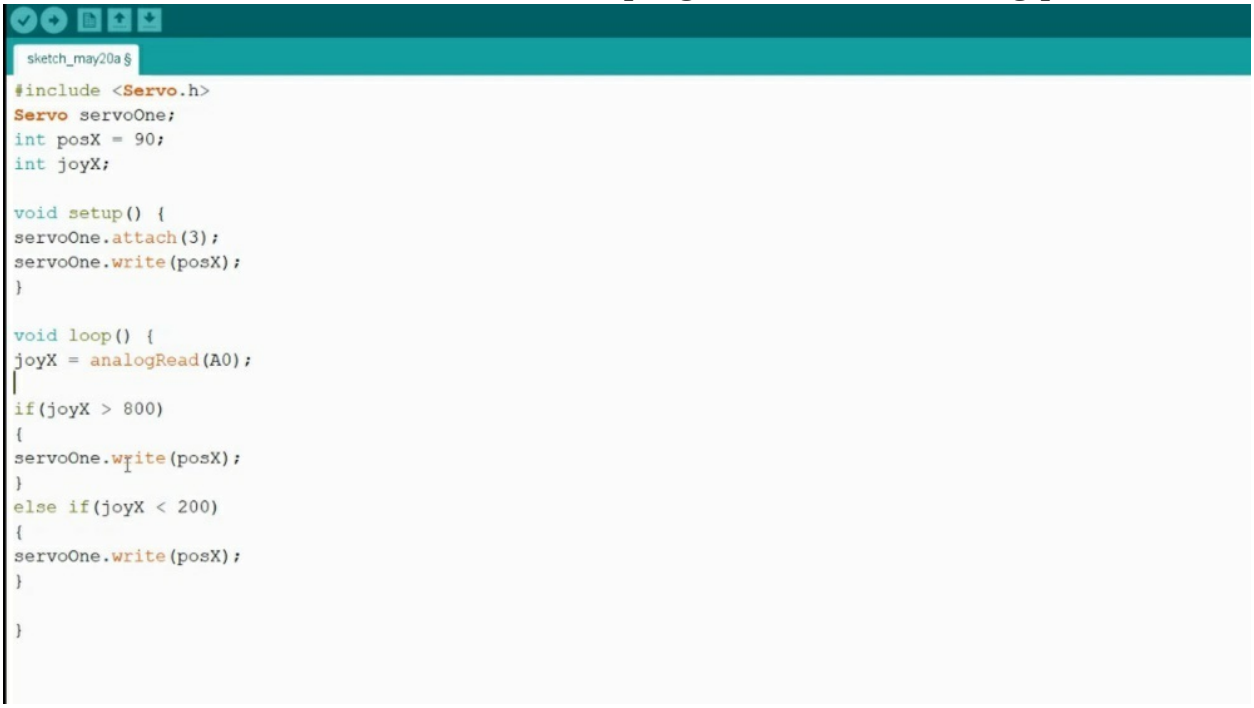


So whenever we're talking about servo control, we often want way more control than just going to zero and 180 degrees. And so let's go ahead and take a look back at the code to see if we can't add a little bit more. All right. So getting that basic server movement is. Right. So we, uh, name our server. We tell the server which pen to go to. We say what degrees that we want that servo to be at and for how long we've got it taken care of that being said, you often want way more controlled. You know, your survey going to a specific degree. And so there are a few different ways to do that. Um, there are some great built-in examples, um, in the servo library.



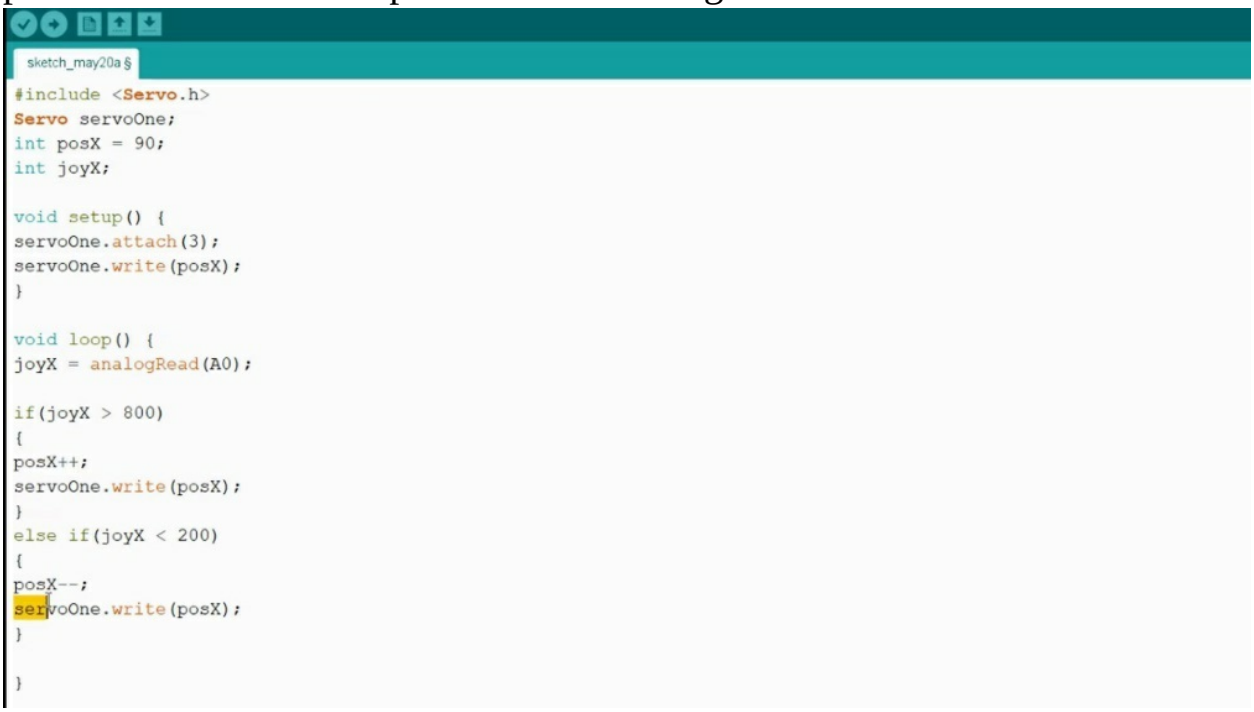
So if you go over to examples of servo there's knob and sweep, so sweep uses a for-loop to, uh, move a servo backward and forwards over and over again. And that's great if you're unfamiliar with four loops and how those function, we also have the knob example and that takes a potential amateur reading, right? It's from a, uh, like a potential amateur for a guitar or, um, smaller pen geometers for breadboards or even like a joystick and, uh, correlates that movement with the, uh, the map command. And that's also really, really awesome. But in this example, I'm going to show you a super-easy way to get a lot more control using position variables. We're using the ACPR robotic way you can. And as an example, um, I'm going to go ahead and share the whole schematic for this project on the website. So there's always a tutorial, um, on our website that goes along with these projects, we can go in a little bit further depth and show pictures and stuff like that. So if you're curious about what this actual schematic looks like, you want to build this on a breadboard, um, head over to the website. Cause we've got that information there, but in this case, I need a few more variables. I'm going to go ahead and make a position X variable, and a joystick X variable. You can name these variables wherever you want. Right. Does not matter. Um, let's see, you know I'm going to go ahead and give this a value. I'm going to say position X

equals 90. Cause we want our, um, our servo to start at 90 degrees. Okay. So in the void setup, we can do something pretty tricky here. Um, we can make sure that our servo always starts at 90 degrees. So in the void setup, right? Cause that runs once at the start of the program, we're still using pin three.

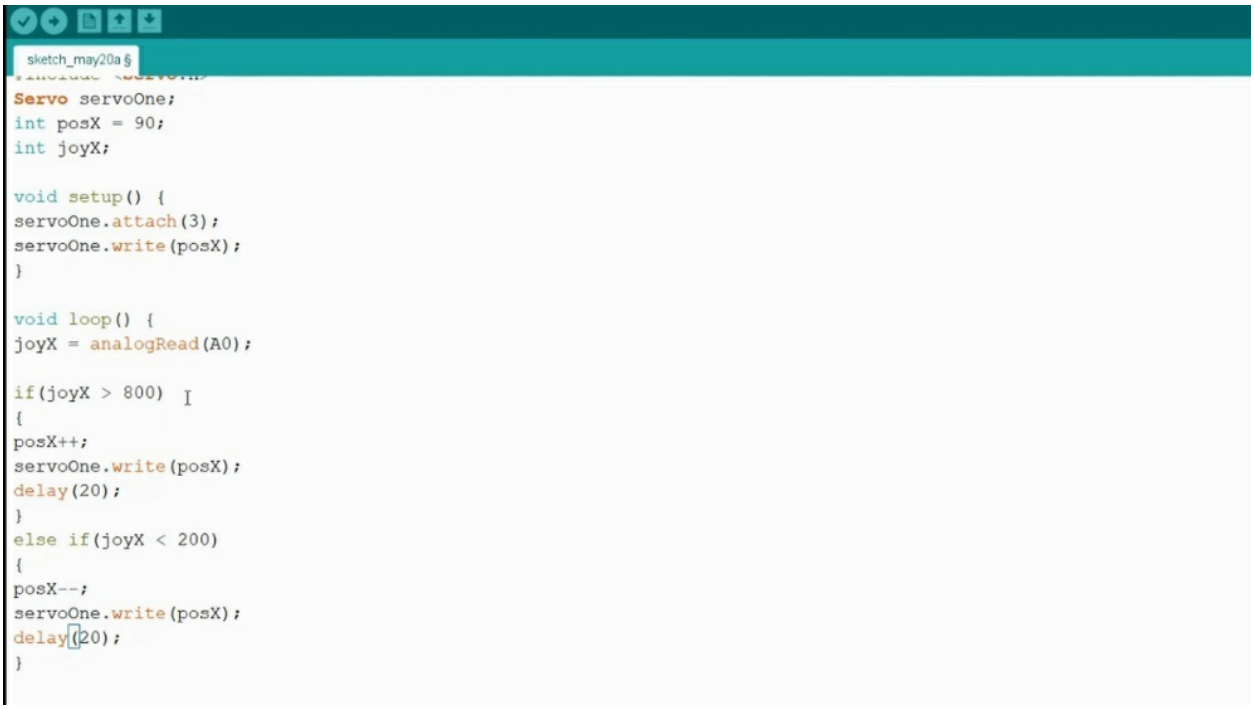


We say the very first thing I want you to do is to go to the position. Okay. Now our void loop here is going to look a lot different. So we created two new variables. We had positioned X and we had joy X. And so in this case, I'm going to first give joy X a value. So I'm going to say joy X equals analog, read a zero, and easier with the pin that I have hardwired in the circuit. But again, check out the schematic, um, that we have on the website. If you want to see this, uh, where this would be wired up on an Arduino if you're using a breadboard. Okay. So how joysticks work it's cool. So all the joystick is doing is it's taking you. Voltage in most cases five volts. And as you move the joystick forward and backward left and right, all you're doing is changing the voltage, potentially amateurs. So in reality, there's, uh, two potential and there one for X and one for Y or one for forward and four backward, one for left and right. And as you move the joystick, you're changing the voltage, making it go up or down. And so what's cool is if we use the analog read command, The pin that we've plugged into our joystick. We can read that

change in voltage as a numeric value. And if we were to just run this, you'd see that you see the numbers zero to 10 23, whenever you're in the serial monitor. And so we can use those numbers that we get from the joystick to move our server. Okay. So I have a value. I say joy X is whatever voltage I read on the pin. Okay, so we're going to need a conditional statement here. So I'm going to say if that value goes greater than 800, then I want my servo to go to position X. Else if joy X is less than 200, I want my servo to go to position X. Now if we uploaded this code right.

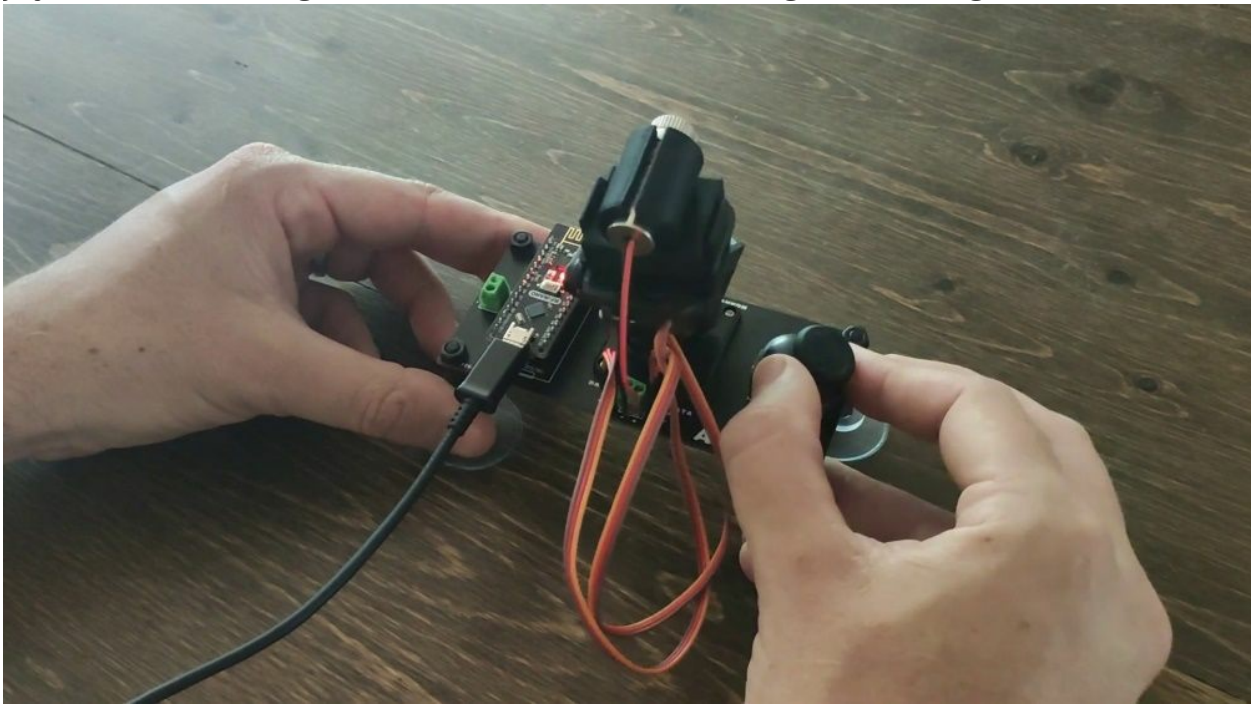


Absolutely nothing would happen because we've said go to the same position under two different conditions. But what we want to do is we want to move our servo based on. That joystick position. And so, um, I said a few minutes ago that as you move the joystick, if you were to open up the Sierra monitor, you'd see the numbers, uh, zero to 10 23. So, uh, 10, 10 23 would be if you held the, uh, joystick in one extreme and zero in the other.



And so if I make a condition based on the number 800. Then I know that servo must be, I'm sorry that joystick must be close to one of the extremes because you get right about 500 or so in the middle. And so if I move the joystick one extreme, move my servo again with 500 being in the middle. If that value goes less than 200, I want you to also do something with the servo, except, in this case, it's not going to move until we change the value. Of that position variable. And so in this case, I'm going to use a little bit of shorthand using plus and minus. Now, um, if you're familiar with programming in general, this is going to be, you know, super familiar to you. But if you've seen this for the first time, it can be kind of confusing. And so all plus is, is shorthand for add. So add one to the value of position X minus, it's just the same. It's take away one or subtract one. And so now what's going to happen is that as I move my joystick forward and backward, I'm going to increase and decrease the value of my position variable. And as that value changes, my Servo's will respond in kind. Now, in reality, this is going to move way too fast. Um, so I usually put a little delay in there. Let's say we'll do 20 more seconds. Um, one thing that's cool too is if you're, if you're writing this code here and you want your servers to move faster, all you need to do is make this number right. Okay. So I am now, uh, reading some voltage here and I am

moving my servo forward and backward based on that there. Okay. So here's, we're going to do were to go ahead and upload this code again, and then we'll, we'll see if we can tell them. and so sure enough, as we moved the joystick left and right, we have our server moving left and right.



So for more information on the actual laser candidate, self hetero website, we have tons of stuff. I'm all about how to program that on there. I'm also, have some more projects here on our YouTube channel, but before we move on to how to do this same thing with ESP 32, I want to show you one more thing. Uh, this isn't exactly necessary, but it's always a really good idea. Let's head back over the code quickly. And so, as we saw in that last example, and we had nice steady control, right? So we had, uh, the, uh, voltage, we were reading from the joystick. We moved the servo backward and forth. And that worked well, but there's one more thing that I always recommend adding, um, which is not necessary, but I think it's a really good idea. So we talked about how all this part of the code here is doing is making the position variable smaller or large. Now, here's the thing. If you move this joystick forward, right position, a variable here is getting smaller. So it starts at 90 and that goes to 90, 80, 70, 60, 50, all the way down. But if you keep holding that joystick down, that number is going to.

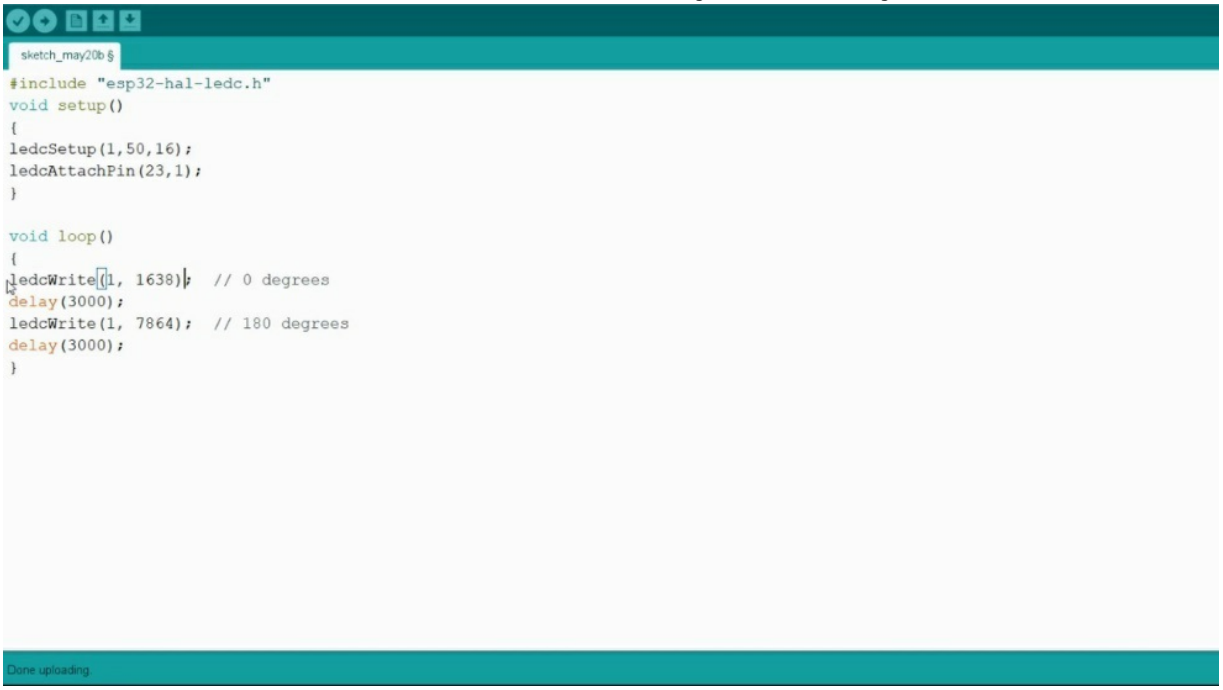
Counting. And so it counts to negative. And so obviously, a servo can't go to negative 60 degrees, right. And so the same thing can happen on the positive side. So, uh, the joystick variable goes forward. That variable just gets bigger and bigger and bigger and bigger. And, [00:17:00] uh, it'll go past 180. So it goes to 180, 1 9200, and the server just won't move. And so what can happen is if you keep holding the joystick down, it'll seem like the servo is not responding anymore, but the problem is your range is totally out of whack. And so luckily there's a really easy solution. So we're going to add in a second condition. All of the ad it is. I've said, Hey, if the joystick value goes greater than 800 and the position value is greater than zero, then take away from the position value variable. So this will make sure that we never go past zero. Now down here, I'm going to add in the same condition with a little bit of a twist. If position X is less than 180 because the servo can't move past 180.

And so I've said that, Hey, so long as position X is still less than 180. Go ahead and add to the value of position X. If you were to upload this here, you would see that you could hold the joystick down for as long as you want. And a servo would never, um, try to go out of its normal range. Um, and again, that's not necessary, but I thought I would add that just in case, uh, people

were curious about the best possible way to use a position variable to control the servo. Okay. So we've got the Arduino side down, let's head back over to, our ESP 32. So we talk a little bit more about that. All right.

So to demonstrate how this works for the ESP 32, I thought I would bust out our brand new ESP 32 robotic arm driver board. Um, I am so pumped about this new, uh, We have this as a kit available up in the shop. And, um, right now I have one of our me arms plugged in and we've got this in the shop too. Um, now it is a great little robotic arm, um, but it's gonna be perfect for showing us the differences between using servos with the Arduino and servos with the, uh, ESP 30 SU because if you look closely, we're using the same servos for this robotic arm. Um, but one thing that's cool about this, uh, driver board here is we also have the whole patterns. You can kind of see it, right. Uh, for the much larger all-metal robotic arm that we use on the Voyager and explore a robot. So the code that you've learned for this small robotic arm will be the same. If you wanted to use this for the Voyager or Explorer or put that other larger robotic arm on there. So let's go ahead and take a look at the code so that, we can talk about the differences between how you program the servo for the Arduino versus the ESP. All right. Let's take a close look at what it takes to do the same thing that we just did with the Arduino, with the

ESP 32. So there are a couple of differences here. Um, but after we break down all the code, you'll see that in reality, we're using different numbers, but there's only one extra. And that's this guy right here, but before we get to that, let's talk about the Arduino file. So I'm sorry, the library file.



So, um, we do need a library file just like before. Um, now the name is not as simple as servo dot H. Um, that being said, you know, it's, it's not that much harder to type in this other library file. Um, so one thing I'll mention though, is there is an ESP 32 servo library. Um, it's an unofficial library that somebody else created that is really. Um, and it functions the same way as the Arduino library. Uh, that being said, you don't have quite as much control over your servo and the PWM channel that you're creating. And so I highly recommend that you do, uh, try this method first. Um, go download that library, see how it works. There's also some great example code, um, in that library, that's not the method we're going to show right here. I want to show you guys how I would recommend doing it. Um, and it's a little bit more complicated, but you have a lot more time. So, all you have to do here is grab this library file. Um, and this is, um, included in the, uh, ESP 32 board files. So whenever you load the ESP 32 into Arduino, you have this library built-in. Um, and so this led to seeing that's referring to the led channel and that's kinda misleading. All we're doing is creating a PWM. Um, but that's what

that C references there. Okay. So this first line of code right here, this is where it starts to be a little bit weird here. Um, so we have led C setup. And so what this is doing, and this is setting up our PWM channels. So there's that C for the channel. And so what this means is we're using GPI. I'm sorry. We're doing, um, we're setting up channel one. We're going to use a frequency of 15. And we're going to use 16-bit resolution. Now, if you don't know what the frequency and the resolution have to do with all of this, don't worry about that. I highly recommend research can further cause it is super, super interesting. The most important thing to take away from this is that this is the line of code that we need to set up the channel. 50 Hertz, the frequency that we're gonna be using for our PWM signal and a 16-bit resolution. So whenever you hear, uh, using a command in Arduino, like analog, right to 55, you're using eight-bit resolution. And so that's one of the cool things about, um, the ESB 32 is it allows you to use a higher resolution for your PWM signals. Now for smaller servers, that's not necessary. You can do some crazy stuff with that. And it's cool. So this is a line of code that we need to set up our PWM channel. Number one. Okay. Now, this next line of code is similar to what we had in the void setup on our last bit of code. We are attaching a pin to that PWM channel. So. We are, uh, attaching PIN 23 to channel one, the channel that we just set up. So instead of just name dot attach, we have led DC or led channel attached pin 23. So it is written differently, but saying pretty much the same thing. Okay. Now, if we want our servo to move, what we need to do is we need to tell that channel. How many microseconds we want our PWM signal to be now in this case, 1638 is not as simple as just saying zero. I get that. That being said once, you know, this year was the same thing as 1638 for this servo, you can repeat that over and over and over again. So in this case, I'm telling channel one to create a PWM signal at 1,630. Microseconds now down here, I have channel one sending out 7,864 microseconds PWM signal. And so that is the same thing as saying 180 degrees. Now for these a microservice. They go a little bit further than 180 degrees. Um, and that's one of the cool things about, um, the using, using this method here is that you have way more control. So if your server goes over 180 degrees, you have that range to use that full range of motion. So let's say

you want it to go a little bit smaller or your circle is a little bit different. You can play around with these numbers a little bit. So you could like, I don't know, uh, 7,400 as you are. Or you could use the numbers that we use here. Right? So you could also try instead of like 1600 here, try 2000 and see what happens. So the numbers that I'm putting in here are directly correlational to the number of degrees. It just happens that this number right here was perfect for us for zero degrees. And this number right here was perfect for us for 180 degrees. And more than likely those numbers are going to work pretty well for you guys too. [00:25:00] No matter what server you using. Now, that being said, you may have changed them a little bit, but, um, It's super, super easy to play around with. So the code isn't that much longer in reality, we only have one extra line of code and that's setting up that channel. Let's go ahead and we'll upload this code here to our ESP 32 and see what happens once we have our code uploaded. Sure enough, we see that the servo is moving from roughly zero to 180 degrees, just like we had before. So in this case, as I said, feel free to mess with those numbers a little bit. So say you want it to go a little bit further in not be quite so, uh, so wide, neither direction. You can do that, but you see that the differences between the two codes are not all that crazy. So just like before, um, we don't want to have our server moving by itself. We want to have control over it. And so what we're going to do. Instead of using an onboard joystick since this guy doesn't have one, we're going to use the PS three controller, cause that's a great way to get joystick data wirelessly and use that data to control, uh, our ESP 32 and our servers. So I have a whole guide on doing exactly this in another project. Um, and so if you're interested in that, go through that project. It's awesome. Um, but I'm also going to show that here, just so you guys see those differences in that code, let's head back over toward Reno so we can see that. All right. So let's take a look at what it takes to get that same level of control that we had before, uh, except this time wirelessly cause we just PS three controller.

Um, but using that position variable to move our servo backward and forwards. So now this example code does have a lot of extra stuff just for that PSD controller. I'm not going to go into too much detail on that. Again, I have other more in-depth tutorials on our YouTube channel and our website that go into that. The most important thing I want you guys to see is using those position variables. So if you look here at our main conditional statements, I'm getting the PS4 controller data, I'm using one of the joystick data, um, cannibalism RX, the right joystick on the X-axis. And I have different numbers here in terms of, uh, what I received from the, uh, controller. So in this case, instead of 800 and 200 being the joystick values, I'm using negative five and five. And, instead of 180 and zero, I'm using 8,000 and 1500. Cause if you remember our minimums and maximums, one was around 1600, the other was around 7,800. This is doing the same thing. So this is just more of a generalization. I'm doing the same thing. So if my right joystick goes less than negative five, therefore moving in one direction, add 25 degrees to the value of position one. If my choice, Dick moves in the opposite direction, then take away. 25 from that. Now, in this case, we're not using plus and minus, because we're dealing with a way a wider set of numbers, you know, from 8,000 to 1500. And so if you used just plus or

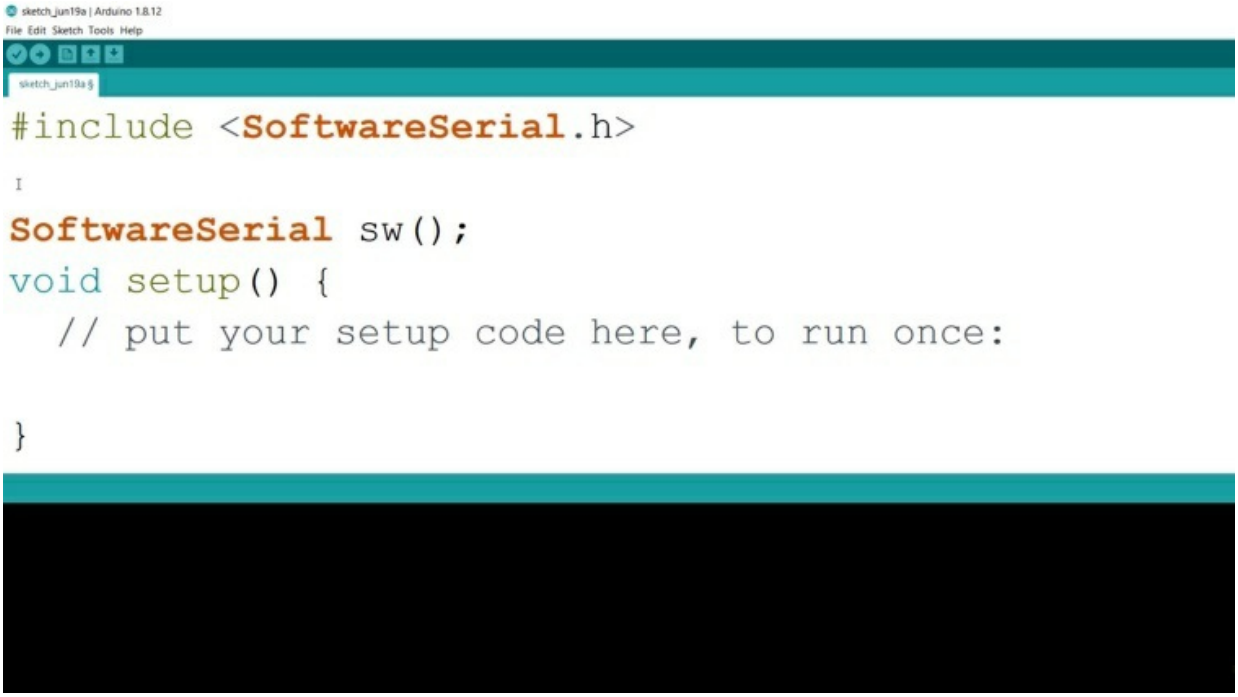
minus your service would move incredibly slow. So the only, the only difference here is, uh, is the numbers we're using. But the concept is the same. We have our same channel setup, right? We're still using pin 23. Um, in this case, our position variable, we're using 5,000 cause that's right around in the middle. Um, if you remember in our last code for the Arduino, we used 90 because 90 degrees is exactly in between zero and 180. Other than that, with the addition of the PS three controller code, this is the same. Um, so let's go ahead and upload this here, and then we'll see what else.

So sure enough, if we turn on our robotic arm here and we hold down that middle button to pair the PS three controller, once it is paired, you'll see that I have that same control. And so we've made a couple of steps from that, uh, first Arduino code, right? So when you to Reno code, we use the onboard. To control servo backward and forwards. And so here we have a wireless joystick doing the same thing. Now, admittedly, the code is a little bit more complicated, but if you break it down, it's pretty simple. So I hope that was helpful, guys. Um, I'm planning on having a lot more content for this ESP 32 robotic arm driver board down the road, maybe even an ESP now controller, um, a wifi controller, a sketch who knows there's so, so, so many options

here, but I hope you enjoyed this tutorial all about, uh, the differences between controlling servers using ESP 32 and they are, do we know they're both awesome platforms and both great for creating robots for more tutorials, head over to our arts Toros page on our web. We also have tons of really fun stuff here on YouTube. So check it out. Have a great day guys.

ARDUINO CODING

Now let's go to our artillery, you know, on a ball so at it can send data through that serial communication protocol to our E. S. P board. Nothing will be fancy on our Arduino side. We will take samples from an analog-digital converter and send them over soft cereal to do this. Let's first open up. Are we not be? Now will take the census reading. And we will send it through the soft serial communication protocol. This creates a new code. No.



The first thing that we need to do is include the soft Syrian communication protocol, so include. Soft where serial takes. Now, the software Syrian. Will be conducted using two pens and let's choose pens, number two and three. As an example for the receiver. And that runs Miss Russum rookie. So there are X and T, X will be number two and three, four pins. Number two and three. And here we are not using the hardware in Syria. You have to make sure that you understand the difference between software, serial, and hardware serial. Now, inside the voice, it up. What we need to do is initialize the serial communications here that begin at a specific rate. Then we need to plan something to make sure that it's working. So brand new line. And let's right project. Or interfacing with Dwinell with E. S. P, first to.

Now, once you are done, you must start the software Syria using the object that we are defined here as W. So as W dot begins. And we have to assign a board rate. We will use that same board rate. Now, this Syrian is the one that you see when you click the serial monitor. And this is the software here. So this is the hardware serial. And this is the software Syria.

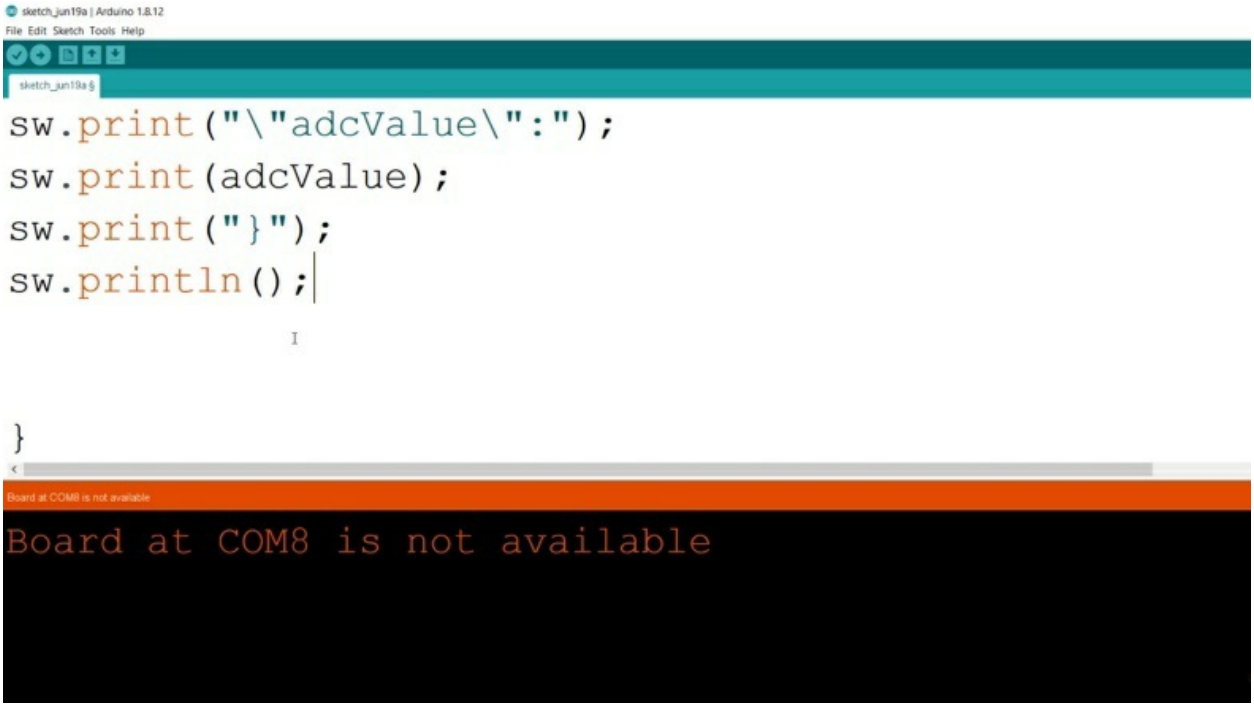
Now, let's go inside the void loop. We started the hardware, the software

Syrian. Now inside the void loop, we need to print something. So let's try to see the brand new line. Inside its lights and. Data to. PSP 32 bought. After that, we need to read the Sensor, and let's connected to a zero or zero. So ADC. The value will equal analog fields. And inside it, we can fly zero is zero now after this. We need to send that to the Syrian. So Syrian. Trent. And inside it, we must enter that in a specific format, so the format will usually be like this. We have to, right, OK, load it. We have to add to calibrate the seas and add slash here and between these two Gilbreath seas. We have to instill that sense sort of data sense or I. D.

So let's say that we have more than once and so here we can define an I. D. So. And since all I. D. equals, let's say 50. And this all C1 this is the first since all that while adding here. This would be more reasonable. Okay. Now, inside here, you have tried that in a specific format so that E. S. P can receive it and send it to the web. Now. You have to act exactly like me. Let's start by adding that curly parentheses after birth. Add a slash then add to. Text marks inside them, but I. The name of the variable, which is a sense of I. D. after that, is a slash lane between these two marks and double points. That's it. This is the first line.

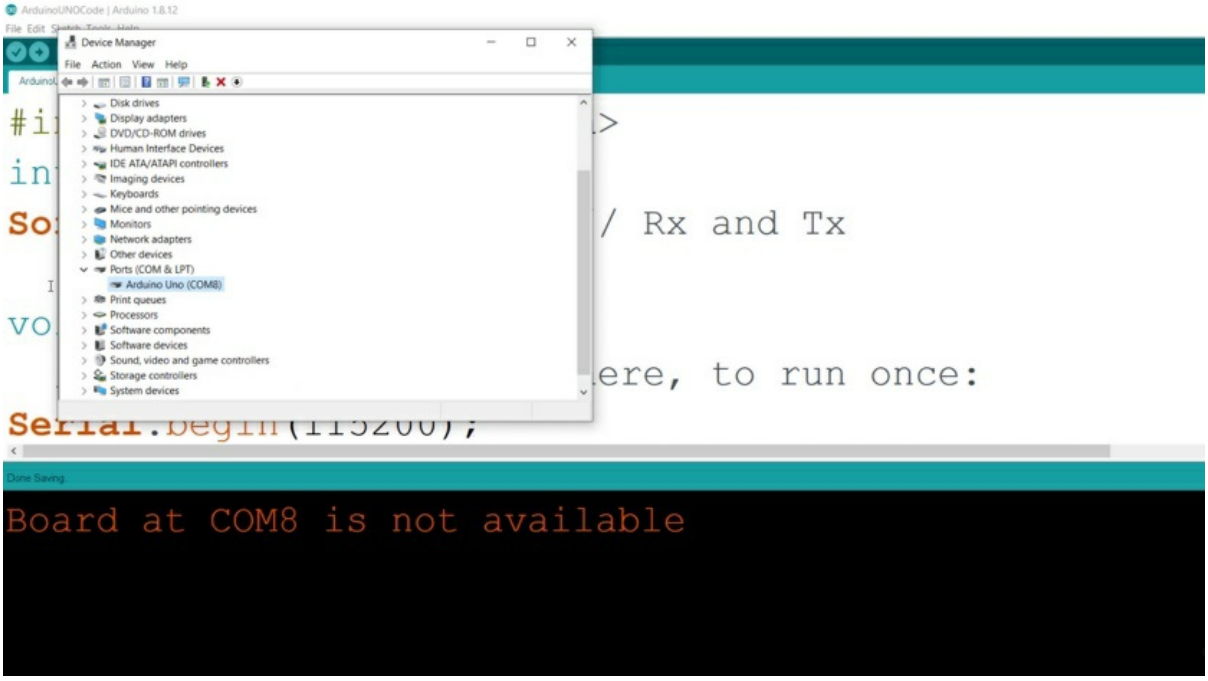
Now, the second line will send the. Since we already value. So we will have to paste this variable since or idea, this Waterhouses or I. D. the text or on the Syria monitor. And this will send its value after that. We must add a comma. Then we call out Syria. Brent. After the coma, we need to send the ADC value that we just received. So add slash here. Then after this at ADC. Value. CAYLUS. Again, Celia. So here you need to slash it. You need to add ADC. Value and club slash then. Between these two, you have to add double points. Then you need to plan the value. So serial the trend. Here you will plant this variable, which is the value of our sincerity. And we need to end this. So we have tried cereal print. Inside it, we need to add that Kerbel at the scene. As you can see, we started with the Colonel parentheses and we ended it with that closing tag for the killer. But he's now doing this. You must in turn your line, which means that you have been done and the data must be sent. So, Prince. And that's it.

This is the day that we are going to send to the serial or the hardware seller Moto. Now we need to send the same data to the software serial that we just created. And to do that. We just need to copy the very same code so we can copy all of this and based it all. We can light it again but save time. I will just copy. Discord and base to tears.

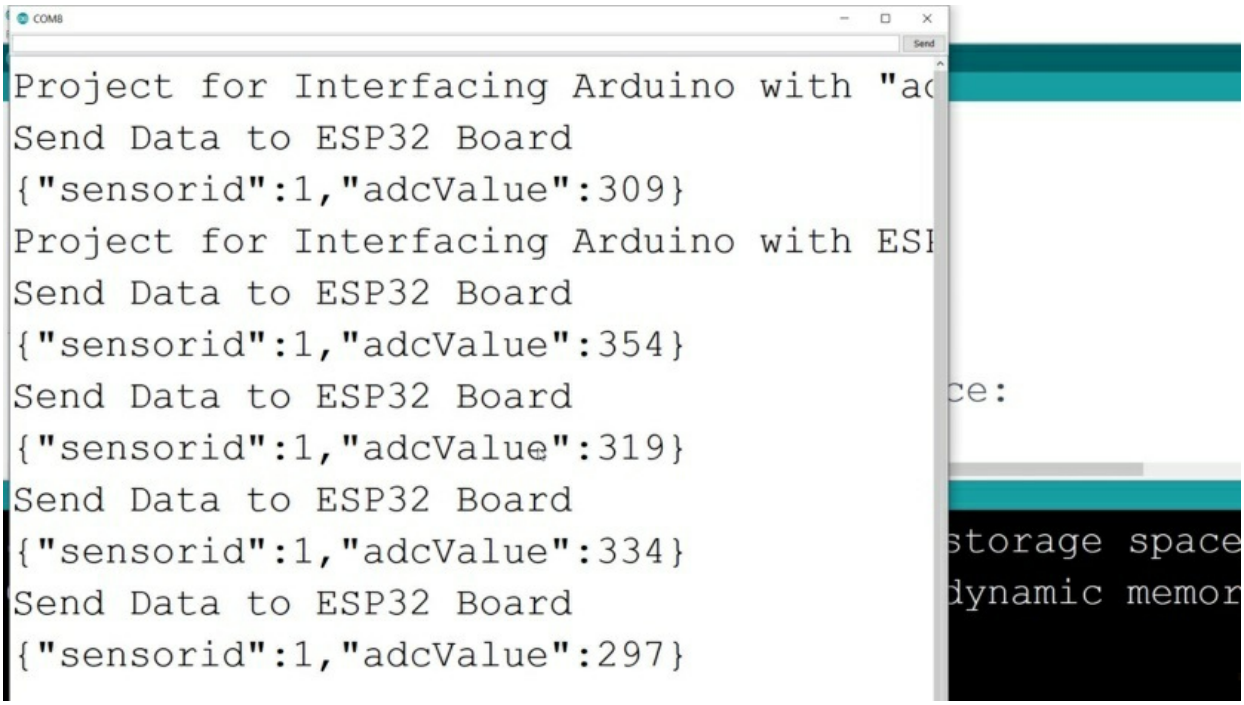


Now you have to change the serial with S. W. . It takes four, it uses the very

same delusion. So we will send the very same values. And in the end, we need to add a delay. Let's make it three thousand milliseconds or four thousand milliseconds. Now let's save our old. Let's name it out of Guido. All. Could click save. Now, what you need to do is simply upload this call to or are doing it online. So let me hook up my Arduino one. Now that we have the boats connected, go to the device manager.



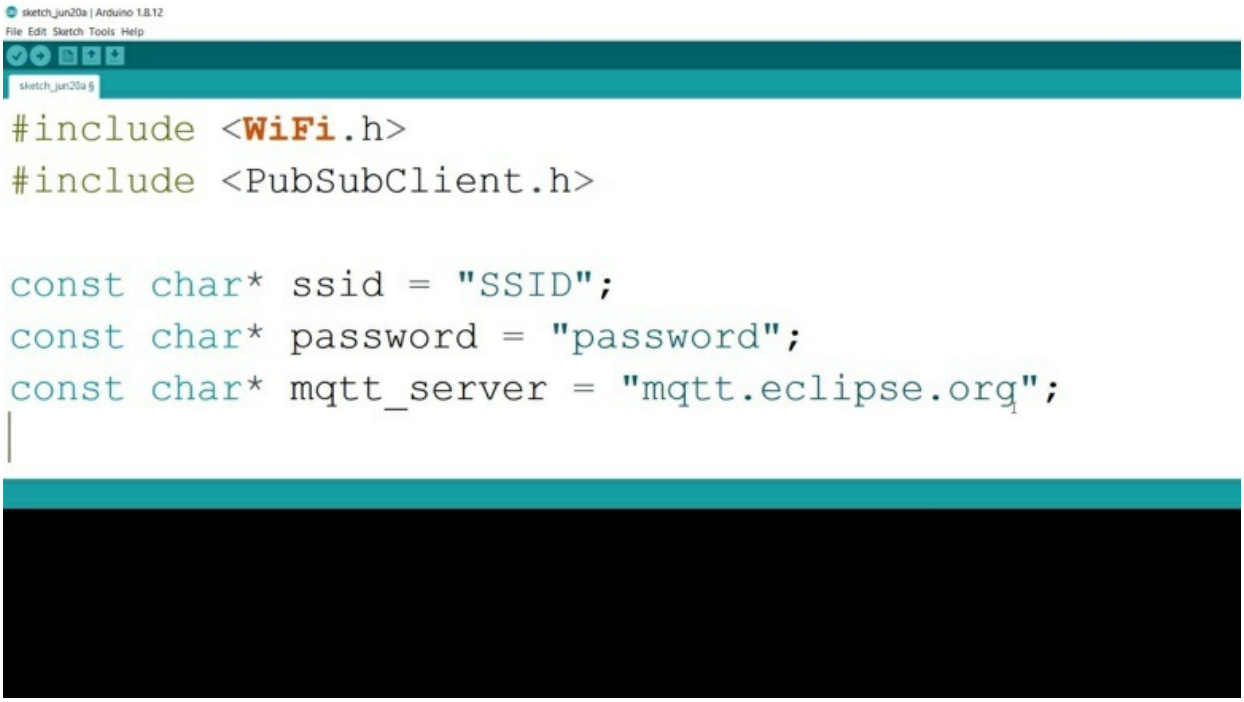
And from here, you can see the port on eight. So go to the tools. Make sure that the command is chosen and are doing all, then verify the code to make sure that doesn't have any Iran's. Now, a blog called. Now, once the call is uploaded, you can open up a serial monitor.



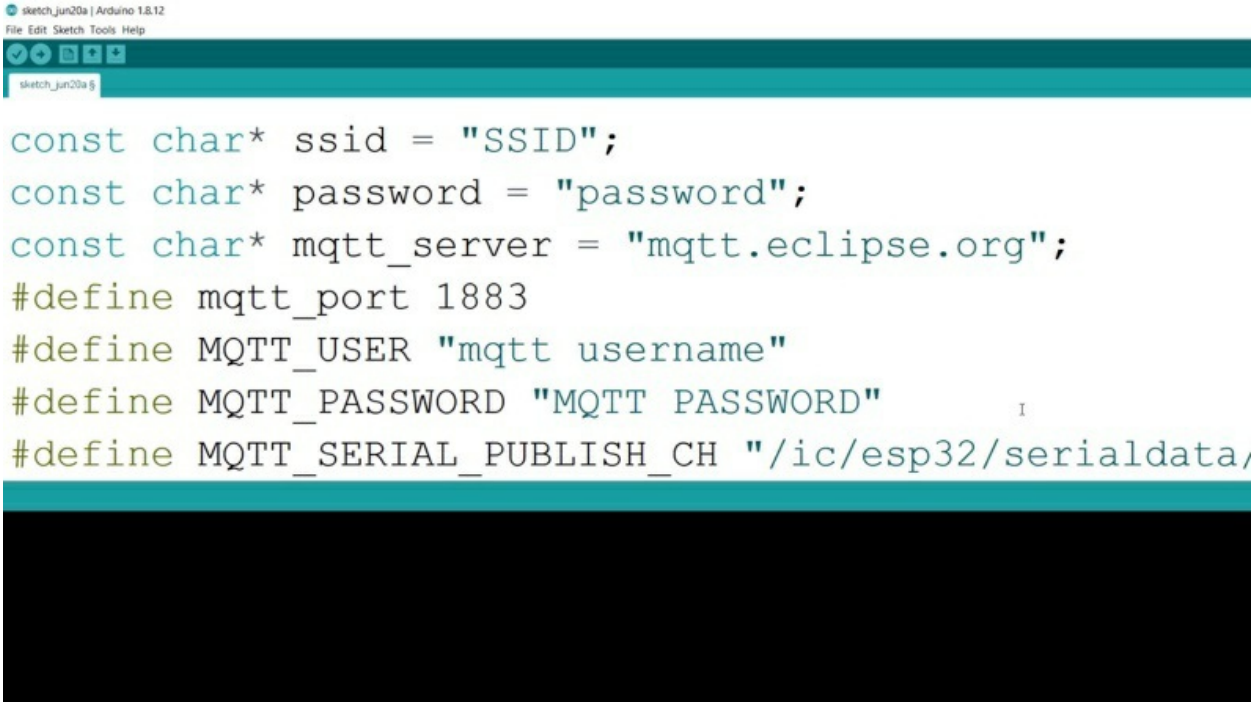
And see if this kind of data that's being sent, as you can see, seeing that E. S. P fit the tool and now a sense of ideas, one they do see value is three hundred nineteen. It's changing now. This data will be sent to the E. S. P board through the serial or the software protocol. And we are going to cover all of the E. S. P 32 codings. Now we are done with our coding. Now you can send any other value depending on your project. But this is it. If you see motors showing, this means that the SEAL communication is working correctly and you are sending data in the correct format, Carol. Parentheses and opening and closing. But since that idea is between two double quotations, they do see a value between two double quotations. And we have that sense or idea value and the same sort of value. And we have a comma here between them. So if you want to send other values, you have to add a comma and add that in the very same format. These two points, then the value. So we can send 100 values from ALGUIEN to E. S. P. Or you can even send Tinson sort of speeding. Depending on the sense of I. D. and the ADC value for each census. But here we are only sending two values since of I. D. and the sense of value.

ESP CODING PART1 DEFINE VARIABLES

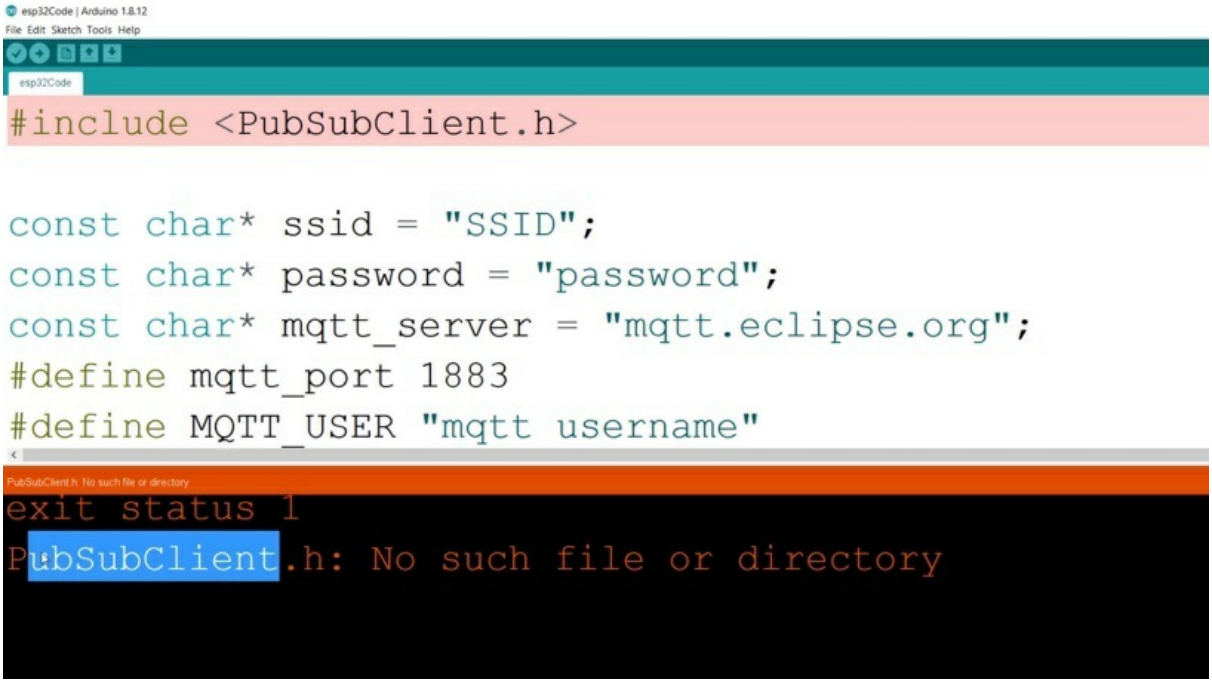
CHA going to court or E. S. P bought. Now, the first thing that we need to do is open up Arduino software. The software that we are going to use to program Arduino or E. S. P tells it to. Now, what we need to do this called is first we need to connect to a Wi-Fi network or access point. Then we need to connect to that, um, Q Titi broadcast. And we also need to send every new line received on the Syria line to the um Q Titi blocker. So I'll do an all send data to the Syrian line and E. S. P will receive this data and it will send it to the Ampule Titi broker or our baby. I whatever you want. It's up to you. Button this cause we are using Amcu D. T. now create a new sketch. The first thing that we need to do is include some libraries. So we need to include the Wi-Fi library. And we also need to include a public subclan Tabari, which is, Atlant, liable for that E. S. P 32 that provides support for Amcu S. T. . Now just try to include. Hashtag includes them inside it. You need to write Bob sub-planned attacks. Now we need to initialize some variables before moving onto the setup and loop functions. And we will start with constant character. For this is all U. B. for our Wi-Fi network. And inside here, you must try it on the Wi-Fi network name. And we need another constant character for the password and we need to write our possible to use the Wi-Fi password. After that, we need the MQ TTR setup link.



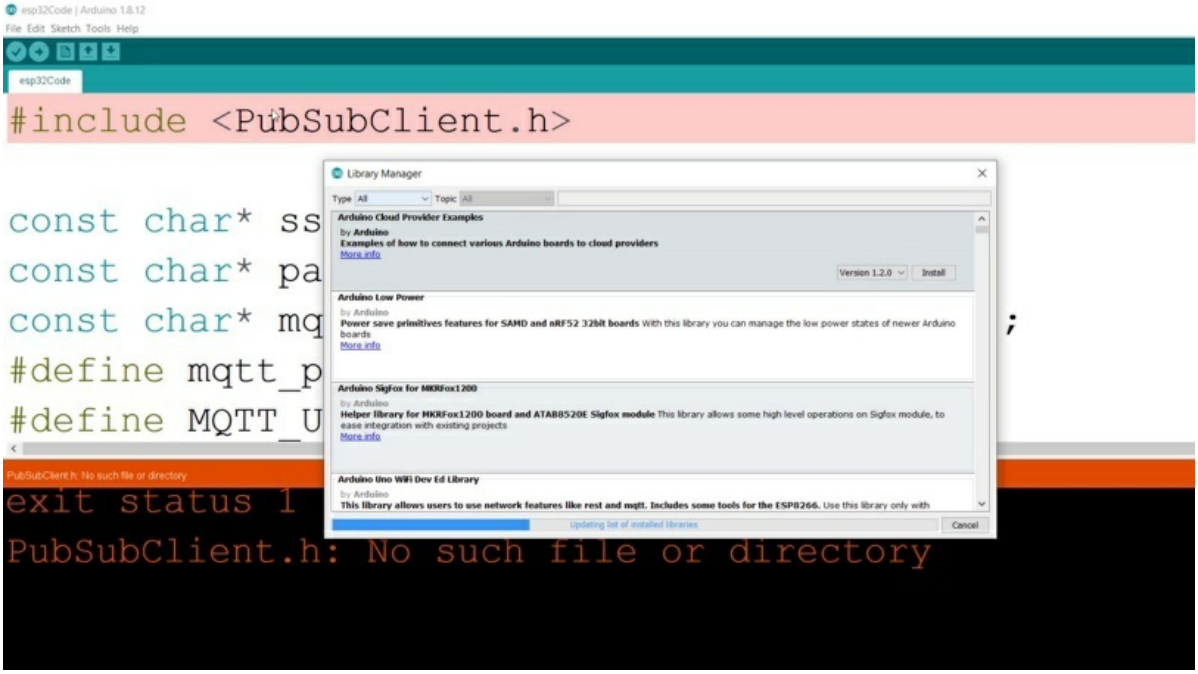
So we need to initialize Khalistan character for q t t server. And inside here, you can buy that server name. We can change that later. Now we need to define some variables. So define mq t t port for them, q t satellite port. And we need to define a user name so uncute, Titi. User. And here we come, right? Thank you. Titi, who was that man after that? We need to define the password. So I'm sure Titi. Password. Then we need that serial populist channel. So when it comes to I define Kutty serial publish. Challenge. And here we can define it depending on what we have. We can change this later. Syrian daughter and all. OK.



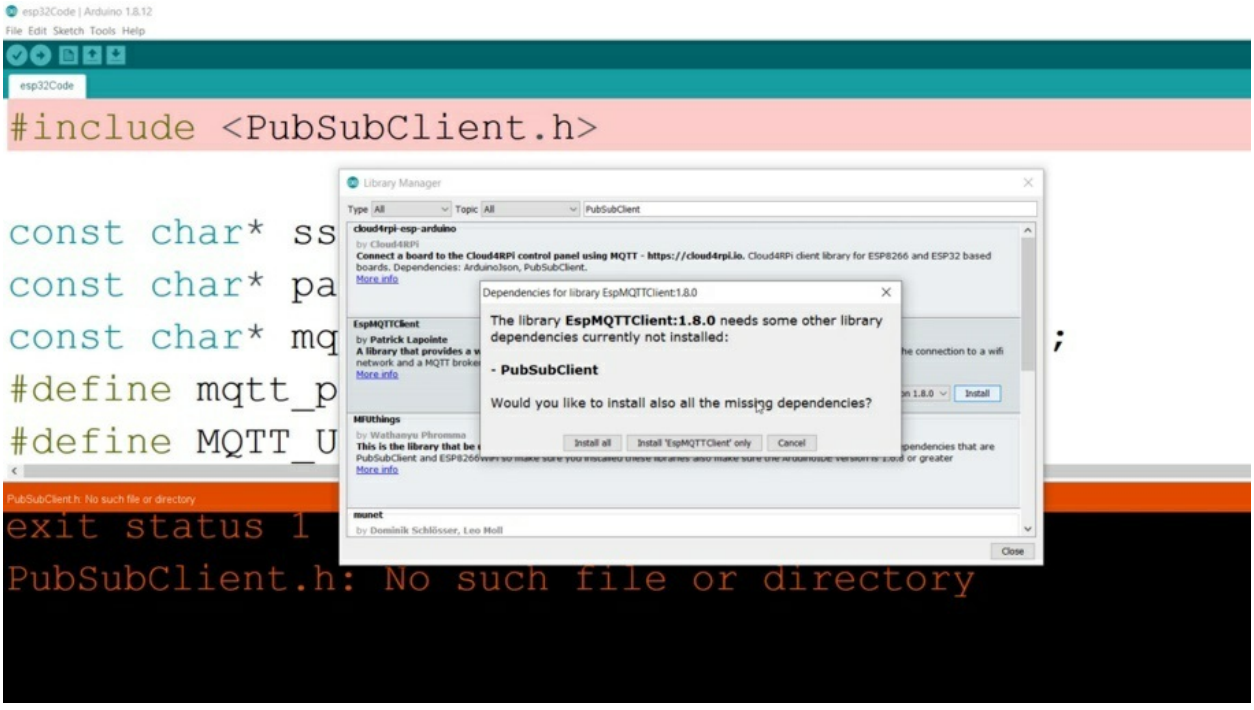
Now we are done with defining and creating variables and including the libraries. We can move on to. The. Suit up Wi-Fi function. But first, let's if this were.



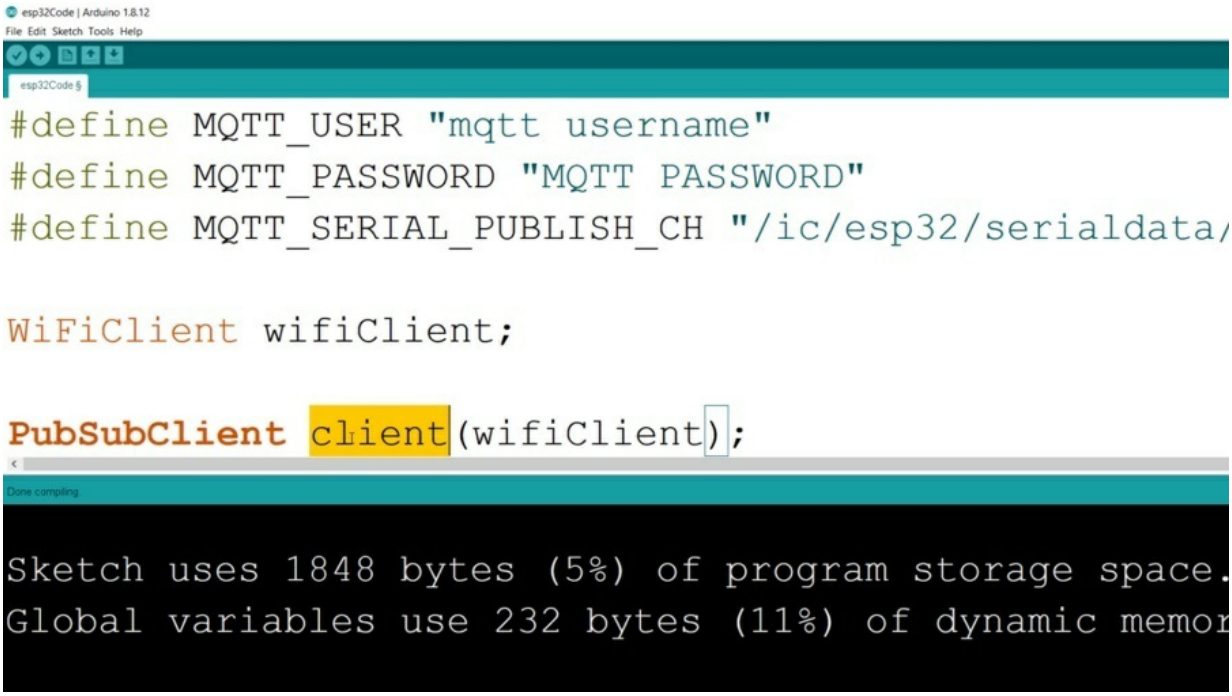
Now, if you did verify your code. We'll see where that is and at all of this library is not defined. No such file or directory.



So I need to go to Tool's managed libraries. And here you need to write pub. Public sub-clients then head into a tool.



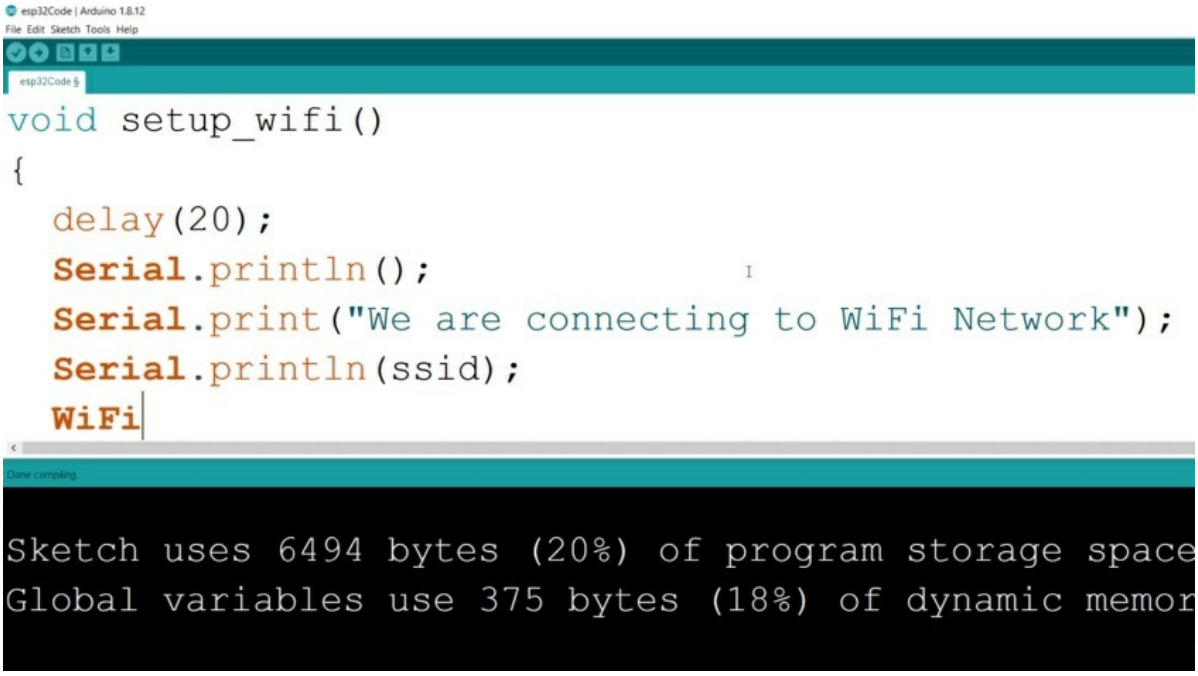
Search for the library, as you can see, this is at E. S. P. I'm to a client. Click, install. Click install on. Now, let's vilify the court again. As you can see now, it's recognizable.



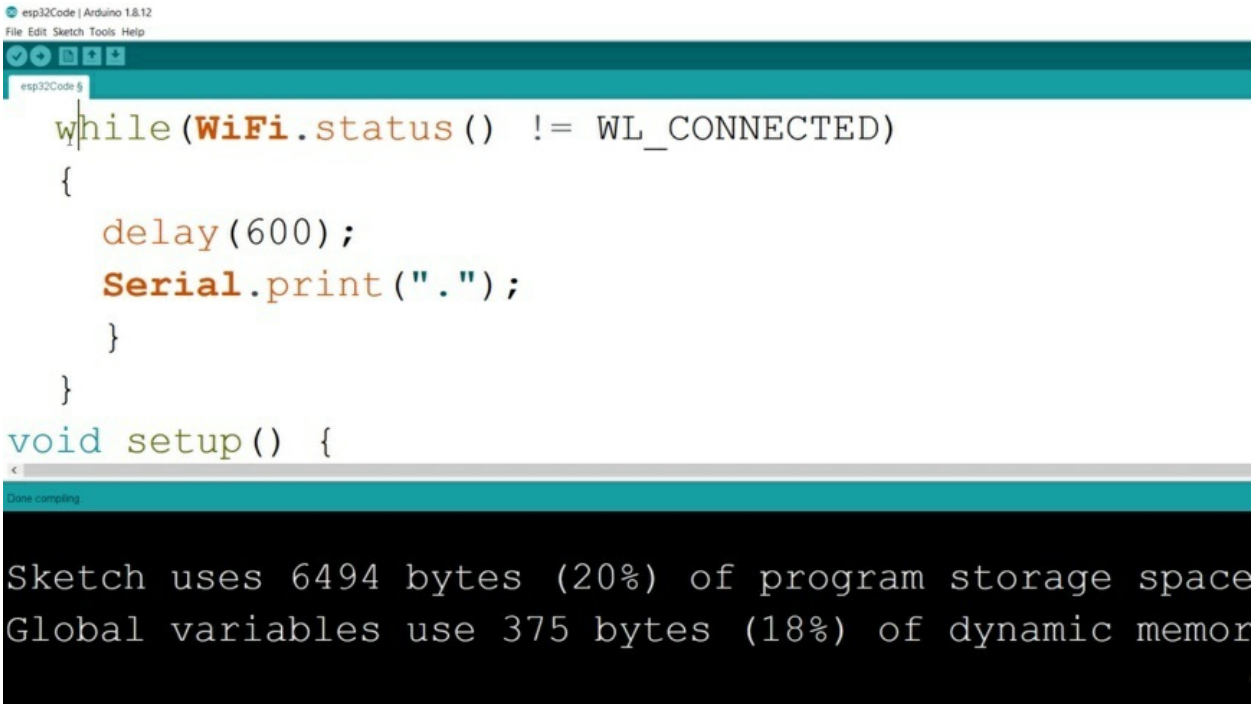
Don, combining and we don't have any drawers. So that's it. This is the first step. Now let's move on the wind to create a new Wi-Fi client object. And we need to create a public sub-client object as well. So Wi-Fi. Plan to take white flight lands, which is this one as a report, and its name will be land. Now, we can't verify the quote being. Then combining without it. That's it. This is the first third, including our E. S. P ball. Next lesson, we are going to explain how you can create a Wi-Fi connection using the yes people.

ESP CODING PART2 WIFI AND MQTT

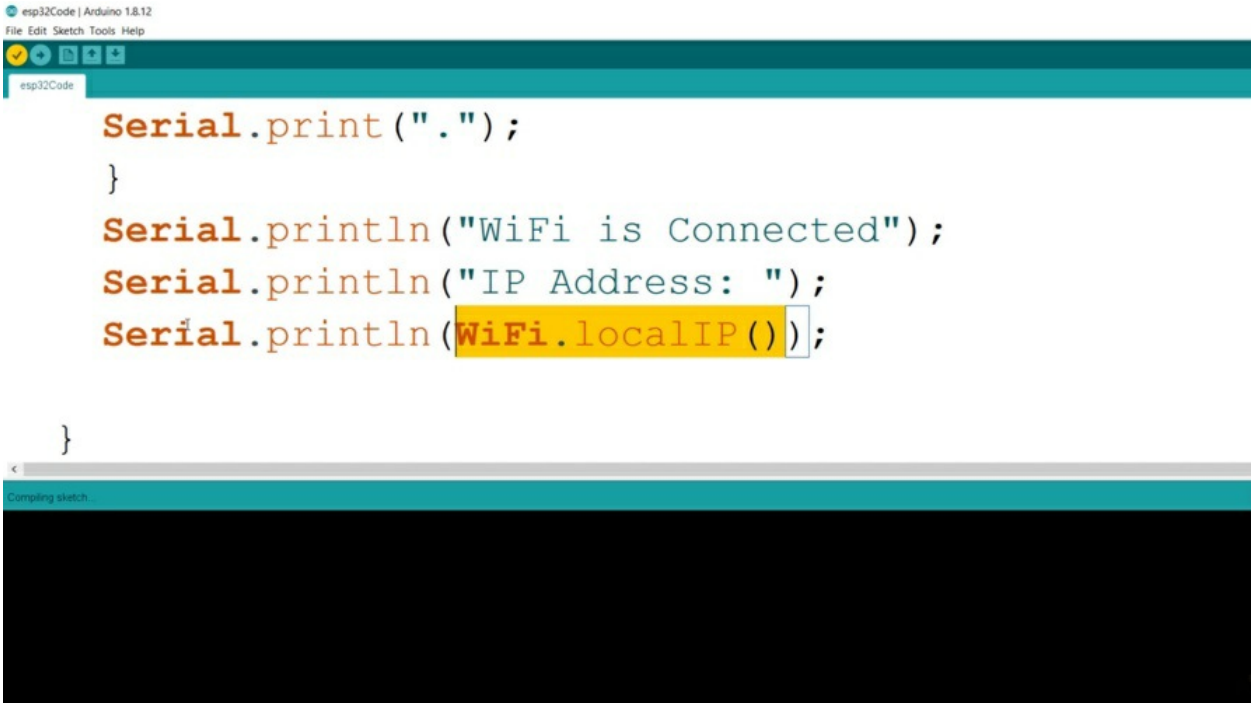
Now that we are done with the first part of our code, let's move on and create a method so that we can connect to the iPhone that works easily. Let's name it void. Who's there to set up Wi-Fi? Now, inside this method to add a delay of 20 milliseconds and Syrian, the prompt new line. Let's print something to tell you, though, that we are connecting to Wi-Fi from the monitors. Okay, now. That's right.



The name of the Wi-Fi. What is his idea after that wildfire Wi-Fi does begin to start connecting to this Wi-Fi? Is this I. D. and password? So it would take the Wi-Fi. This is I. D. and password. Well, try to connect. Now, we can't otherwise statement. If Wi-Fi status. It's not connected then we need to delay and try again.

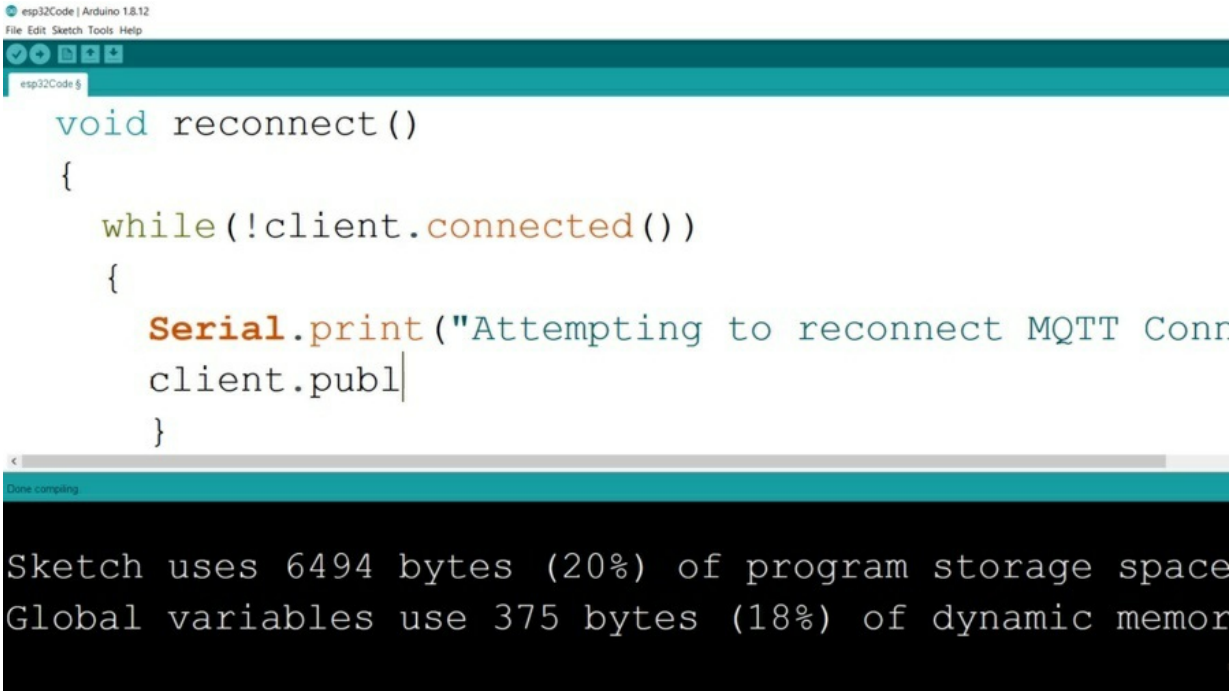


And while it's trying to connect, we needed to plan Dot K now. After this, after a while, that will keep striking. We'll keep trying. Connecting to the Wi-Fi network. We need to add. Once it's connected, it will keep pulling this until it's connected. Now, once it's connected, we need to write a serial footprint, new line. No Wi-Fi is connected.

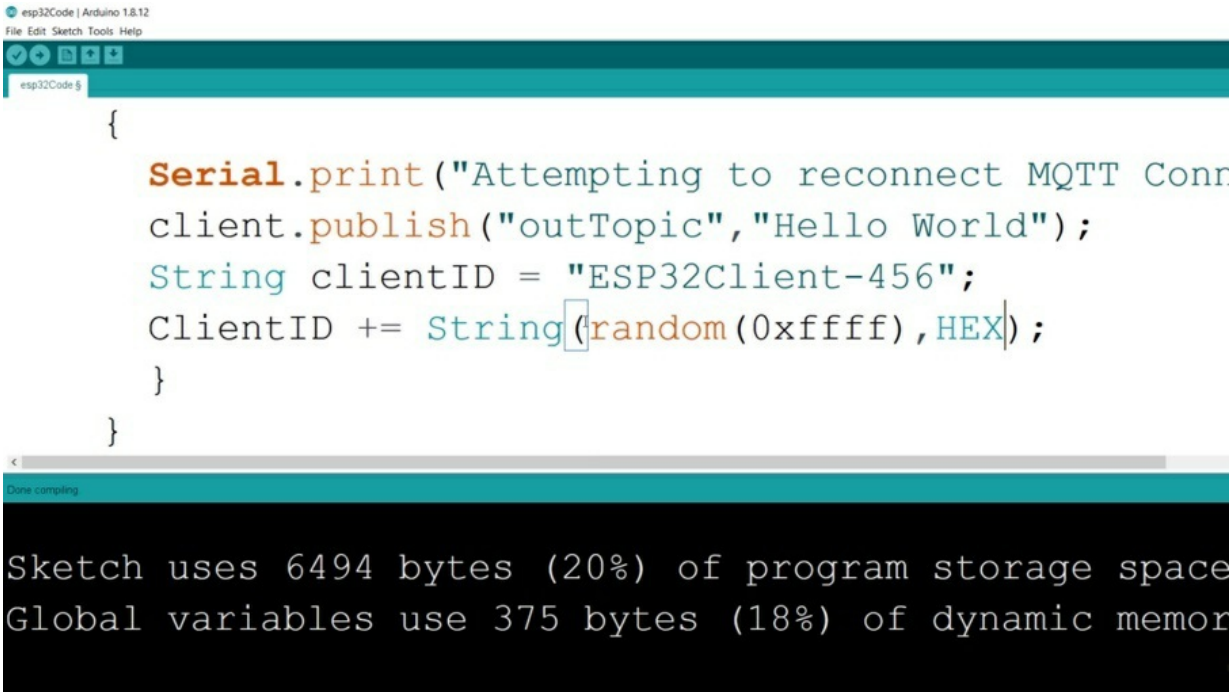


And we can even plan the IP address. And to print it, we need to call a

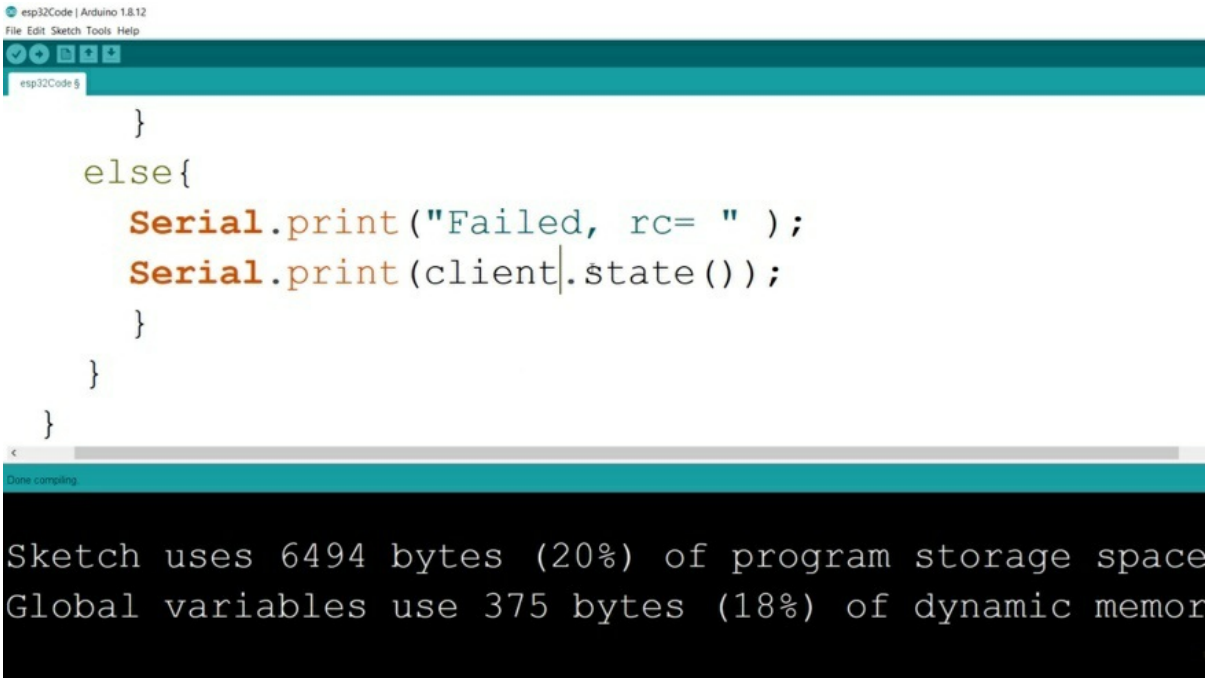
function called Wi-Fi. The local IP. And this will tell. But I'd be others to which our E. S. P 32 boards connected and will print it out on the serial. Now let's verify the code to make sure that everything is working just fine. Kay. That's it. Now, this is the first function.



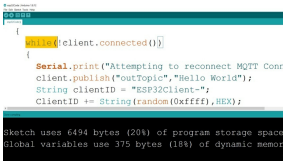
Now the second function is to reconnect by void, reconnect. Now we will loop until we are reconnected. So we will add a wild statement. This is a function to ensure that we are connected to the Q t t cell. So one layer dot connected is not true. We need to try. To connect. So the slightly Brent. A terrible thing to do. I want you to teach. OK. Now the next line will be fly on. The populous inside it. We're right out topic. That's right. Hello. Who are? Now let's create a random client I. D. string line. Heidi. Let's call it E. S. P 30 to play on the Euro Canada. I think after that we can add Clague Heidi. Lusby, one string. And so this strength we can add around or function. Zero X for us. Now we can make sure that Hicks is the tribe. And this will make sure that each time won't have a new client I. D. That is unique because you are on the whole value.



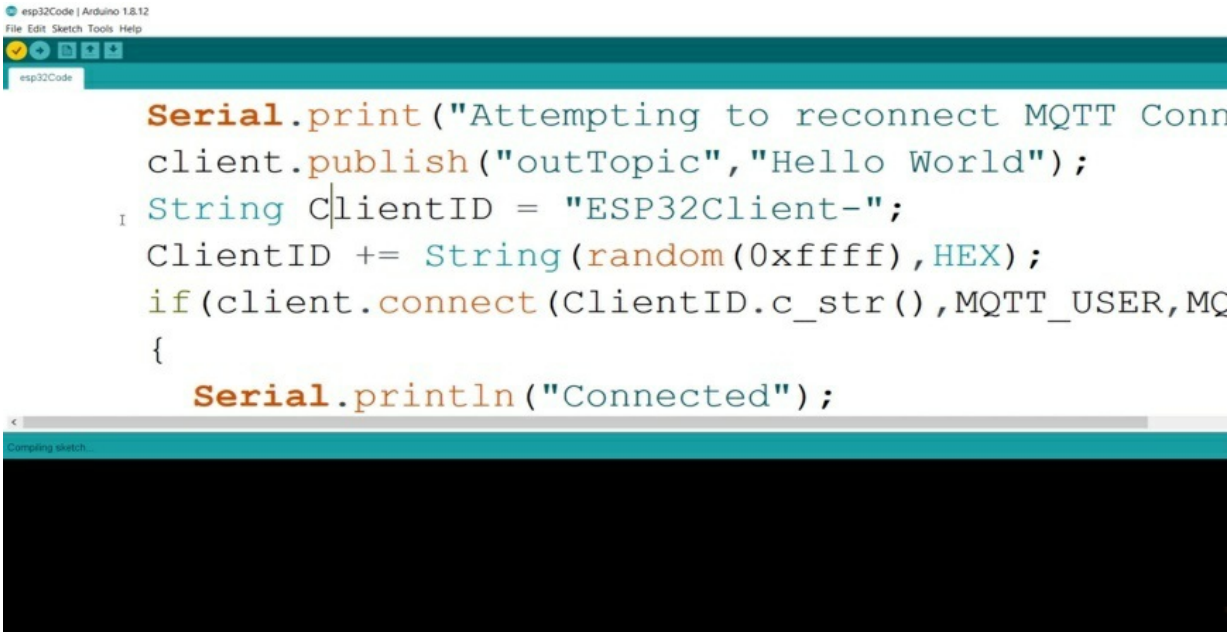
And we are adding it to the previous value so we can remove this number and this time will be different. Will be this. Plus, the last thing that comes out from this line. Now. Let's attempt to connect, right? If statement, then ELDs. Let's start with that statement now inside that statement. We need to ask if a client that cannot function would choose, rotates their client Ivy to this one dot. Now we are going to use a function called C String, which converts the contents of a string as a C SETI, which is the one that we need for this F statement and to use it. We just need to write C, underscore Stee R, then we can add M Q Treaty, use us. She's Boser's name for the server and in Q T password. Pay, let's. Now, as you can see, it takes the connect function, takes three things, the client I. D., those are and the password, so that if the theft statement is through, we can print the word connected and the seal on top. So print a new line. Okay, now, once connected, we can publish an announcement by writing a client, not publish inside that we can write. I see. Rhizomes. And E. S. P, first to. Now, the announcement will be Hello, world.



Now, otherwise, if this condition didn't hit, we would go to the end statement and inside it. We need to right field. And we can also add on another line that says, are sea equal and leave us space. Um, we will print out the land state. So Syria rents land. Let's do it. Now. Let's make sure that we have everything correct. Battleground. The state function. OK, now try again. In five seconds. And we can't add a delay of five or six seconds to give it time to try again.



That's it. This is the reconnect function for the Kutty server. Now we can have this called at the end, but for now, this is everything that we need first. We need to make sure that the client is not connected if it's not connected. We need to right that you are attempting to connect and we need to set Atlant I. D. . Now, if we have seen the client I. D. and cutesie user name and password, uh, it will connect if there is any problem. There is no Internet connection or any other problem.

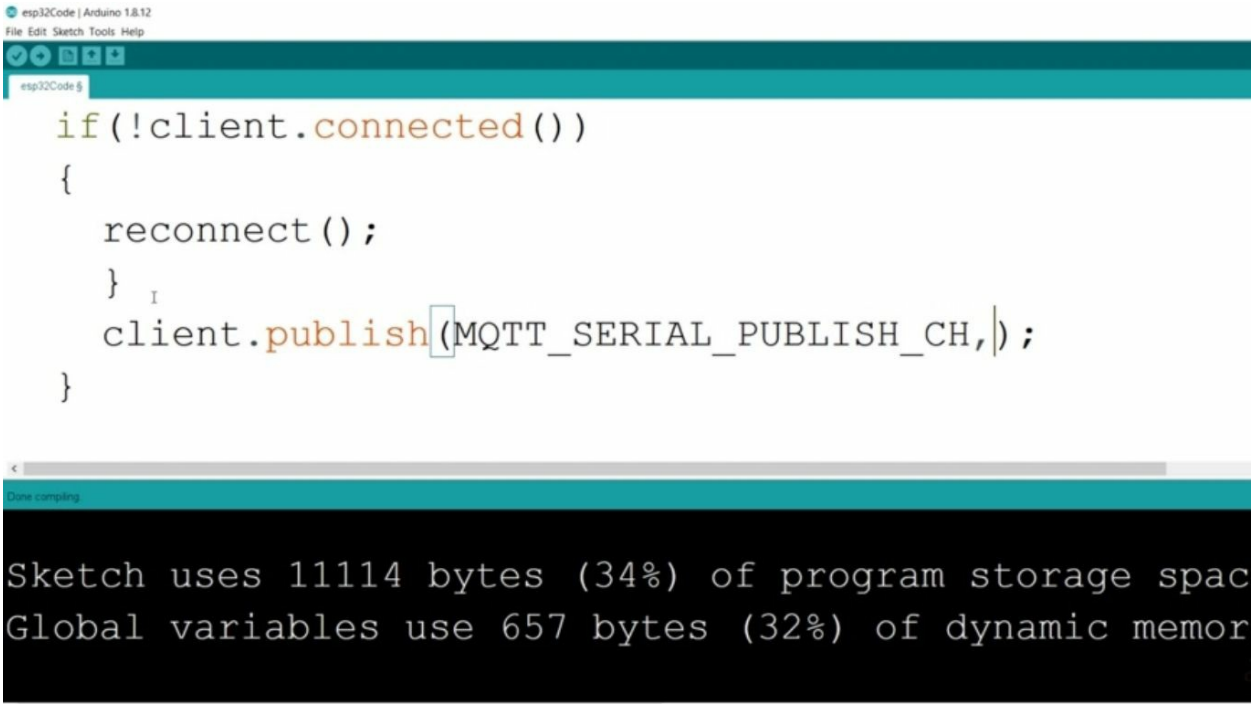


It will print failed to plan the state of this cloud until plan that you should try again in a few seconds and will add a delay to ensure that you are signed after a good amount of time. Now, if we didn't verify our code, then we get. OK. An arrow, that plant, is not defined because we have the C small. So let's make it a capital. Okay. Duncan Mobilising, everything is correct, and in the next lesson, we are going to start the sit-up and lube function clothing.

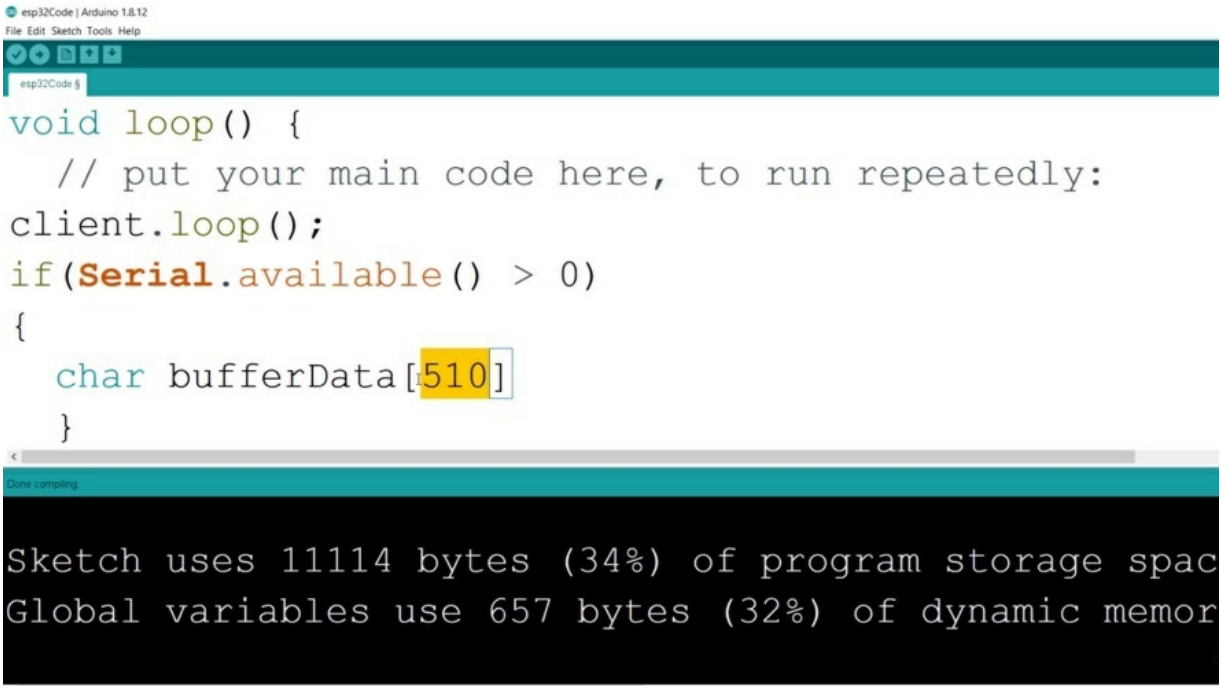
ESP CODING PART3 READ INCOMING DATA FROM ARDUINO

The first thing that we need to do is initialize Pacioli communication to a specific third-rate, a simple date, or in the board. Then we need to set a time out for the serial communication. It will be five hundred milliseconds, so half a second is enough. Then let's call the setup Wi-Fi function. Sure. That we have the right name. This is it. It's the Wi-Fi. After that, we need to call the client to set the server to pass the server parameters and we'll take them Kutty Server Link and Insecurity Port as input. Then we must call the function to reconnect.

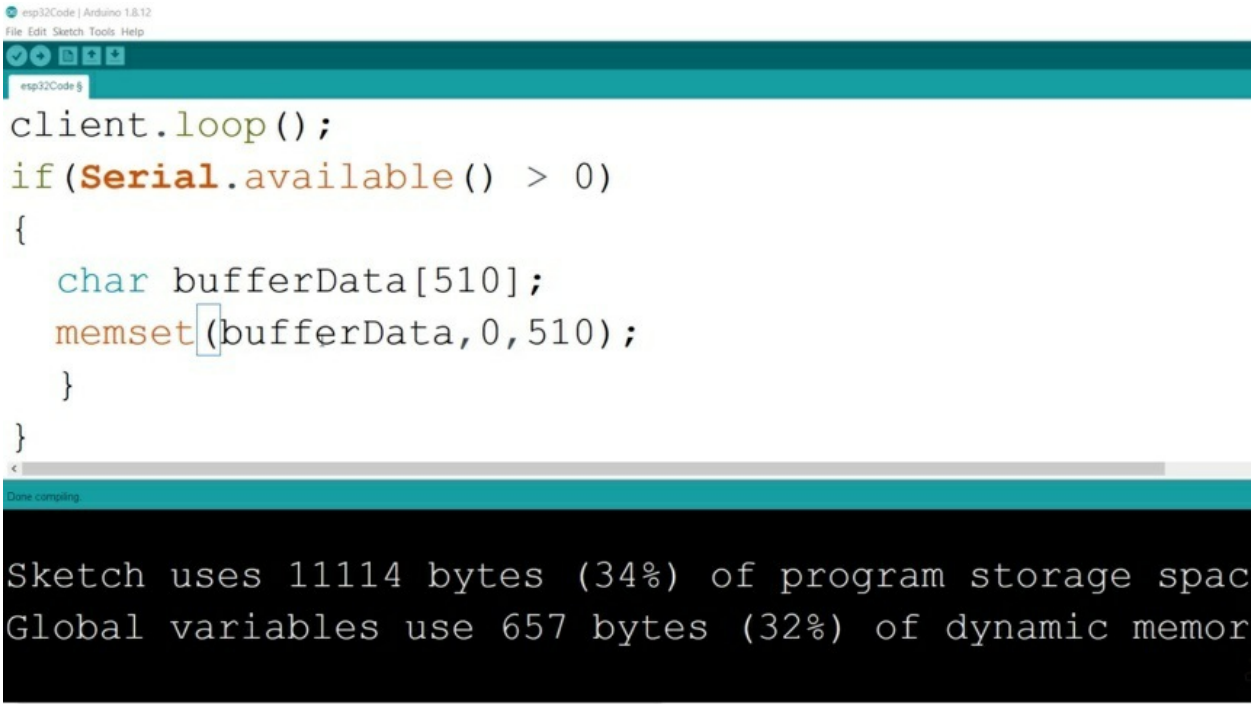
Now let's look at a file called. Great. We don't have any issues. Now we need to define a new function between the setup and Laub, let's call it to publish. So void bubbles. Syria to. Now, this functional take. See if your letter has input. And I thought taking Syria that out and put it will.



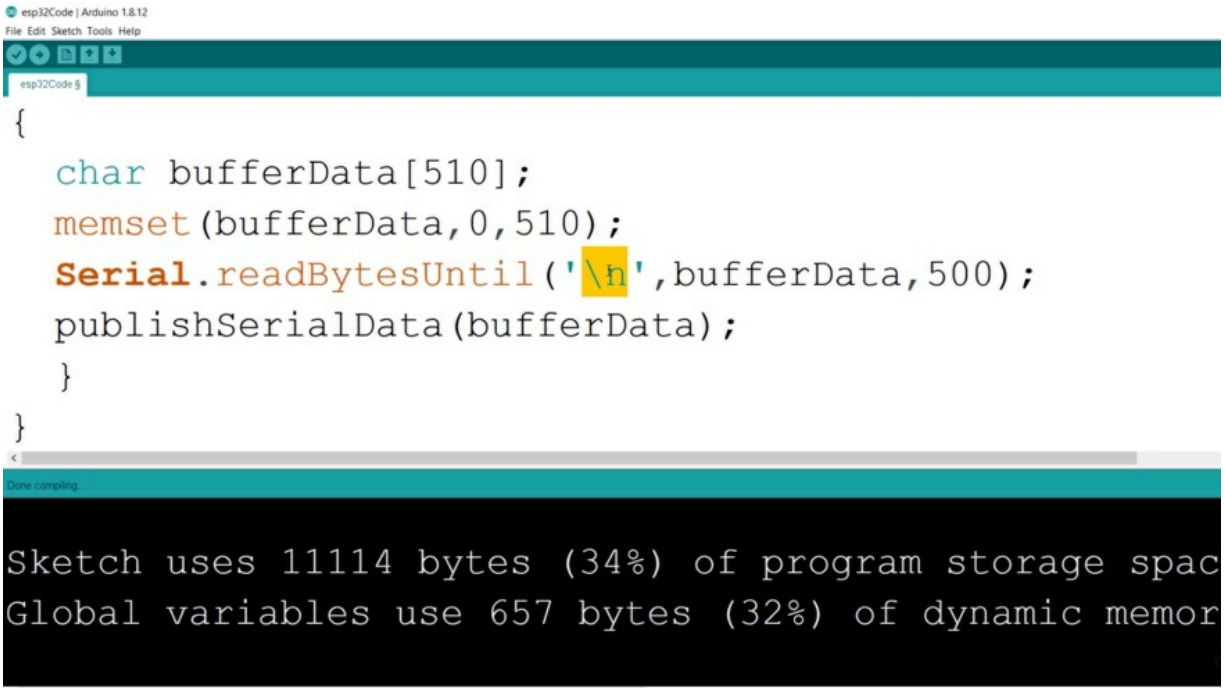
Make sure that the Ben Kutty server was connected using an if statement. Now, inside, they've said and will write if the client did connect to. Now, if it's not connected. Call that iConnect Connect function to connect it. Now, once it's connected. What we need to do is publish the data. So publish and publish will take the MQ Tweetie Serial Publish channel. As in both, and will also take the serial data from here. And this is already defined here. See our published channel. And you can't change it too. On Published Channel. For now, let's go back. Okay, now plan to publish. The client will publish this. Cereal, dirt out of this channel after making sure that the MQ Tweetie server is connected. Now, this is the angle from this function inside the void Logi is going to call this function.



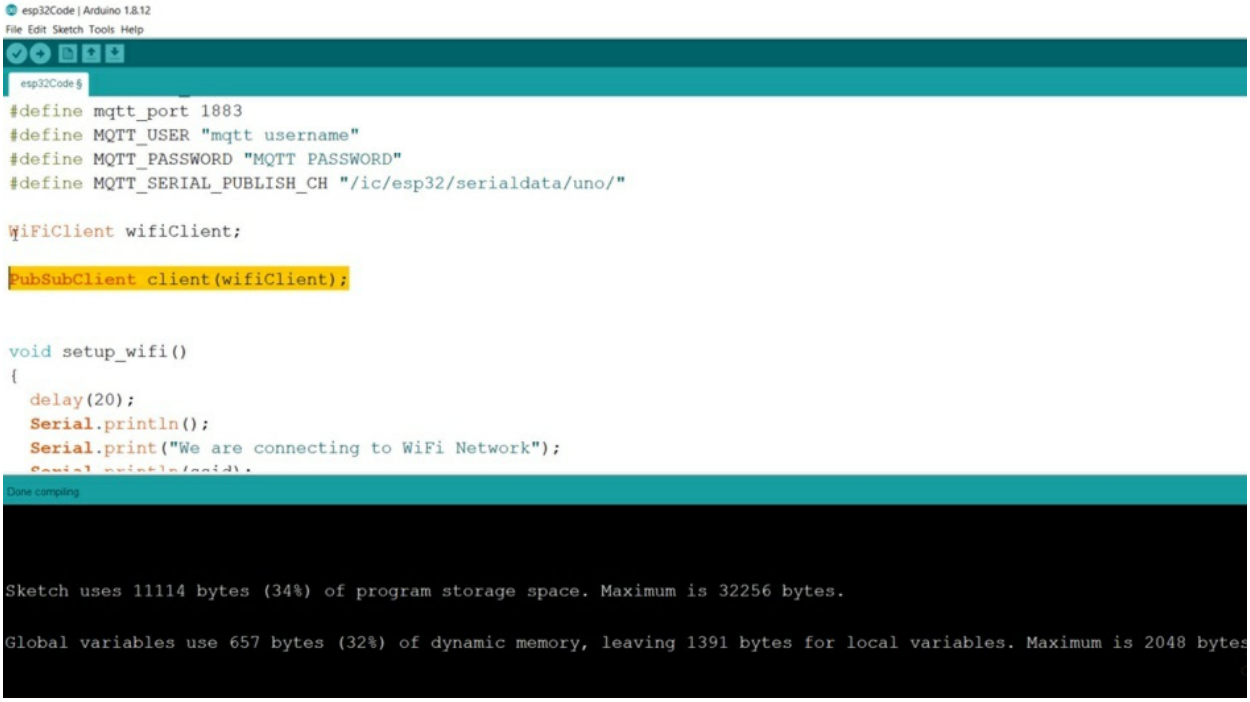
So inside the void law, we will ask. First, let's call our client to doot lube function. Now, we would ask if. That cereal that is available is more than zero or above zero means that we have incoming cereal data. And in that case, if we have incoming cereal, that's what we need to do, is take the cereal that is stored in a variable. So we will define our character. Let's call it Profar. Dr. . And you can make it. This will be an army of characters and its size will be 500. Then you can change the side to whatever fits your needs. After doing this. We are going to use our function Sealab function called Maemi Set. Uh, the C Laro function void Nîmes it Koby's the character to the first in characters of the string pointed by the early. Now I will write it d. Then I will explain to you.



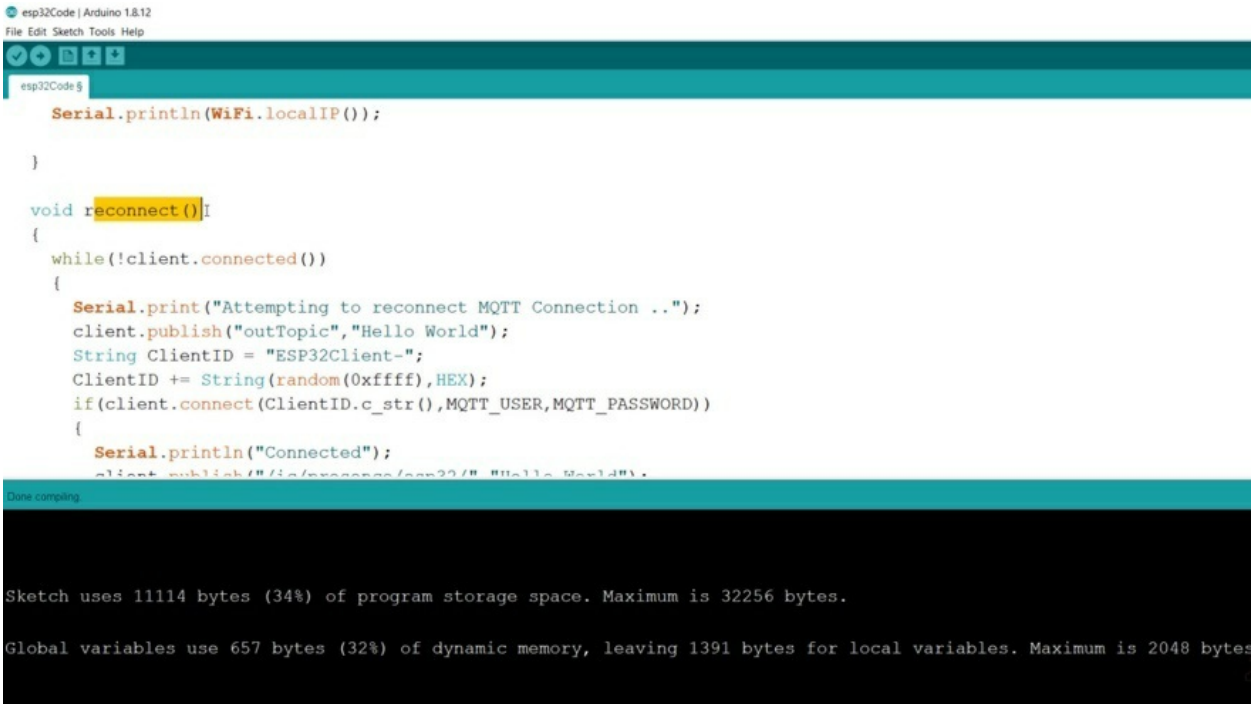
And while you are using it here, the first thing that you need to write here is me set inside it. It will take that buffer as input. And zero, then the buffalo that are of this size, which is five hundred, ten. Now, what this function will do is the following. The first thing that Buffalo data. This is the point after the block of the new order fails. So we need to fill this block of Niemöller with that. Now, in the second part, I'm torture's this number to zero. This is the value to be set. The value is passed as an integer. But the function that fills the block of new is using the unsigned char conversion of the value. Now we come to the last variable or the last iterable, which is five hundred and ten. This is the number of bi's to be said to the value, which is the size of the early. This function will return appoint up to the remote area where Barford that as told. Now we need to read data until we see a new line, which is basically what we did, not under an accord, we say. We have sent data and at the end of each block of data, we are adding a new line.



So. Right, serial the 3D bytes until. And here are going to other conditions. The first thing is LaShun, which is a new line and the data that we will read will be stored here in the Buffalo data and the data size. Let's make it 500. We already have an RJ with a size over 500, 510. So 500 is the safe value to be inserted here. Usually, you have to set a below bad value. Sighs After this. And then you need to publish this data. So bubbles Syria data. This is the function that we defined here. And we'll take that that that we did receive as input, which is the heart of that. So this function will be the incoming Sevele data and sort it here until a new line is sent from all the NTSB. Then it will send the data to the bubbles serial data function, which takes that as input and this functional, make sure that I'm Kutty server is connected after making sure that it's connected. It will publish the data to the Kutty serial channel. And as you can see, this is the serial that is the main function of our coding practice here. Now again, what we have done here.

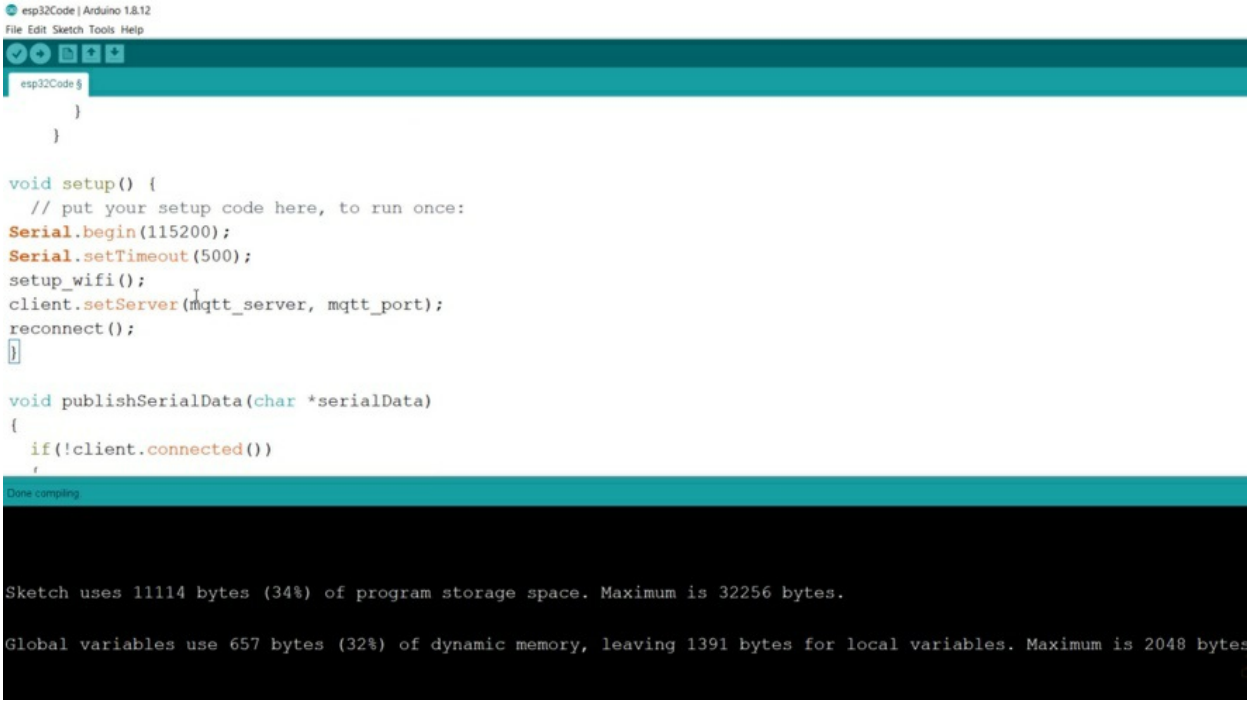


Let me zoom out. We have created defined some variables and we have used them, Kutta T Ledbury, the wildfire library set up wildfire to make sure that we are connecting or we are connected to our Wi-Fi network. And we are online.

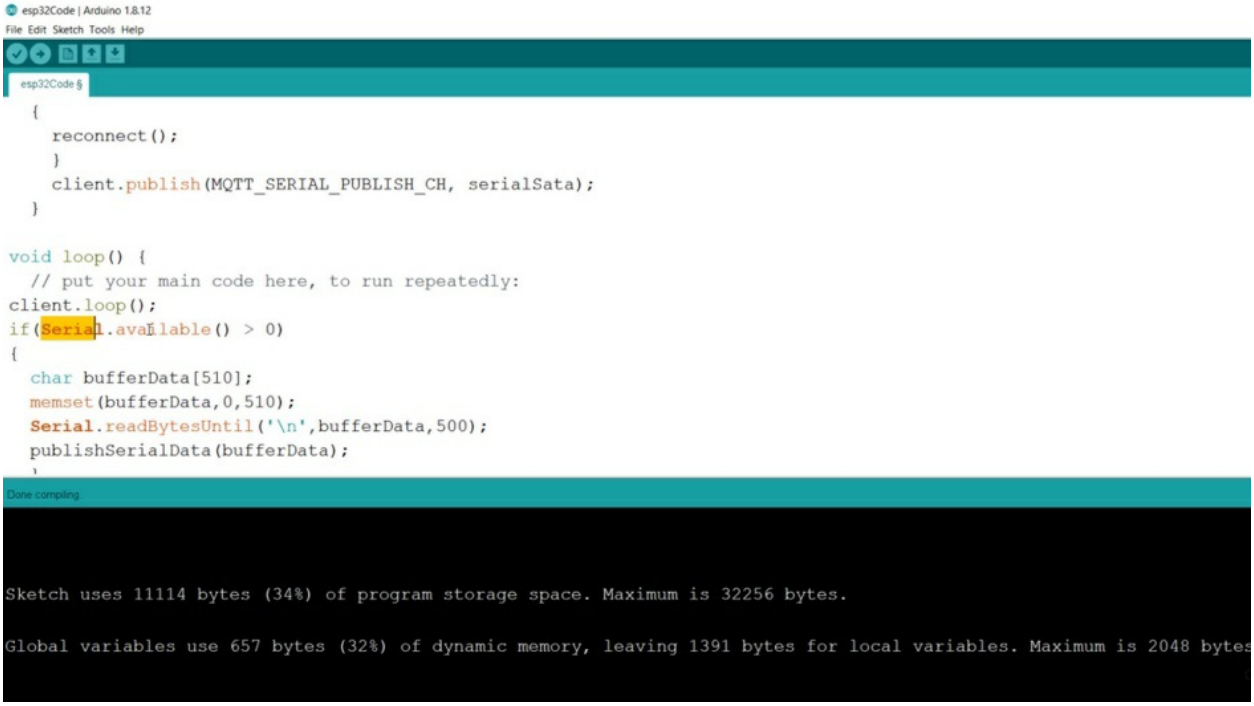


That it cannot function is to make sure that we are connected to the amputee's server and we have the setup function where we have connected to Wi-Fi and

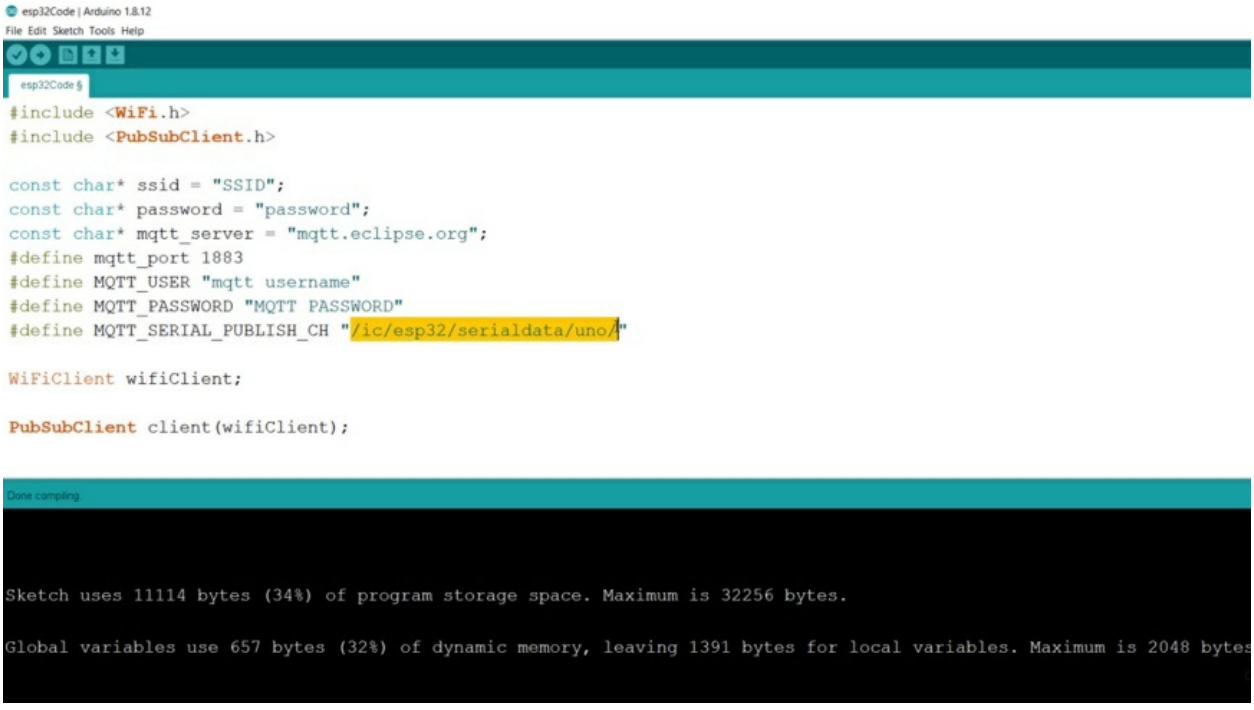
them Kutta t cell.



Well, then we have the loop which will make sure that we are over. We have incoming Syril data. And it will store the incoming serail that is inside this variable bifold that she's L'Abri of character. And will send it to the published serial data function, which is this one first. We will make sure that we are connected to them. Kutty server. Then we will publish the data to this channel that we are defined here.



So this is our main coding practice for ISP 32. And the next lesson we are going this out and make sure that it's working just fine.

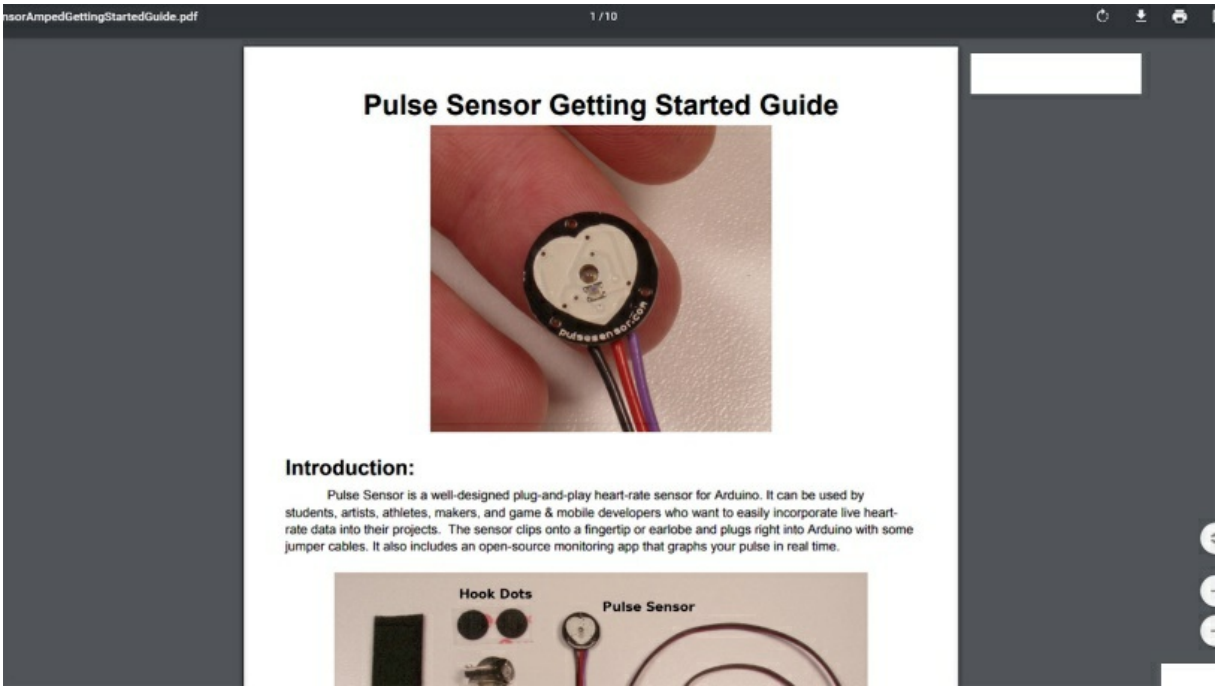


If you have any questions regarding any of these lines, please ask that you are enabled. Now, just to make sure that we don't have any arrows. Let's verify our code. One more time. Okay, now we have an error, as you can see, see

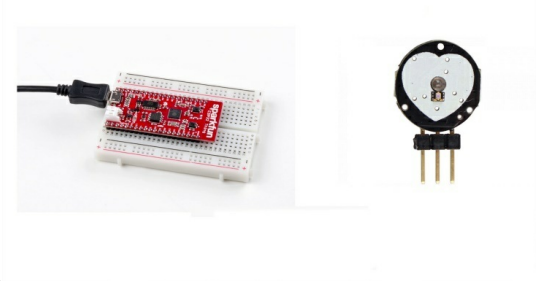
Siedel that was not declared. It's here. So we have spelling errors for the fire again. Done combining without any errors. That's it. Now, we might have to change the values for our Wi-Fi network name and password and the impurity user name and password, and the channel or the server links depending on our needs. Now, if you are using your API, you can simply send the serial data to your API using Jason coding or any other method that fits your needs, your needs. Bu t for now, I'm using them Kutta server. So this is the coding practice for the server. It's a very well-kn and widely used one. So I will change the Wi-Fi network values and the user name and password values with mine.

BLE SERVER AND CLIENT COMMUNICATION

We will learn the following How to establish a BLE connection between the two ESP32 Things one set as server and the other set as client Send data from the Server to the client In we looked at how to establish a BLE connection between the Thing as a server and your smartphone as the client Now let's look at how to establish one ESP32 as a BLE server and the other ESP32 as a BLE client We will be using a pulse sensor for this project



We recommend you to go through the pulse sensor getting started guide in



the resources section before moving forward

We will be reading the raw analog values from the pulse sensor using one of the ADC pins on the BLE Server thing We will then be detecting a heartbeat if the value of the pulse sensor rises above a certain threshold If a heartbeat is detected, we will notify the BLE client T hing using the notify property

```
   3. Create a BLE Characteristic on the Service
   4. Create a BLE Descriptor on the characteristic
   5. Start the service.
   6. Start advertising.

   A connect hander associated with the server starts a background task
   every couple of seconds.
*/
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;
bool oldDeviceConnected = false;
```

Up until now, we had used the read and write property Lets now see how to use the notify property First let's look at the code on the BLE server-side We need to import these libraries for the prebuilt BLE functions Here we are initializing pointers and variables which we will be using later in the code

```
*/
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;
bool oldDeviceConnected = false;

int PulseSensorPurplePin = 34;        // Pulse Sensor PURPLE WIRE connec
int LED13 = 5;   //  The on-board Arduion LED


int Signal;                // holds the incoming raw data. Signal value
int Threshold = 3900;      // See the following for generating UUIDs:
```

Next, we are assigning the pulse sensor signal pin as PIN34 on the Thing Here we are initializing the Led pin as Pin no 5, which is the PINto access the onboard LED on the Thing Then, we create a variable signal to store the readings from the sensor We also assign a certain value for threshold here which will discuss later in the code

Next, we define the Service UUID and the Characteristic UUID for the

customer service and characteristic which we will be using Next we have the My Server call back which will get executed when the Server has connected or disconnected from the client Inside the Callback when the BLE connection is successful the boolean variable device connected is set to true value If the BLE connection gets disconnected, then the device connected variable is set to False This is done to know the status of your BLE connection

```
void setup() {
  Serial.begin(115200);

  // Create the BLE Device
  BLEDevice::init("ESP32");

  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // Create a BLE Characteristic
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_READ   |
                      BLECharacteristic::PROPERTY_WRITE  |
                      BLECharacteristic::PROPERTY_NOTIFY
                    );
```

Now coming to the setup code, it's pretty much the same as seen in the for the BLE server Here we are setting the server callback for the device connection status The process to create the BLE server and the BLE characteristic is the same as seen in the key difference is that this time we have included the Notify property for the custom characteristic The notify property enables us to send messages from the server to the client device

```
// Create a BLE Descriptor
pCharacteristic->addDescriptor(new BLE2902());

// Start the service
pService->start();

// Start advertising
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pAdvertising->setScanResponse(false);
pAdvertising->setMinPreferred(0x0);  // set value to 0x00 to
BLEDevice::startAdvertising();
pinMode(LED13, OUTPUT);
Serial.println("Waiting a client connection to notify...");
```

Now this is used to create the BLE descriptor for the characteristic to enable notifications Next, we start the service Here we set the advertising configurations and start the advertising process Next, we set the onboard LED pin as the output We will use this to confirm the presence of a heartbeat Next we print this message to the serial monitor, to indicate that the server has started advertising and is waiting for the client to connect Now coming to the loop function

```
void loop() {
  // notify changed value
  Signal = analogRead(PulseSensorPurplePin);  // Read the Pulse
  // Assign this value to the "Signal" variable.

  Serial.println(Signal);
  if (deviceConnected && (Signal > Threshold)) {
    digitalWrite(LED13, HIGH);
    pCharacteristic->setValue("Heartbeat detected!");
    pCharacteristic->notify();
    // bluetooth stack will go into congestion, if too many pac
  }
  else {
    digitalWrite(LED13, LOW);                //  Else, the sigal
```

we use the Analog read function to read the analog value from the previously assigned variable for pin no 34 PIN34 on the Thing is a 12 bit ADC, so we will get a value from the sensor in the range of 0 to 4095 Now we store this value in the variable signal Now we print the value of the signal to the serial port

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b         361b
```

```
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};
```

Remember the My server callback function which we had used to specify the

BLE connection status? We will be using the variable device connected here

```
void loop() {
  // notify changed value
  Signal = analogRead(PulseSensorPurplePin);  // Read the Pulse
  // Assign this value to the "Signal" variable.

  Serial.println(Signal);
  if (deviceConnected && (Signal > Threshold)) {
    digitalWrite(LED13, HIGH);
    pCharacteristic->setValue("Heartbeat detected!");
    pCharacteristic->notify();
    // bluetooth stack will go into congestion, if too many pac
  }
  else {
    digitalWrite(LED13, LOW);                //  Else, the sigal
  }
```

So now we check, if the device connected is TRUE and our signal is above the threshold, we will execute the following code Now let's talk about the Threshold We had set a certain threshold value earlier in the code which will help us to identify a heartbeat If the signal goes above the threshold value, it means that it's a valid heartbeat, else it's not Here we are checking whether the server is connected to the client and whether we have received a valid heartbeat If both conditions are true, we will use digital write to switch on the onboard LED We will also set the BLE characteristic value as a string Heartbeat detected Next we will call the notify function to send the above message as a notification If any of the two conditions are false we will switch off the onboard LED

```
#include "BLEDevice.h"
//#include "BLEScan.h"

// The remote service we wish to connect to.
static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
// The characteristic of the remote service we are interested in.
static BLEUUID    charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;

static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
```

Now let's look at the Client-side code We have first included the necessary libraries Next, here we specify the service UUID of the service we wish to connect to In this case, it should be the Service UUID we had specified in the Server-side code Here we specify the characteristic UUID we wish to access In this case it should be the Characteristic UUID we had specified in the Server-side code Here we are just initializing variables which we will be needing as we go further in the code

```
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;

static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
    Serial.println(length);
    Serial.print("data: ");
    Serial.println((char*)pData);
}

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pclient) {
  }
```

This is the notify callback This gets executed whenever there is a notification from the server-side This is used to print the UUID of the server characteristic from which we are getting notifications This is used to print the length of the string which we are receiving as the message in the notification This is used to print out the actual string data received My client callback is similar to my server callback and is used to check the connection status Now the connect To server is the actual function which is used to connect to the server If the connection is successful, it will return TRUE Now this is used to print the device address we wish to connect to, in this case being the BLE server Thing next this is used to create a BLE client

```
    connected = false;
    Serial.println("onDisconnect");
  }
};

bool connectToServer() {
    Serial.print("Forming a connection to ");
    Serial.println(myDevice->getAddress().toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    // Connect to the remove BLE Server.
    pClient->connect(myDevice);  // if you pass BLEAdvertisedDevice instead of a
    Serial.println(" - Connected to server");

    // Obtain a reference to the service we are after in the remote BLE server.
```

Here we set the client callback for connection status similar to the server-side Now this is used to connect to the server device Here we try to obtain the Service UUID of the BLE server

```
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    // Connect to the remove BLE Server.
    pClient->connect(myDevice);  // if you pass BLEAdvertisedDevice instead of a
    Serial.println(" - Connected to server");

    // Obtain a reference to the service we are after in the remote BLE server.
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our service");


    // Obtain a reference to the characteristic in the service of the remote
```

If there is no service UUID available it will disconnect from the server and return false If there is a service available, it will printout this message The same thing we do for the characteristic UUID

```
    pClient->disconnect();
    return false;
  }
  Serial.println(" - Found our characteristic");

  // Read the value of the characteristic.
  if(pRemoteCharacteristic->canRead()) {
    std::string value = pRemoteCharacteristic->readValue();
    Serial.print("The characteristic value was: ");
    Serial.println(value.c_str());
  }

  if(pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);
```

Here we are just reading the value of the characteristic and printing it as a string value

```
  if(pRemoteCharacteristic->canRead()) {
    std::string value = pRemoteCharacteristic->readValue();
    Serial.print("The characteristic value was: ");
    Serial.println(value.c_str());
  }

  if(pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);

  connected = true;

* Scan for BLE servers and find the first one that advertises t
*/
lass MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCal
```

Now we will check if the notification is enabled on the server-side If it is, the notify callback will get executed

```
    if(pRemoteCharacteristic->canNotify())
      pRemoteCharacteristic->registerForNotify(notifyCa_____,,

    connected = true;
}
/**
 * Scan for BLE servers and find the first one that advertises the se
 */
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallback
 /**
   * Called for each advertising BLE server.
   */
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());
```

Then we set this variable as true We will see why as we progress further in the code Here we specify the advertised devices callback This function is called for each advertising BLE server So if we have only one advertising server, it will display its name, address service UUID

```
  if(pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);

  connected = true;

*
 Scan for BLE servers and find the first one that advertises the service we are looking for.
/
ass MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
**
 * Called for each advertising BLE server.
 */
void onResult(BLEAdvertisedDevice advertisedDevice) {
  Serial.print("BLE Advertised Device found: ");
  Serial.println(advertisedDevice.toString().c_str());

  // We have found a device, let us now see if it contains the service we are looking for.
  if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(serviceUUID)

    BLEDevice::getScan()->stop();
    myDevice = new BLEAdvertisedDevice(advertisedDevice);
```

Now here we check if the service with the specific Service UUID we are looking for is available in the advertised device or not If it is, we stop the BLE scan for advertised devices, and assign the newly found device to my device variable We then specify these two variables as true we will see why

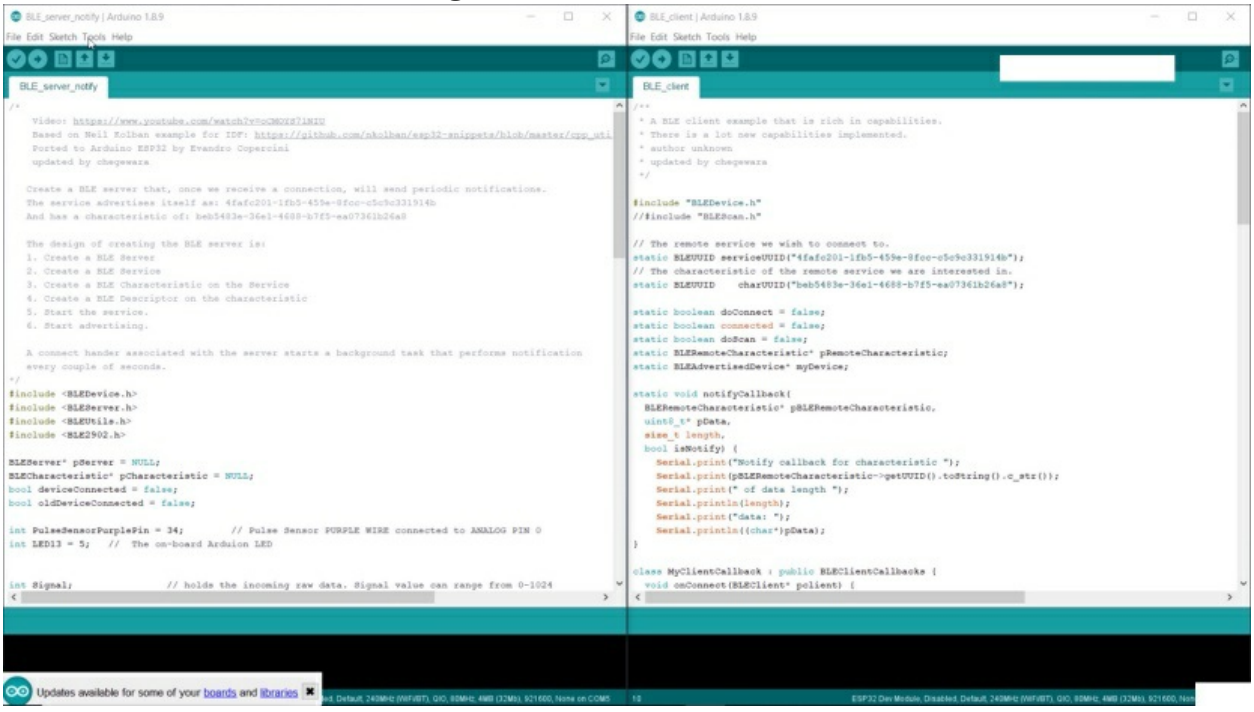
later Now coming to the setup code We are just initializing the BLE client device here Next we will call the BLE scanner callback for advertised devices which we had included earlier Next we set the BLE scan configurations and start the BLE scanner for 5 seconds

```
void setup() {
  Serial.begin(115200);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");

  // Retrieve a Scanner and set the callback we want to use to be
  // have detected a new device.  Specify that we want active sca
  // scan to run for 5 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCa
  pBLEScan->setInterval(1349);
  pBLEScan->setWindow(449);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(5, false);
} // End of setup.
```

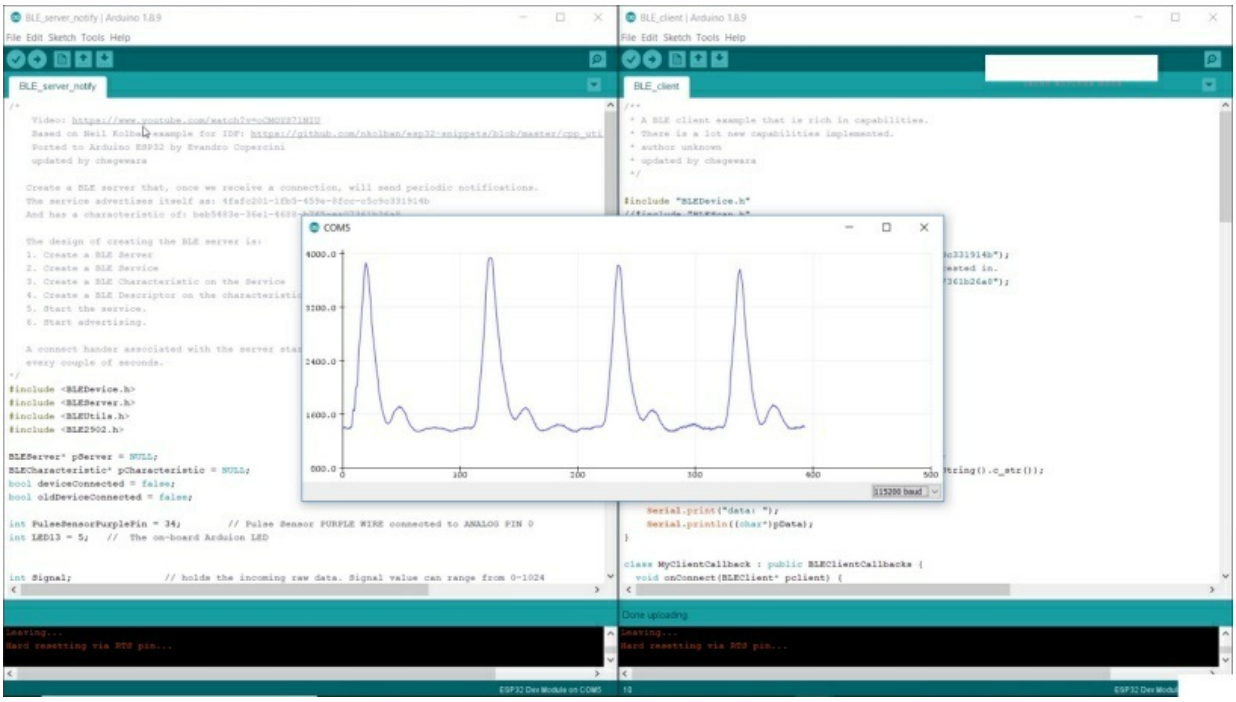
Now coming to the loop code, we will check here if the do Connect variable is True or not If it is, it means that we have successfully scanned the desired server device we wish to connect to Now here if we have formed the BLE connection with the server successfully, the connect to server function will return true and the if condition will get executed This will printout the final message confirming the successful BLE connection of the server and client

```
pBLEScan->setWindow(449);
pBLEScan->setActiveScan(true);
pBLEScan->start(5, false);
// End of setup.

// This is the Arduino main loop function.
void loop() {

  // If the flag "doConnect" is true then we have scanned for and found the desired
  // BLE Server with which we wish to connect.  Now we connect to it.  Once we are
  // connected we set the connected flag to be true.
  if (doConnect == true) {
    if (connectToServer()) {
      Serial.println("We are now connected to the BLE Server.");
    } else {
      Serial.println("We have failed to connect to the server; there is nothin more we will do.");
```

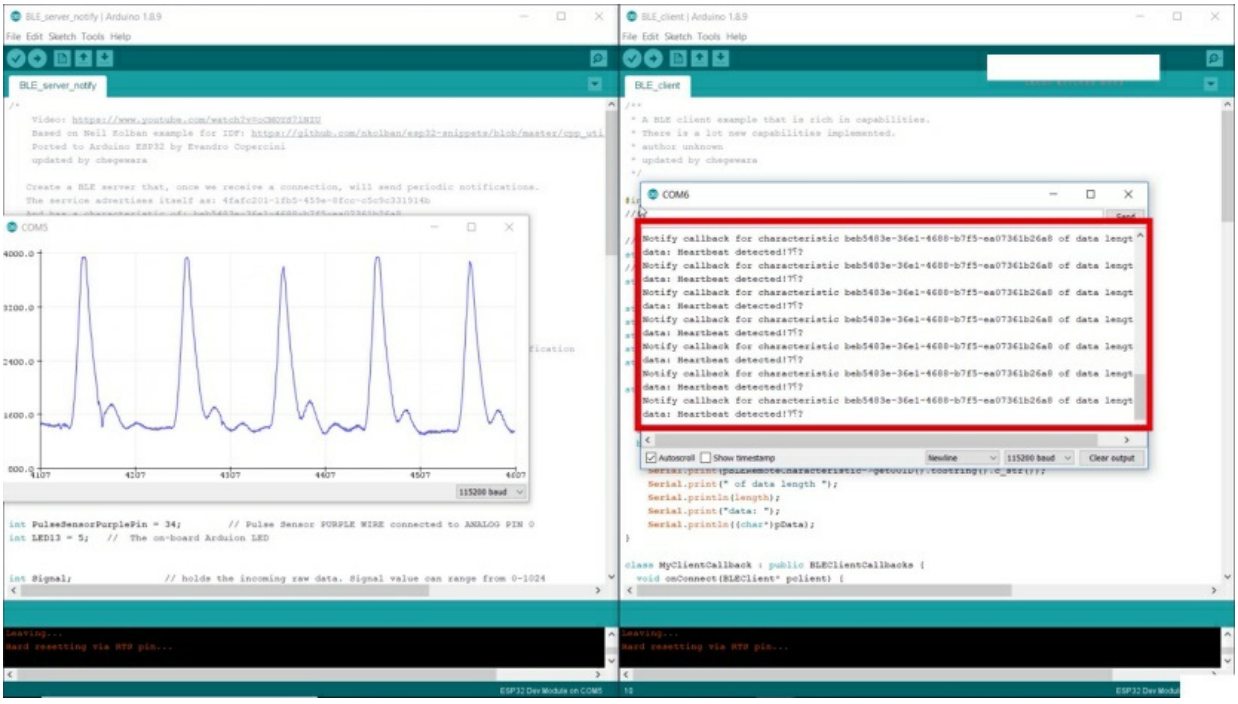
Else it will show this message for connection failure



Now let's upload the code and see it in action Select the correct COM port for your server Thing First upload the Server-side code so that the Server Thing can advertise itself Select the correct COM port for your Client Thing Now upload the Client-side code so that the client can scan and connect to the server

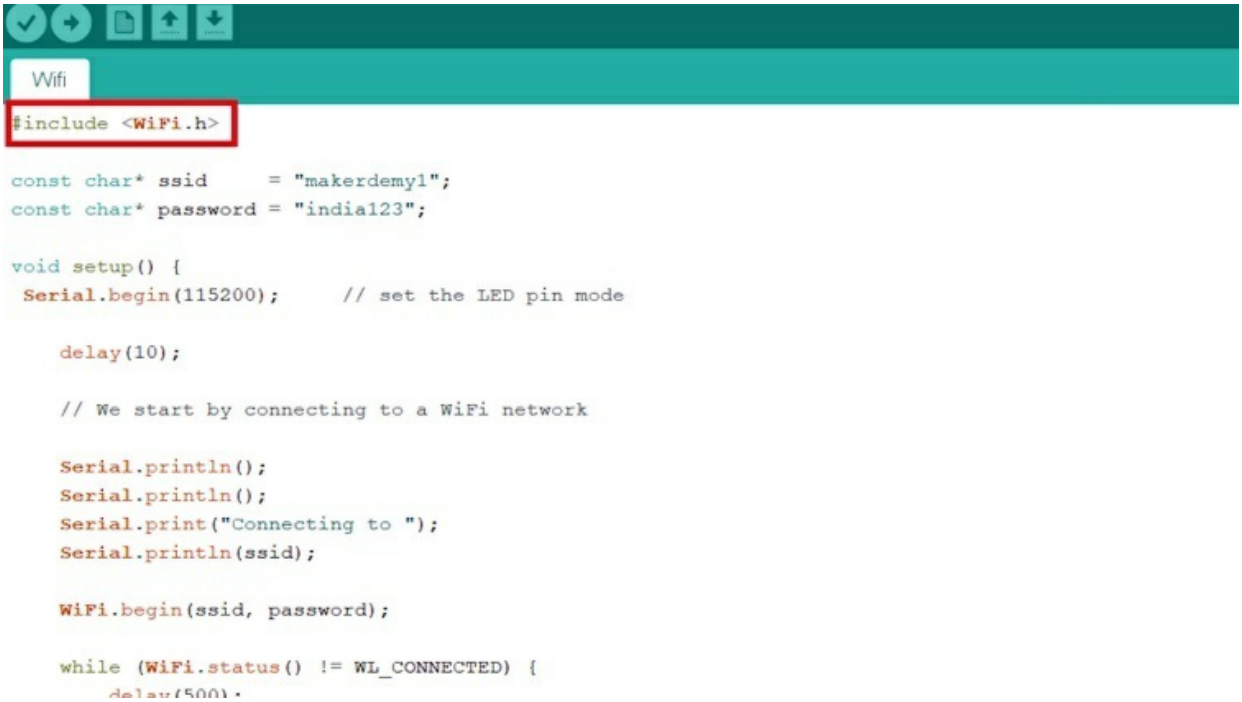


Now open up the serial plotter in the Server code You will see the heartbeats plotted graphically Looks amazing isn’t it? Now opening up the serial plotter means that your code has been executed and the BLE server has started the advertisement



ESTABLISHING WI-FI CONNECTION WITH THE THING

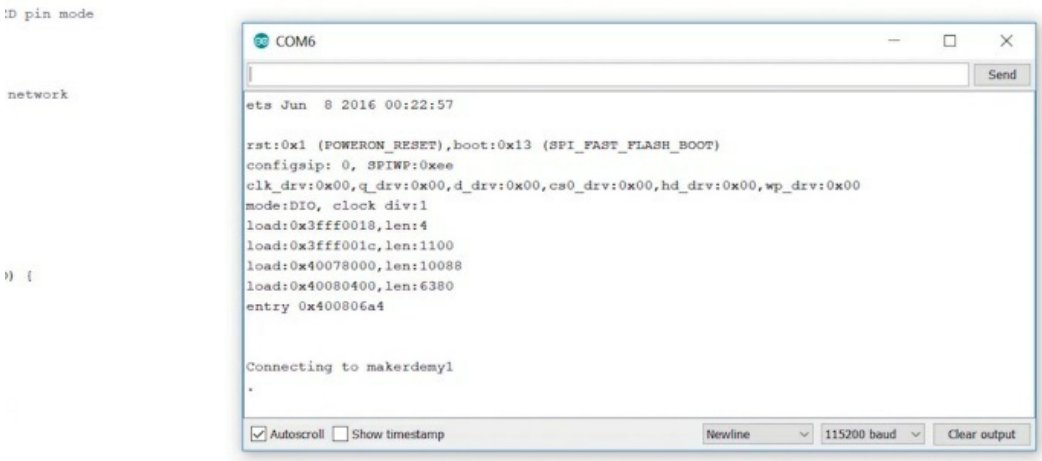
The thing to a WiFi network How to control the On-Board LED on the Thing via the web Let’s look at the code to connect the Thing to your Wi-Fi network



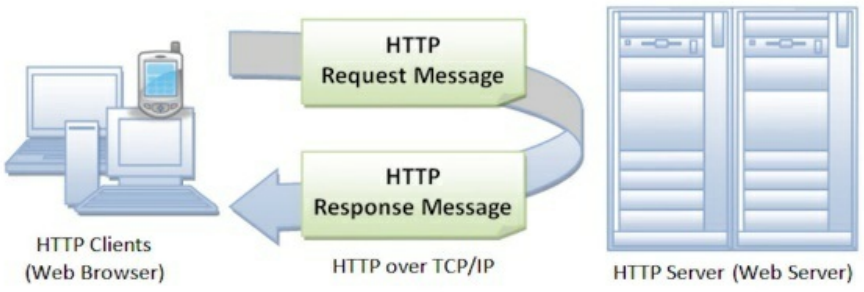
First, we have to import the wifi library to use the various inbuilt functions associated with the WiFi library Here we initialize the Network credentials and assign them to constants The first one is the SSID or Service Set Identifier It is nothing but your network name The next one is the password for your network Now let’s look at the setup code Here we are setting the Serial Baud rate as 115200 Here we are printing the SSID to the Serial monitor

```
void setup() {
 Serial.begin(115200);      // set the LED pin mode

    delay(10);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
```

Next, we have the WiFi Begin function which is used to connect to the network with the SSID and password specified earlier in the code This function will return WL Connected when it is connected to the network, else it will return WL Idle status Now, in the while loop, we are checking the condition for the current connection status If the connection status is not connected, it will printout dots until the Wifi connection status becomes connected Once connected to the network, we will printout this message and the IP address assigned to the ESP32



Now after uploading open up the serial monitor If you do not see anything, press the reset button on the board Now you can see that our ESP32 has successfully connected to the Wifi Network and you can also see the Local IP address Now that we have looked at how to connect to the Wi-Fi network let’s take it a step further Let’s see how we can blink the onboard LED on the Thing from our browser



Here the Thing will be acting as a Server to which we will be placing an HTTP request from the browser which will be the client First, include the WiFi library Then put your WiFi credentials here Now, this is used to create a server that will listen for the incoming connections on Port 80 is the default port for HTTP

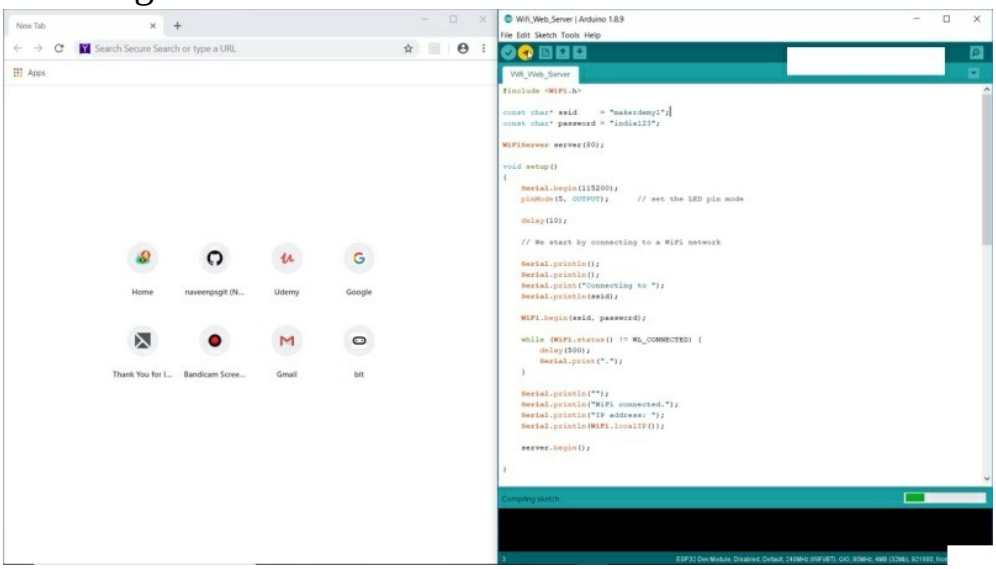
Wificlient

```
#include <WiFi.h>

const char* ssid     = "makerdemy1";
const char* password = "india123";

WiFiServer server(80);

void setup()
{
    Serial.begin(115200);
    pinMode(5, OUTPUT);      // set the LED pin mode

    delay(10);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
```

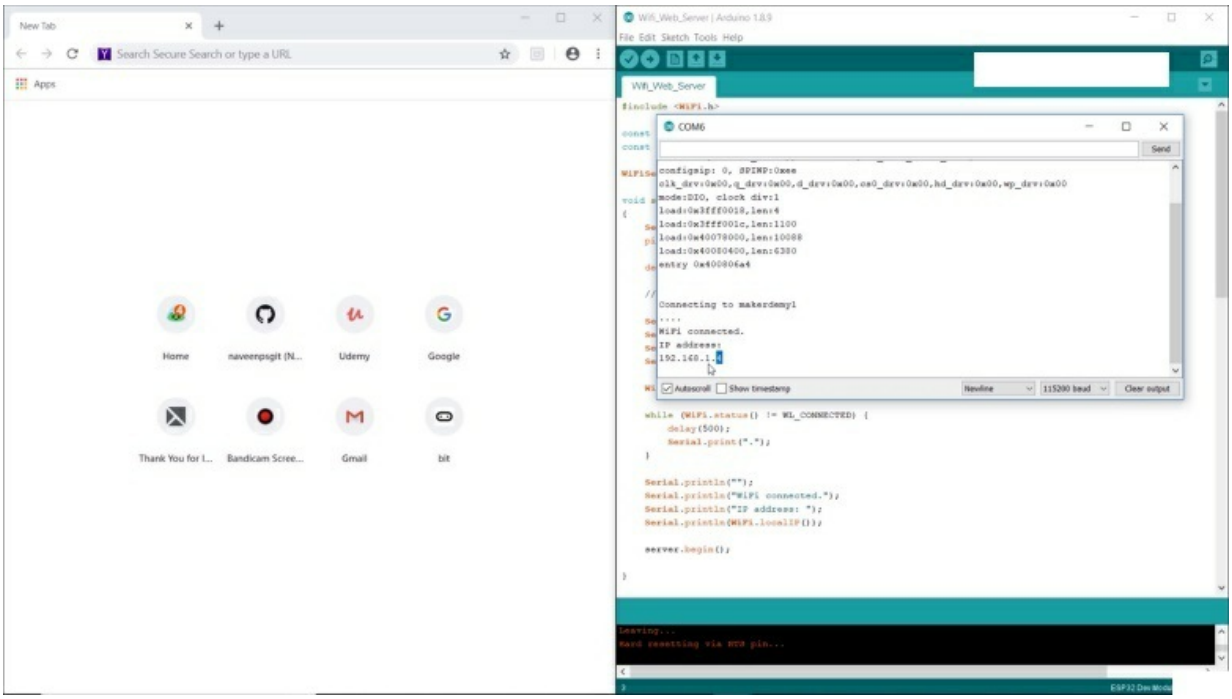
We are using this port to communicate with the Web client in this case being our browser Now the Setup code is pretty much the same which we used for the WiFi connection There are some additions though Here we are setting the onboard LED which is PIN 5 as the output

```
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    server.begin();

}
```

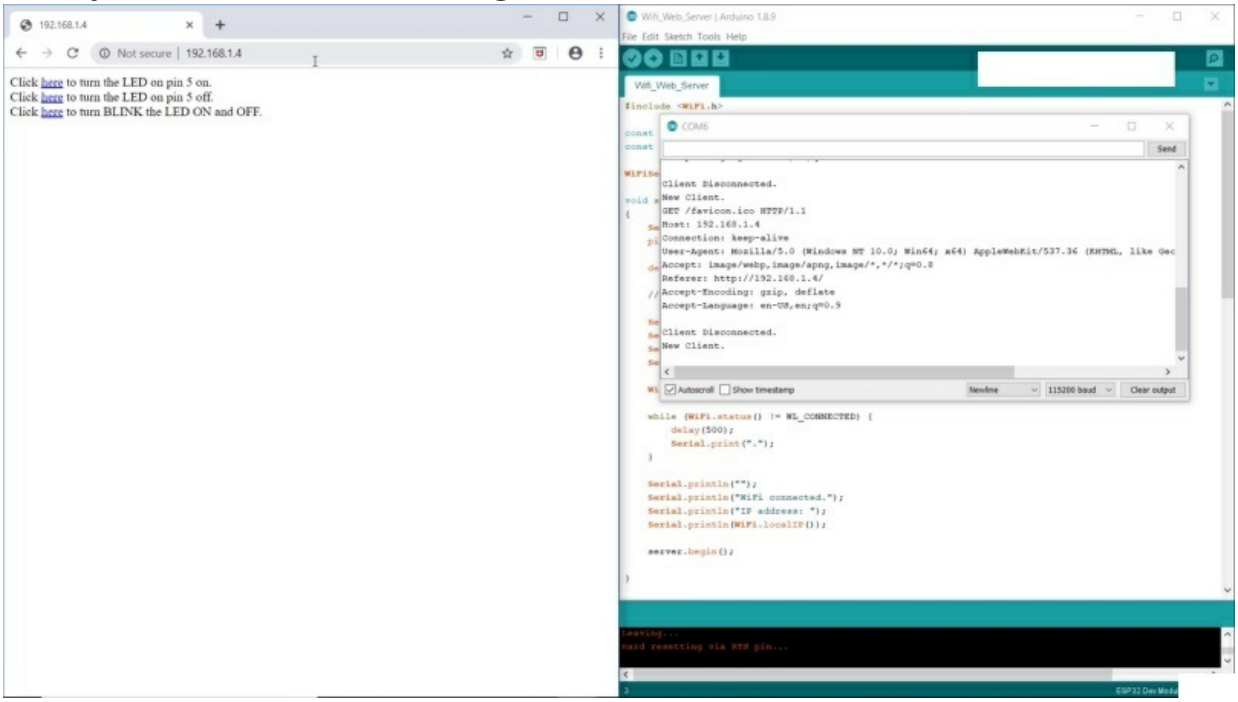
here we use the server begin function to tell the server to start listening for incoming connections



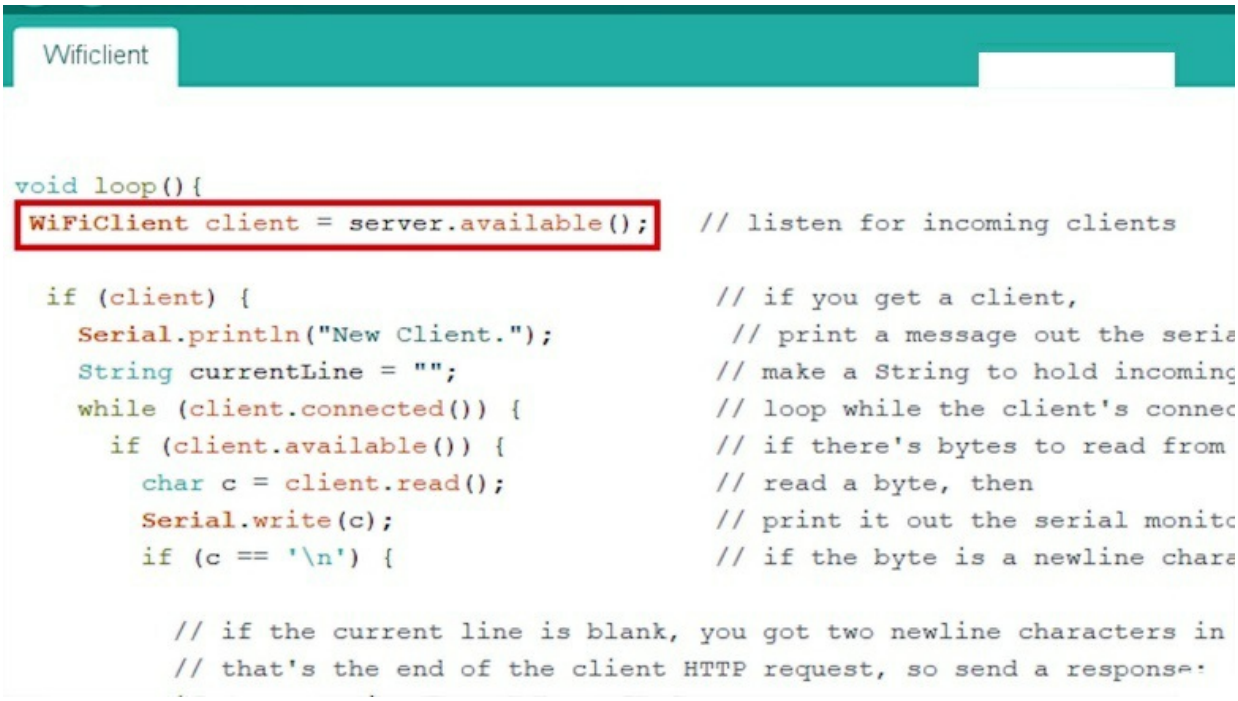
Now to understand the loop code let’s first upload the code and open up the Serial Monitor You need to ensure both the Thing and your computer is connected to the same WiFi network



Here you can see that the Thing has connected to the Wifi network



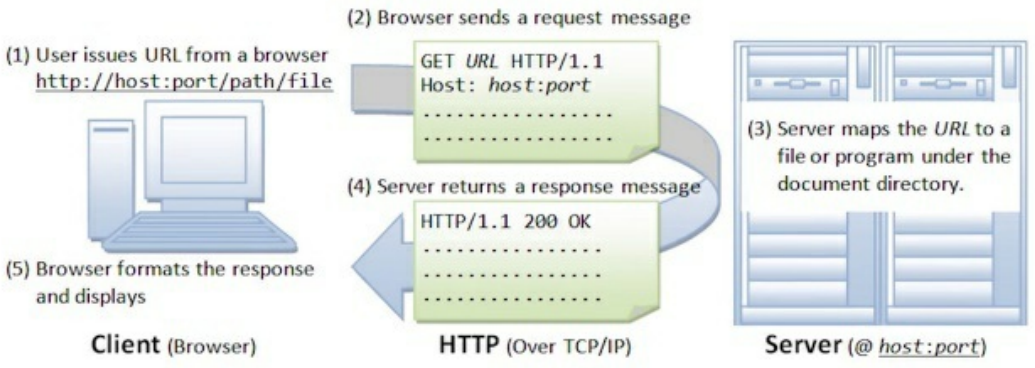
Now open up your browser, copy the IP address of the Thing and paste it in the address bar What you are doing here is that you are making an HTTP request to the server, which is running on the Thing



Now let’s go back to the loop code The Server available function returns a client object for the client which is connected to the server and has data to be read Here, by entering the IP address in the browser address bar we made an HTTP request to the server Now if the client has placed the HTTP request, the if condition will get executed Here we print this message to specify that a new client has connected to the server successfully Now we initialize an empty string variable current line to hold the incoming data from the client Since the client is connected the while loop will execute Now if there are bytes available on the client-side for reading, we will read the bytes using client read and store it one by one in the character c If we get the character C as a newline character, it means that we have just received an HTTP request from the browser

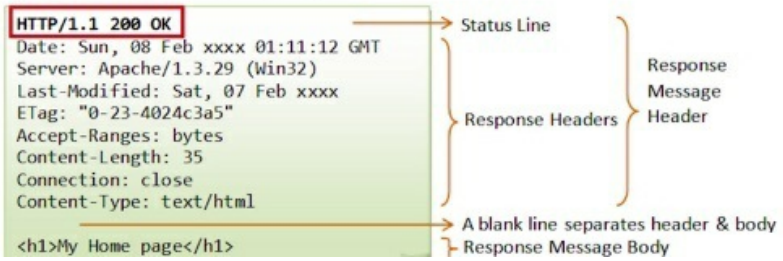
```
while (client.connected()) {                    // loop while the client's connec
  if (client.available()) {                     // if there's bytes to read from
    char c = client.read();                     // read a byte, then
    Serial.write(c);                            // print it out the serial monito
    if (c == '\n') {                            // if the byte is a newline chara

      // if the current line is blank, you got two newline characters in
      // that's the end of the client HTTP request, so send a response:
      if (currentLine.length() == 0) {
        // HTTP headers always start with a response code (e.g. HTTP/1.1
        // and a content-type so the client knows what's coming, then a b
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println();

        // the content of the HTTP response follows the header:
        client.print("Click <a href=\"/High\">here</a> to turn the LED or
        client.print("Click <a href=\"/Low\">here</a> to turn the LED on
        client.print("Click <a href=\"/blink\">here</a> to turn BLINK
```

Now here we will check, if the length of the string current line is zero it means that our HTTP request received has ended



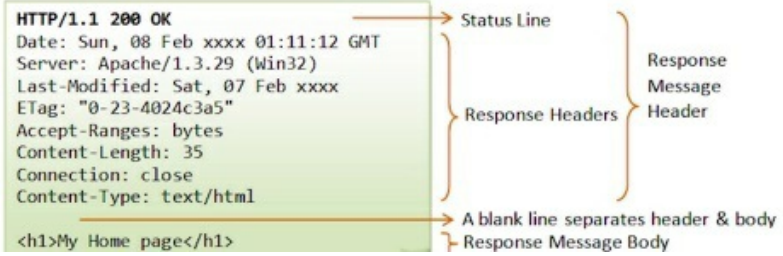
Now we can send an HTTP response to the client along with data in the form of HTML so that we can view it in our browser The HTTP header will always start with a certain response code This consists of the HTTP version being used and the status code

```
if (currentLine.length() == 0) {
  // HTTP headers always start with a response code (e.g. HTTP
  // and a content-type so the client knows what's coming, then a blank line:
  client.println("HTTP/1.1 200 OK");
  client.println("Content-type:text/html");
  client.println();

  // the content of the HTTP response follows the header:
  client.print("Click <a href=\"/High\">here</a> to turn the LED on pin 5 on.<
  client.print("Click <a href=\"/Low\">here</a> to turn the LED on pin 5 off.<
  client.print("Click <a href=\"/blink\">here</a> to turn BLINK the LED ON and
  // The HTTP response ends
  client.println();
  // break out of the while
  break;
} else {    // if you got a
  currentLine = "";
}
```



Here the status code we are sending is 200 OK, which means that the HTTP request made by the client is successful Next, the HTTP header contains the content type which we would be sending, here since we want to view it in the browser we will be using HTML type text This will ensure that the browser renders the text as HTML

```
  client.println("Content-type:text/html");
  client.println();

  // the content of the HTTP response follows the header:
  client.print("Click <a href=\"/High\">here</a> to turn the LED on pin 5 on.<
  client.print("Click <a href=\"/Low\">here</a> to turn the LED on pin 5 off.<
  client.print("Click <a href=\"/blink\">here</a> to turn BLINK the LED ON and
  // The HTTP response ends with another blank line:
  client.println();
  // break out of the while loop:
  break;
} else {    // if you got a newline, then clear currentLine:
  currentLine = "";
}
else if (c != '\r') {  // if
currentLine += c;      // ac
```



Now we have to leave a blank line to signify the end of the header

```
  // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
  // and a content-type so the client knows what's coming, then a blank
  client.println("HTTP/1.1 200 OK");
  client.println("Content-type:text/html");
  client.println();

  // the content of the HTTP response follows the header:
  client.print("Click <a href=\"/High\">here</a> to turn the LED on pin 5 on.<br>");
  client.print("Click <a href=\"/Low\">here</a> to turn the LED on pin 5 off.<br>");
  client.print("Click <a href=\"/blink\">here</a> to turn BLINK the LED ON and OFF.<br>");
  // The HTTP response ends with another blank line:
  client.println();
  // break out of the while loop:
  break;
} else {    // if you got a newline, then clear currentLine:
  currentLine = "";
}
else if (c != '\r') {  // if you got anything else but a carriage return character,
currentLine += c;      // add it to the end of the currentLine
```

This part here is for the actual content we would be sending in the HTTP response Now in the client print function , you need to specify the text which you want the browser to display The word here has been linked to a URLThe here attribute is used to specify the link’s destination We print three lines here, one for turning on the LED, one for turning off the LED and the last one for turning the LED on and off Each one has been assigned a specific resource name HIGH, LOW and Blink Now, the final HTTP response ends with a blank line, after which we break out of the loop If the current line variable length is not zero we will clear the current line variable This is to ensure that the initial HTTP request made did not contain any data Now till here, the server has sent the HTTP response to the client and the browser or the client has rendered the HTML data which it received in the HTTP response After the Initial HTTP request to get the HTML data we now need to place a get request from the following links to access the resources we had created To access the resource, we need to know whether the request placed was for HIGH, LOW or blink To this, we append the current line variable with the new incoming characters in the HTTP request

```
        currentLine = "";
      }
    } else if (c != '\r') {  // if you got anything else but a carriage return
      currentLine += c;      // add it to the end of the currentLine
    }

    // Check to see if the client request was "GET /H" or "GET /L":
    if (currentLine.endsWith("GET /High")) {
      digitalWrite(5, HIGH);   // GET /H turns the LED on
    }
    if (currentLine.endsWith("GET /Low")) {
      digitalWrite(5, LOW);                // GET /L turns the LED off
    }
    if (currentLine.endsWith("GET /blink")) {
      digitalWrite(5, HIGH);
      delay(500);
      digitalWrite(5, LOW);
    }
  }
```

When we press any of the Links, a GET request is made to access the resource HIGH, LOW, or Blink In the If condition here, we check if the clients get request ends with the following using the Ends with function If the string ends with this, it means that the get request is for High and hence we switch on the LED If the end of the string contains this, it means that the get request was made to access the Low resource hence we switch off the LED This is similarly done for the Blink resource to turn the LED ON and OFF

```
    // Check to see if the client request was "GET /H" or "GET /L":
    if (currentLine.endsWith("GET /High")) {
      digitalWrite(5, HIGH);   // GET /H turns the LED on
    }
    if (currentLine.endsWith("GET /Low")) {
      digitalWrite(5, LOW);                // GET /L turns the LED off
    }
    if (currentLine.endsWith("GET /blink")) {
      digitalWrite(5, HIGH);
      delay(500);
      digitalWrite(5, LOW);
    }
  }
}
// close the connection:
client.stop();
Serial.println("Client Disconnected.");
```

Finally, we disconnect the client to stop the connection

Now let's go back to the browser to see this in action If you click on here, you will see the LED switch on Similarly, the LED will switch off If you click here.

UNDERSTANDING RSSI AND MEASURING SIGNAL STRENGTH

We will learn the following What is RSSI and why it is useful Reading the RSSI values of the Access point to which the Thing is connected

Let's talk about RSSI first RSSI or Received Signal Strength Indicator as the name suggests, is used to measure the received signal strength to which a device is connected to It is a measurement of how well your device can hear a signal from the access point or router it is connected to The RSSI value can be used for comparing the signal quality in the same network It can also be useful for comparing signal quality among various Wi-Fi networks The RSSI value usually varies in range, depending on the WiFi adapter being used in the receiving device So for example, some WiFi adapters can have an RSSI range from 0 to 60 while some WiFi adapters' RSSI value can range from 0 to 255 This can create confusion while comparing the received signal strength

for different devices

Hence the RSSI values are usually converted to a standard unit kn as dBm We will look at why we do this now The received signal power is usually in milliwatts, which can get to even smaller values if the signal strength further drops This makes it difficult to read and comprehend Hence we need to convert it to a unit which can be easily read and understood

dBm or decibels relative to a milliwatt

$$S_{dBm} = 10 \log_{10} \frac{P}{1\,mW}$$

$$0_{dBm} = 10 \log_{10} \frac{1\,mW}{1\,mW}$$

That's why we use dBm This is the formula for mW to dBm conversion dBm or decibels relative to a milliwatt sets the reference power value as 1mW, to which all power values are compared Hence, here 1 mW power has a signal level of 0 dBm This solves the problem for signal strength comparison among different WiFi adapters 1 dBm is equal to 1/1000th of a watt or 1mW As you can see, such a small value is converted to a larger value This solves the readability problem, which makes the values more convenient to handle and comprehend

Signals stronger than 1 mW have positive dBm values, whereas signals weaker than 1 mW have negative dBm values Normally received signal power in WiFi networks is less than 1 mW Hence the dBm values usually range from 0 dBm to - 100 dBm

RSSI §

```
#include <WiFi.h>

const char* ssid     = "makerdemy1";
const char* password = "india123";

void setup()
{
  Serial.begin(115200);      // set the LED pin mode

    delay(10);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
```

Now let's look at the code to measure the RSSI value of the Wi-Fi network we are connected to The code up until here is pretty much the same You need

to enter the Wi-Fi credentials of the network for which you wish to measure the RSSI

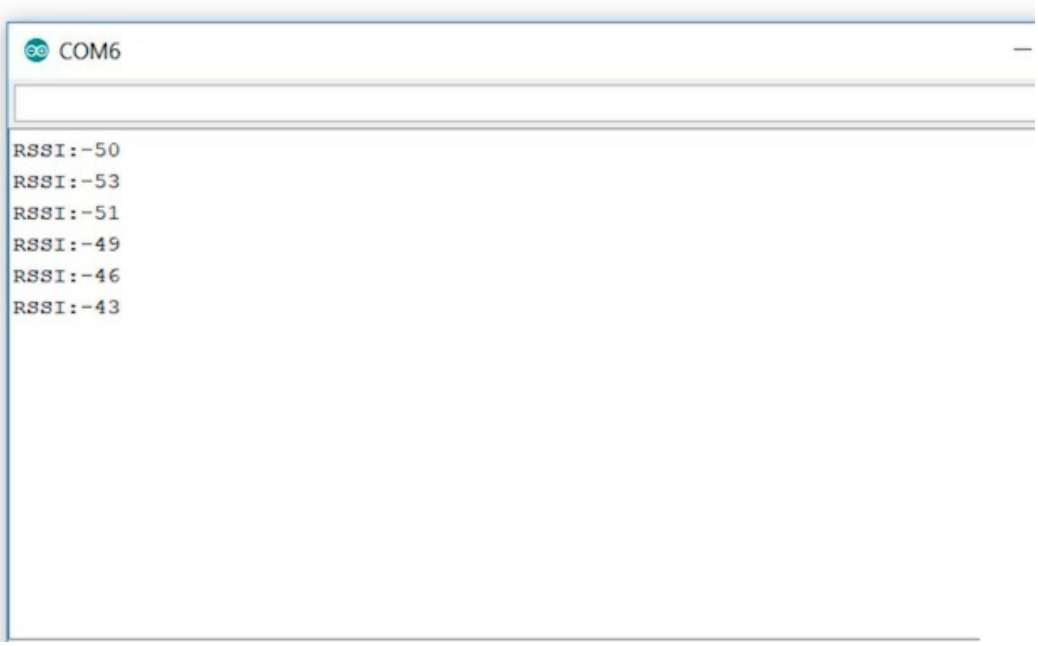
```
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

}

void loop () {

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("RSSI:");
```

Now the Wifi library contains a useful function called WiFI RSSI This function gets the RSSI value of the connection to the Router The value returned is a long type integer value which is stored in this variable We then print this value to the serial monitor every half a second

```
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

}

void loop () {

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("RSSI:");
  Serial.println(rssi);
  delay(500);
}
```

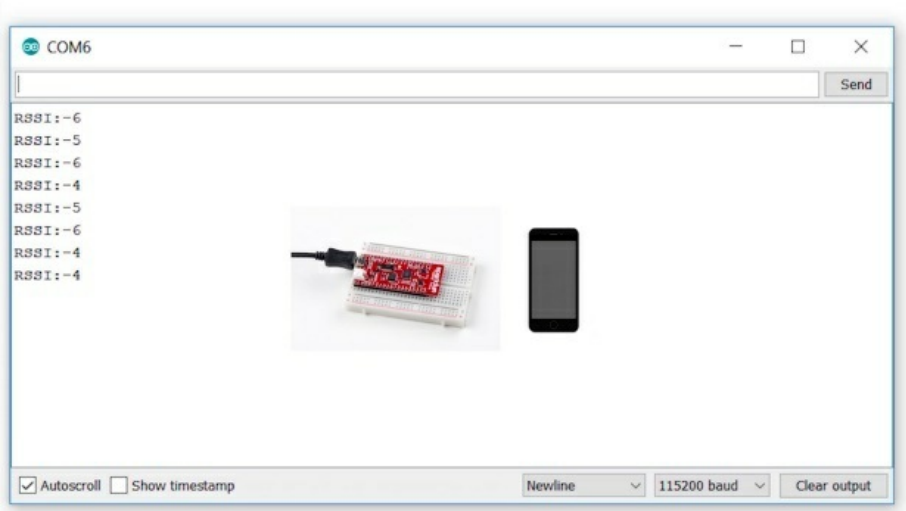
Now if you upload the code and open up the Serial Monitor, you will see the

RSSI values



Here the signal strength for our Wi-Fi connection to the router is pretty good
Usually, RSSI values below -90 are considered to be unusable

Now let's see what happens when we try to connect the Thing to a WiFi hotspot on your phone Enter the credentials of the Wi-Fi Hotspot here and again upload and open up the Serial Monitor



Also, let's keep the phone near to the Thing You can see that the RSSI value is significantly better

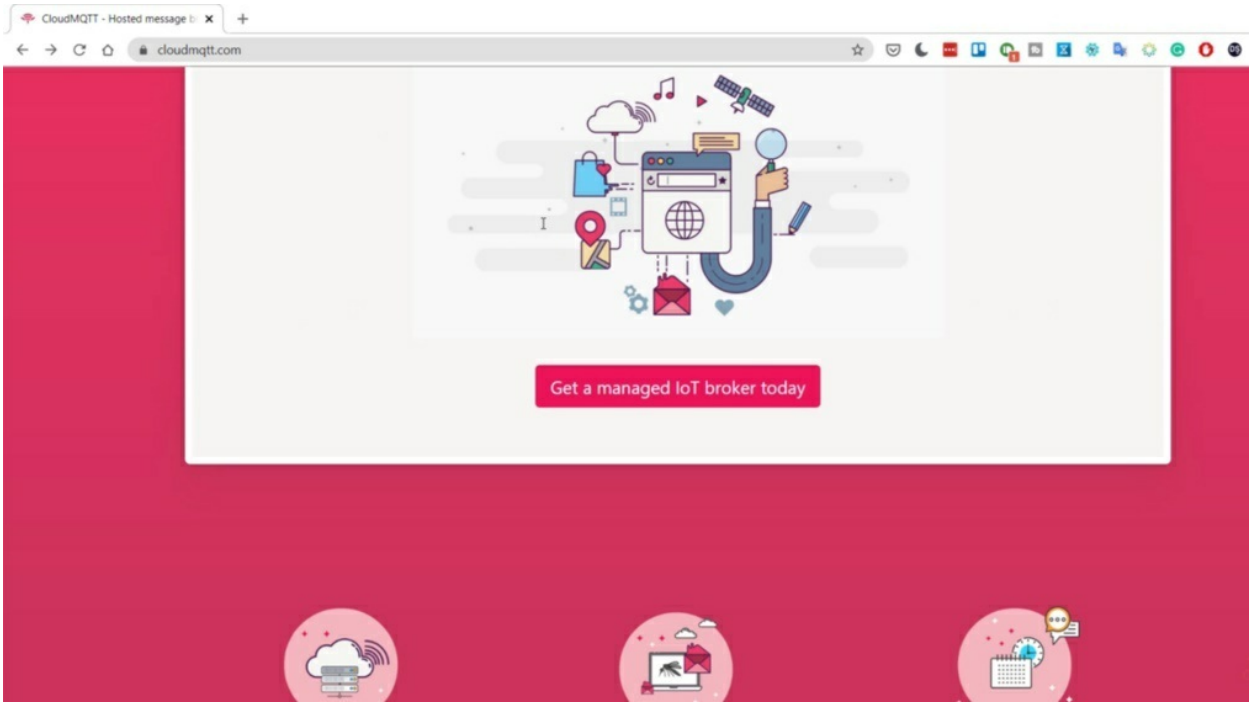
Now let us move the phone away from the Thing slowly, you will see that the

RSSI value starts to decrease

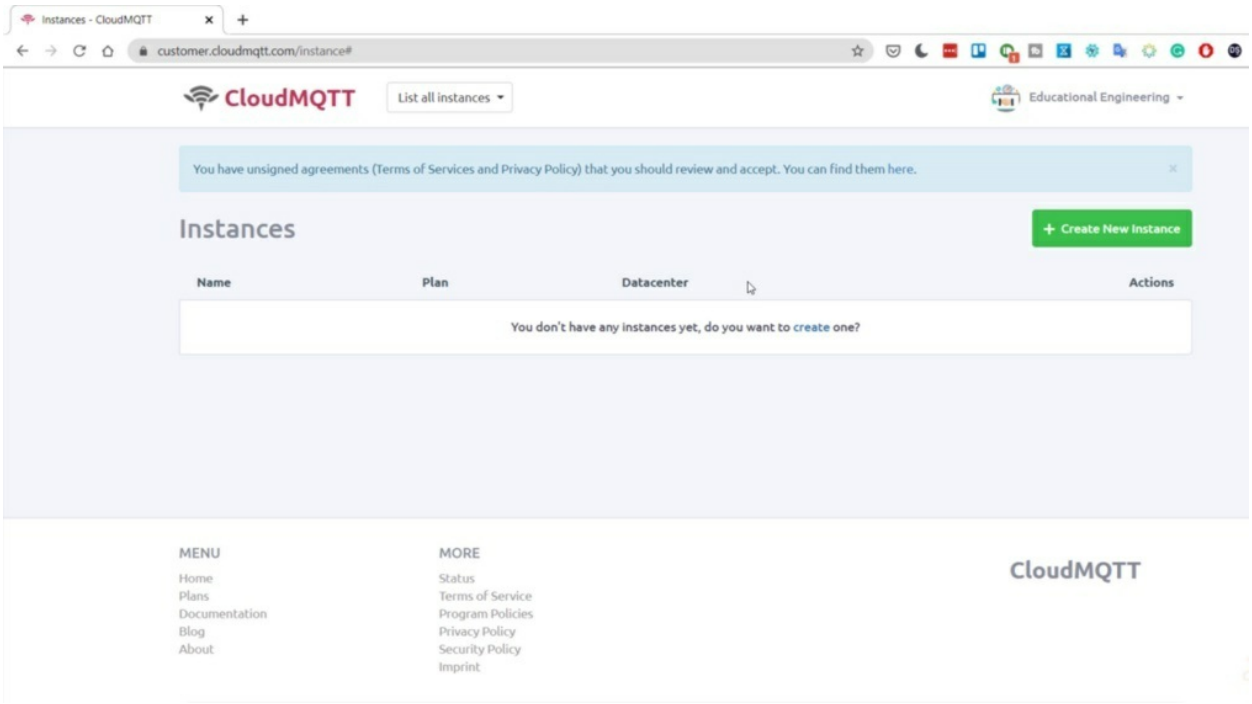
Thus, distance from the access point or router is also one of the reasons which affect RSSI Farther from the router the Thing is, the lesser will be the RSSI value and hence lower the signal strength Factors like the number of client devices connected to the network, location of the access point, etc also affect the RSSI value In this we learned about what RSSI is and how it can be useful to determine the Wi-Fi signal strength We also learned about reading the RSSI values on the Thing In this section, we covered the following Understanding Bluetooth Low Energy and Wi-Fi Establishing BLE connection with the Thing BLE server and client communication Establishing Wi-Fi connection.

CREATE MQTT SERVER ACCOUNT

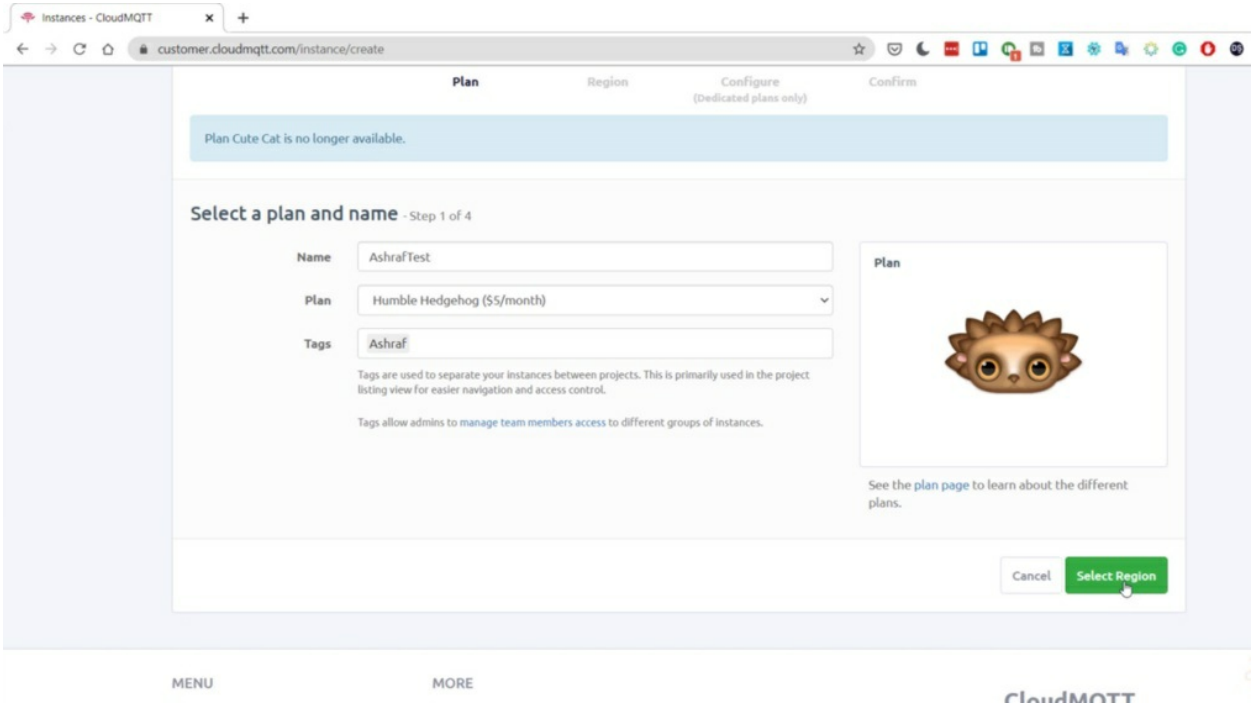
Now let's create a new account in the cloud. I'm Hugh TTD dot com site.



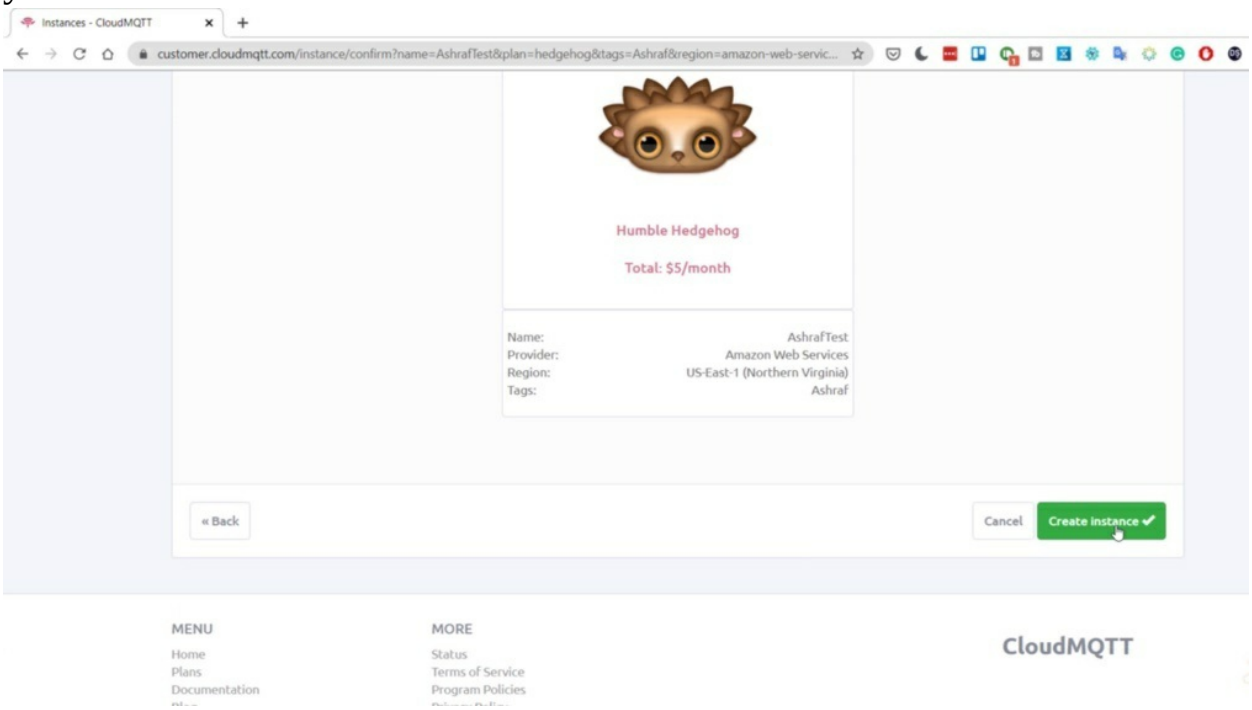
As you can see, they have a lot of plans that you can choose from depending on your project. But let's slogan using our e-mail account. Without creating a new account, this will spare us a step.



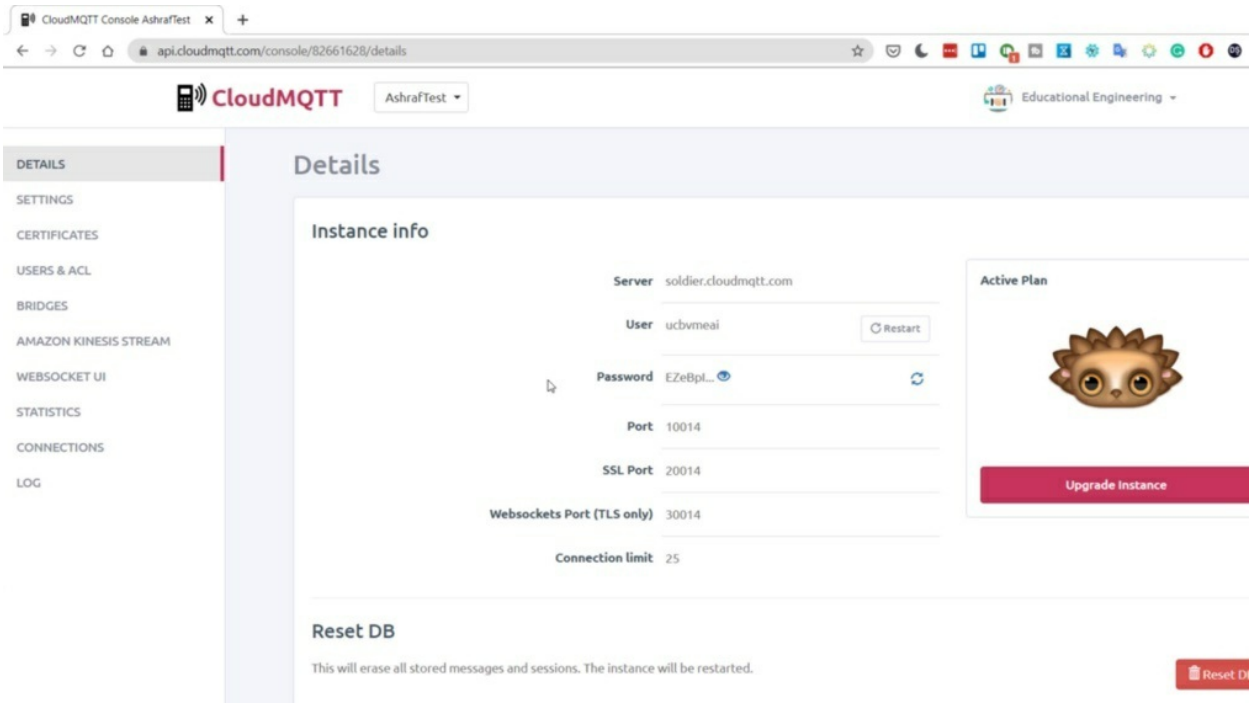
I'll choose one of my accounts, my [REMOVED] accounts, to sign in to this website. Now click create a new instance, the green button on the right side of the screen. Now, give your instance a name and choose your plan and give it a tag. They used to have a free plan, but now they have a five-dollar-per-month plan that you can try. They have a trial period.



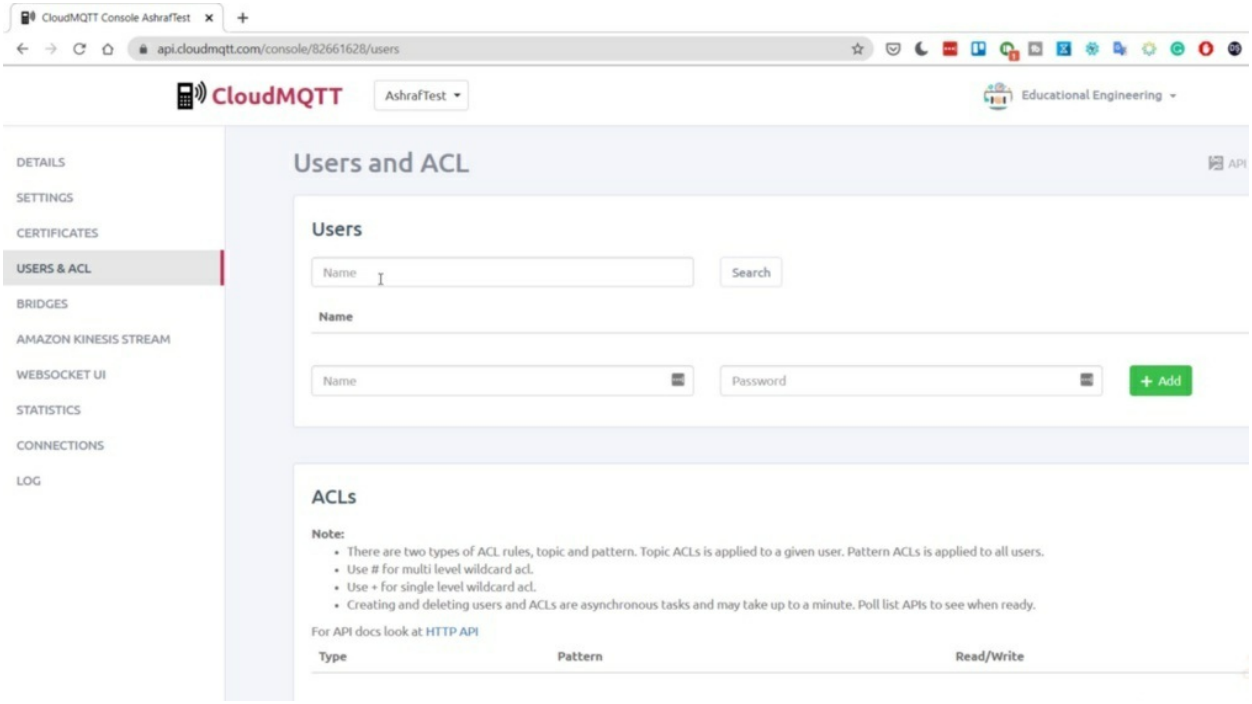
And you can continue their support to get it. Select region after creating this. You can choose the data center closer or the closest one to you, depending on your location. I will leave it to the default. Then click review.



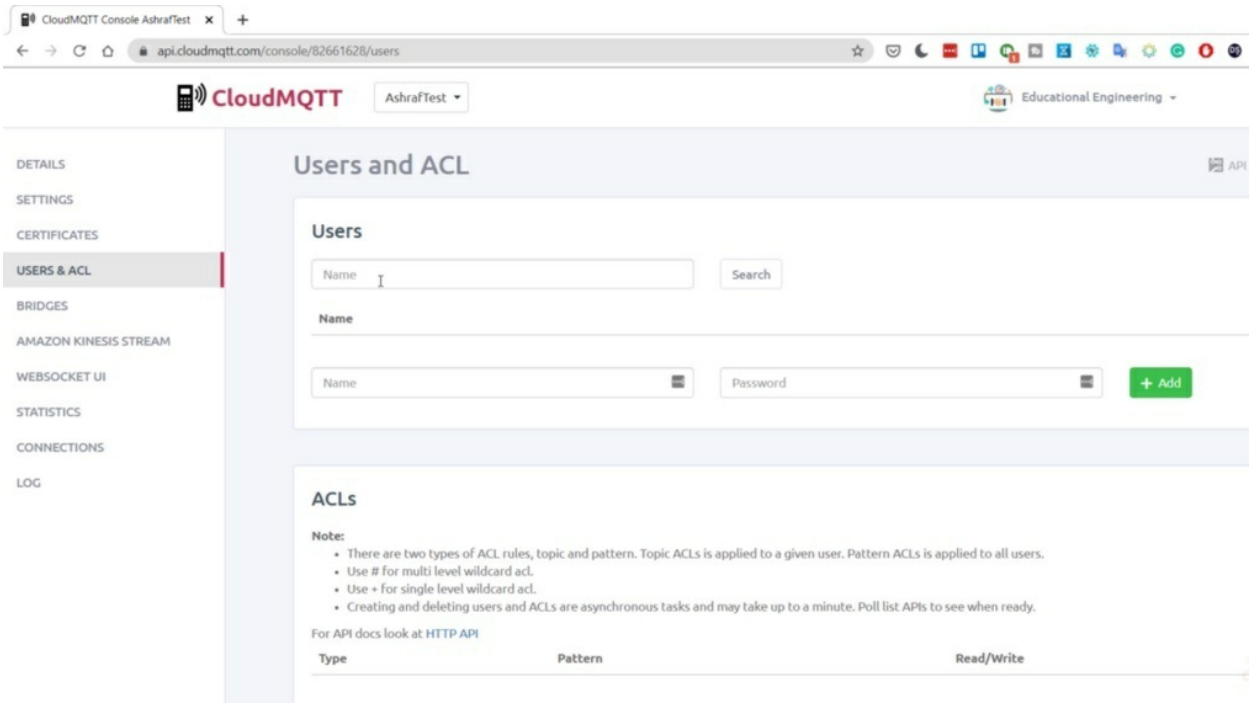
At this point, you only have to click, create an instance and you'll have your stands up and ready. This is it. It's called Ashcroft Test. Now, a few clicked on despite his tone. You can see its proprieties click once. As you can see here,



we have the server on the post, and that is this old post.

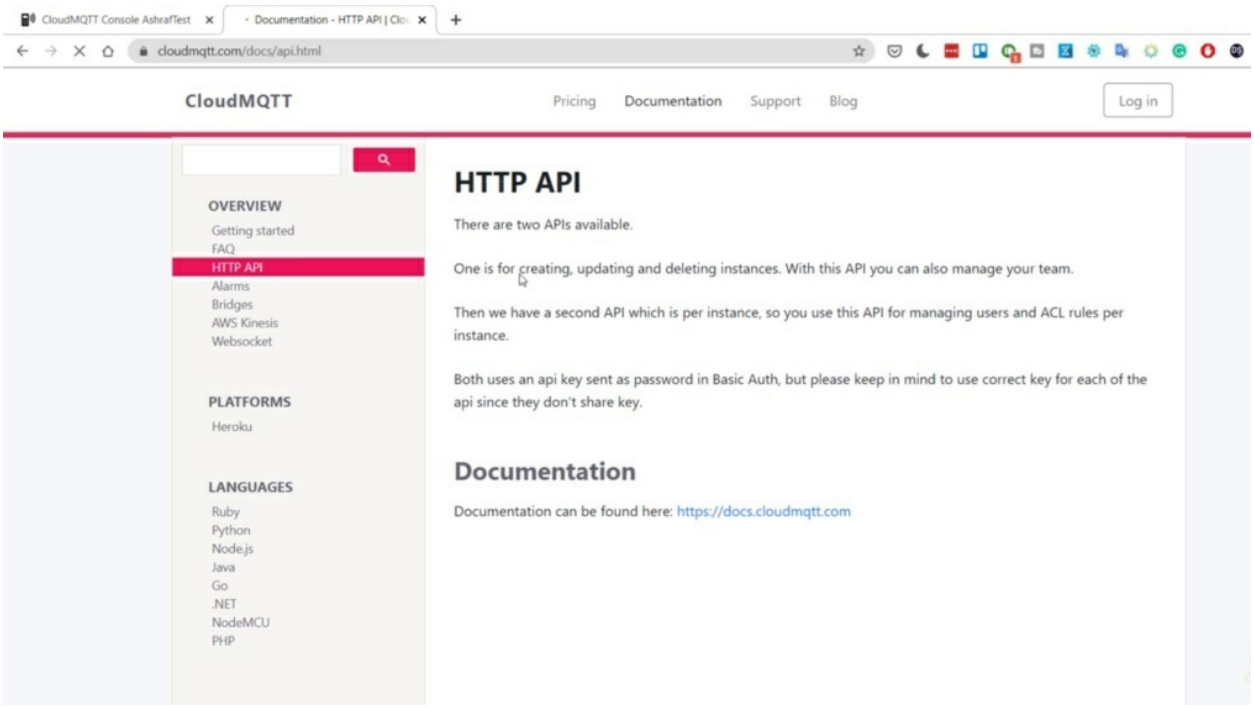


Now go to users and ACL select and aim for your user. And an M. A password for the user. I will choose the same thing. Ah, Greenall E. S. P. Then click, add. Now you have a user connected to your instance. Now we need to set are easy ls. You can't select. S l. By topic or by Petr?



Now, the topic as the LS is applied to a given user and we already have a user that we want to apply. S he owes to. So simply click a topic. Choose your old user and make sure that valid and right acts are selected. Oh, we forgot to add a name.

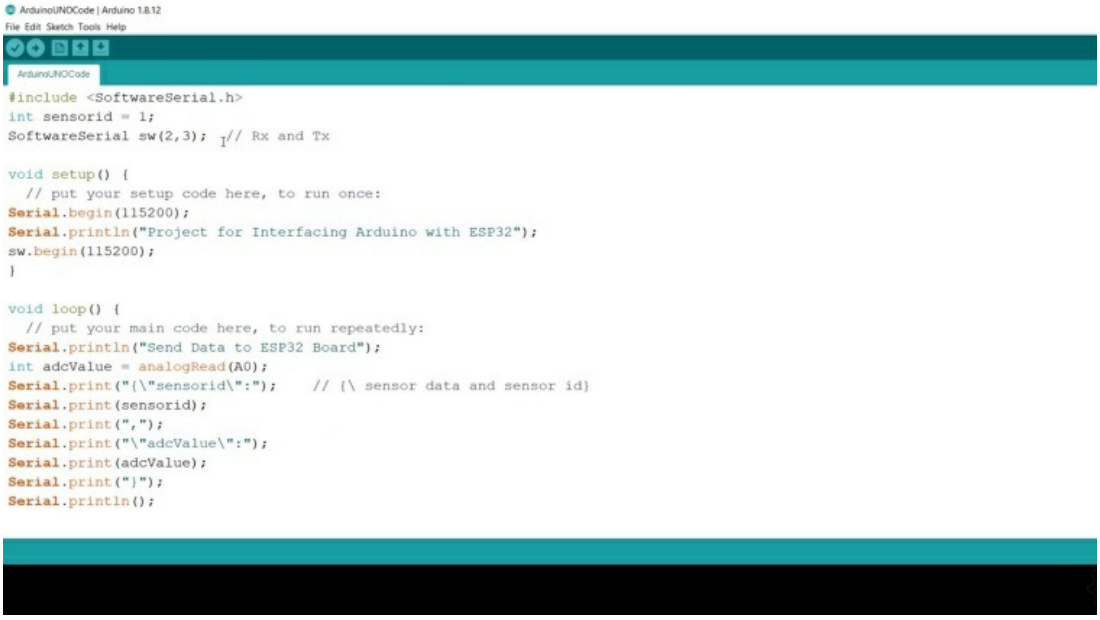
Okay, select those again. Add that pattern name. Make sure that the read online is selected. Then click add. Now we have the user connected. And paths lead to untried privileges of our innocence.



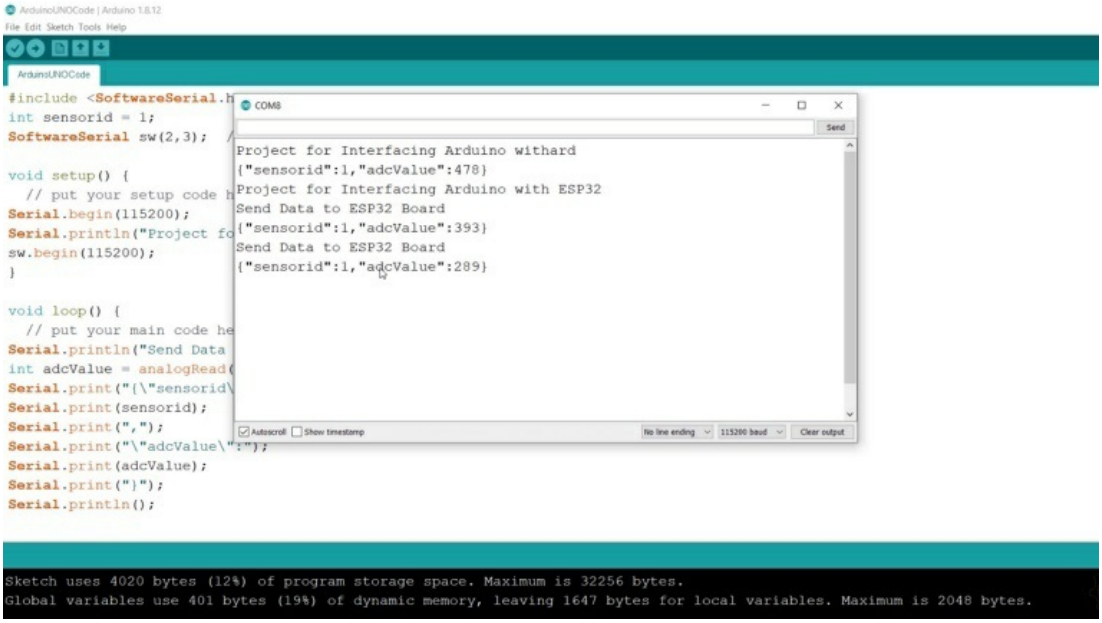
Now you can see their API for further information on this, but that's it. This is how to create an instance.

UPLOAD CODE TO ARDUINO AND TEST IT

Now, the first thing that we need to do is upload the Arduino code to the Arduino board, make sure that you are selecting doing Ono.



And make sure that you have the right balls lifted, then upload your code and I will tell you how to know if what you are doing is correct or not. And I mean it. Once the codes are uploaded tor arguing, done uploading.

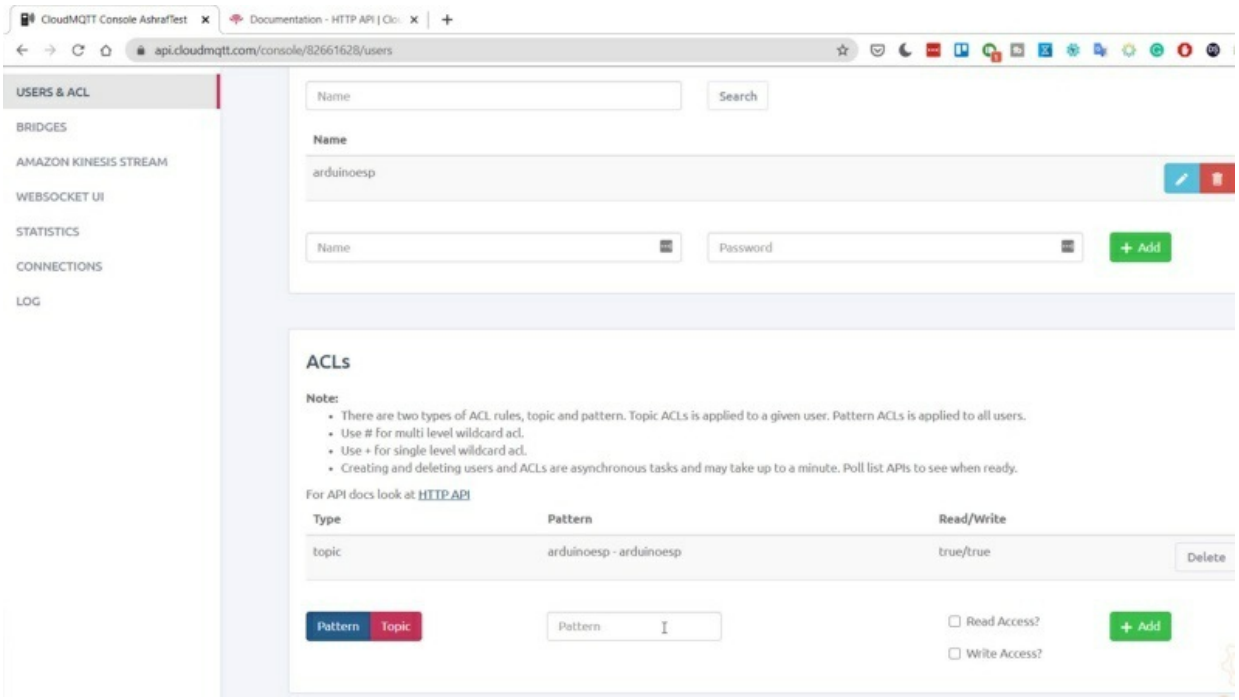


Now go and click that city hall monitor. If you are getting these readings, as you can see, sense of I. D. and ADC value and this. Format, then your Arduino is good to go and it's ready to be connected with or E. S. P board.

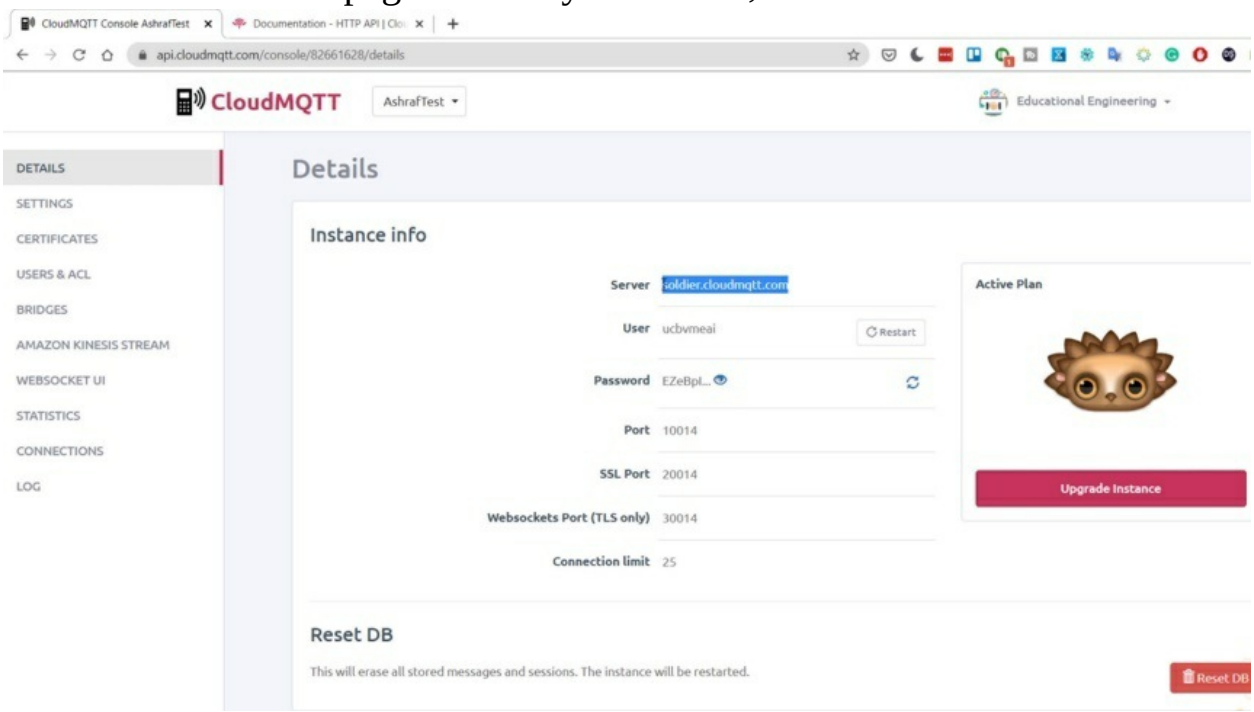
EDIT CODE AND UPLOAD IT TO ESP32 THEN TEST IT

Going to edit or E. S. P 32 cored and upload it to our E. S. P board. Now, to do this or what you need to do is go to the M.

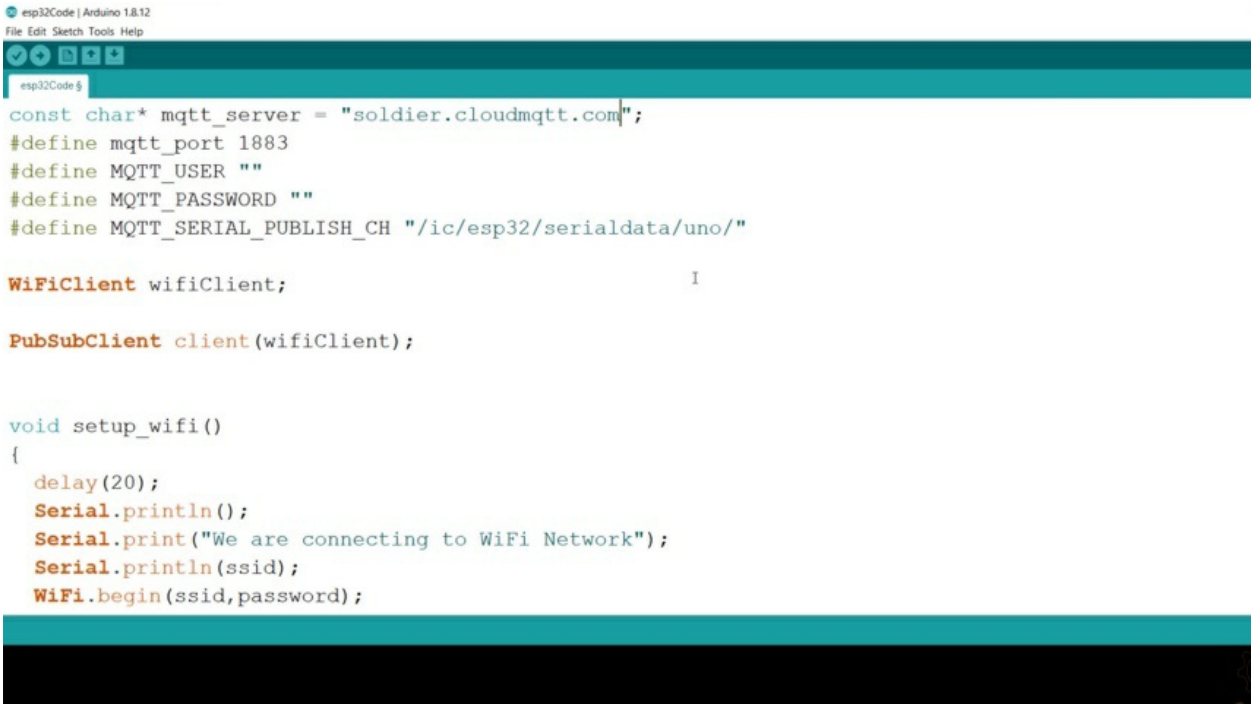
TTR website knows we have created our account. And here, as you can see. We have. For the requested data, we have the user name and password for our assistance.



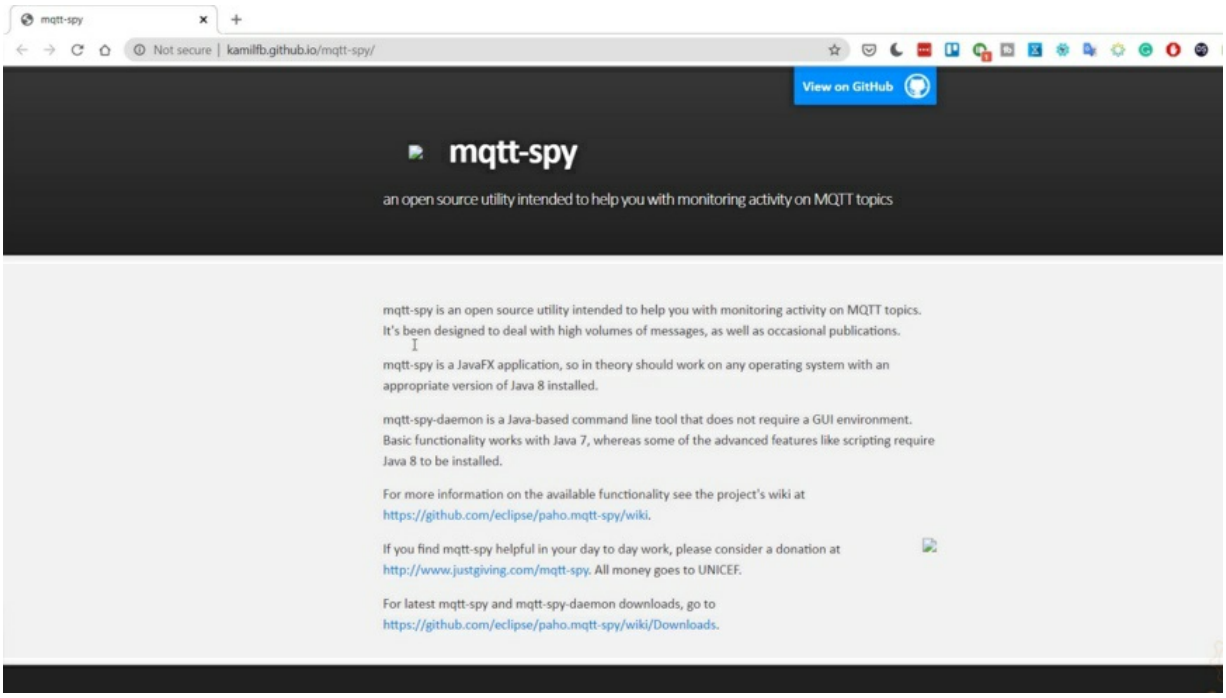
Go back to the Tales page. And as you can see, this is our server.



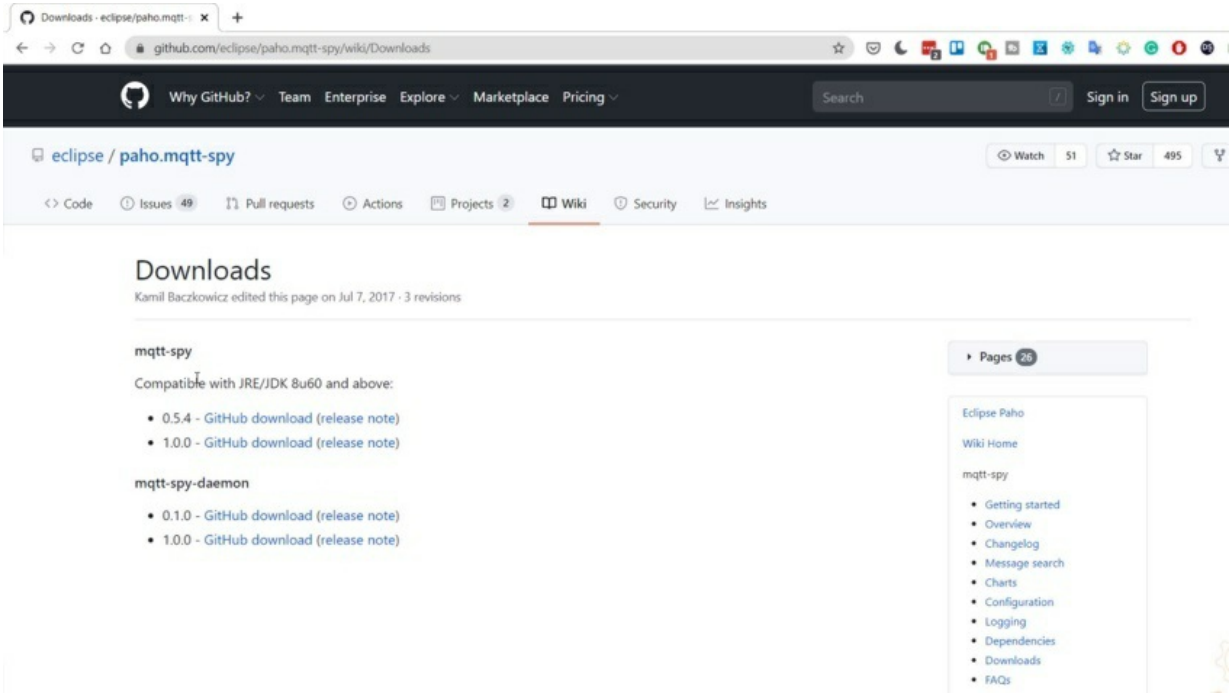
Now, if you want to hear. This is our server link.



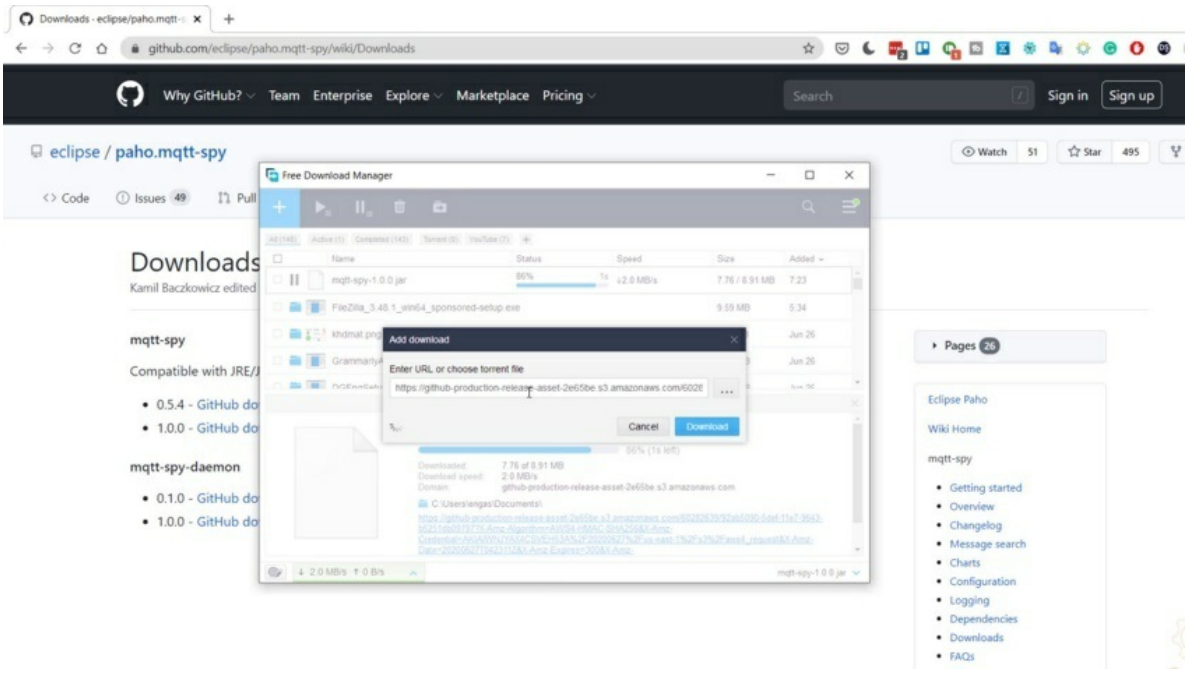
Now, the next step is the post. Make sure that you have the right post here. We have this post, which is copy-paste the post. We have a user on password. So we need to select a username and password and you can invent a little username and password for it. Your distance from this menu. And we already know that we have this username. And we cut light and password to your password that we use to create our account. That's it. And we have this as the bubbles channel.



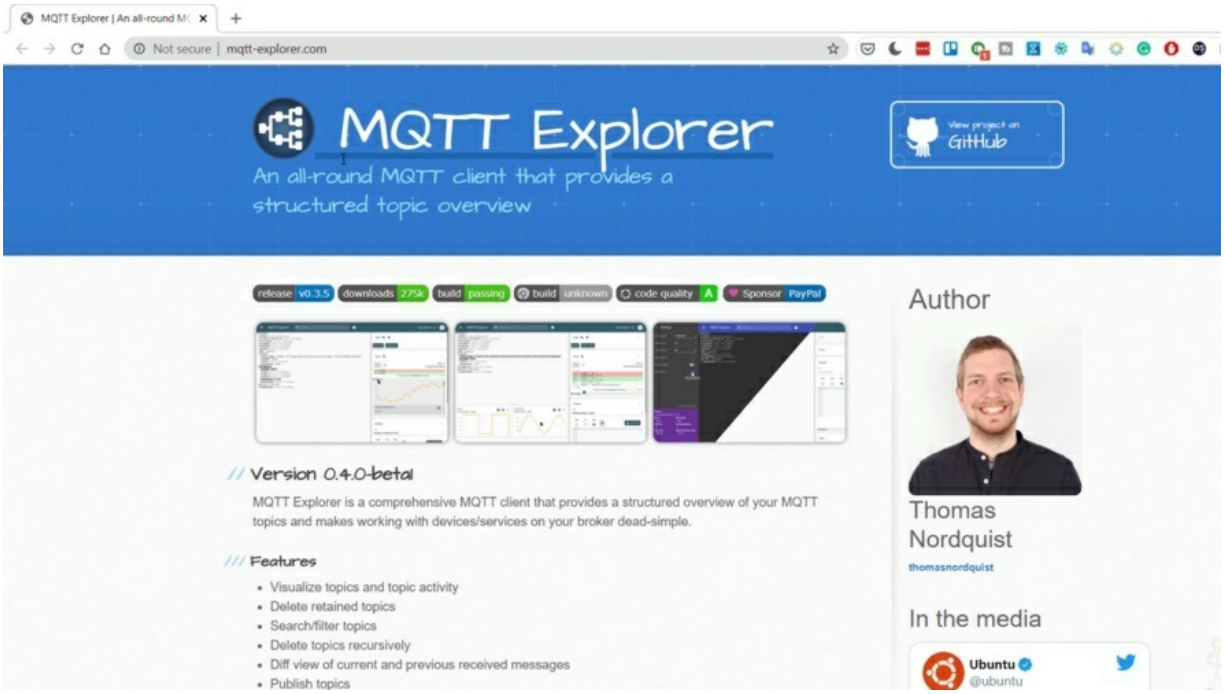
Now, to make sure that that is that you have entitled are working just fine. You can try. Titi Spy, which is an open-source utility intended to help you with monitoring activity on Kutty topics, and you can easily dload it from GitHub.



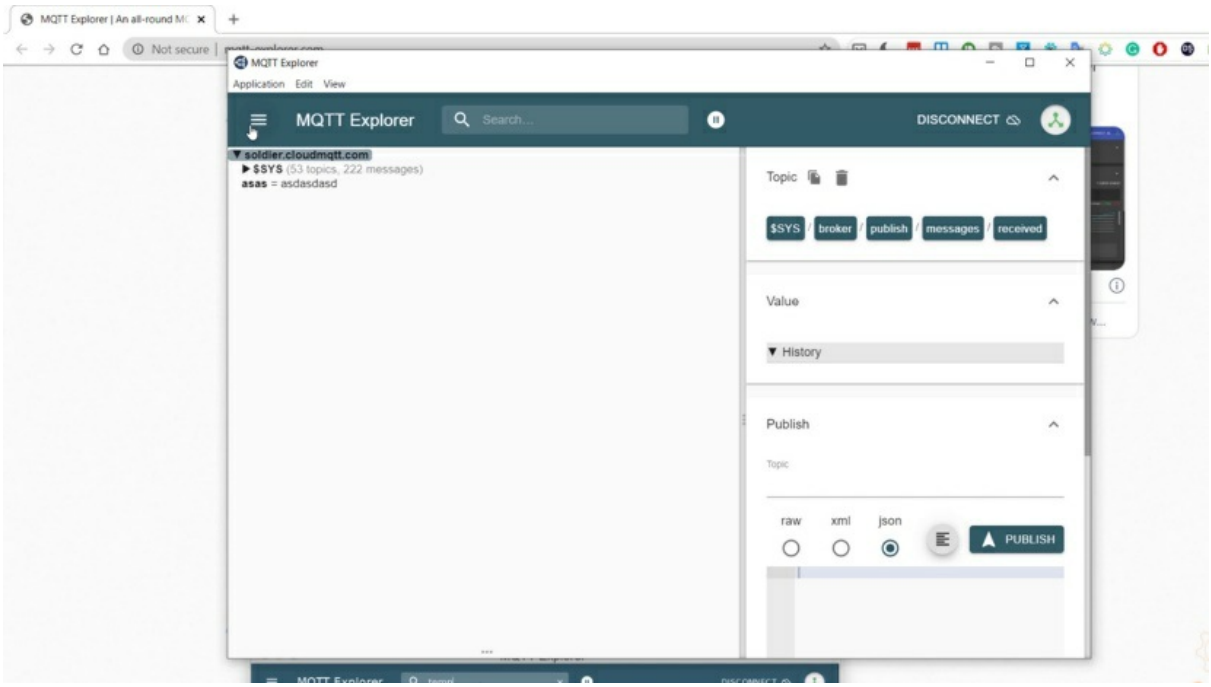
You just need, like computers spy. And here are the available versions. You can dload any of these.



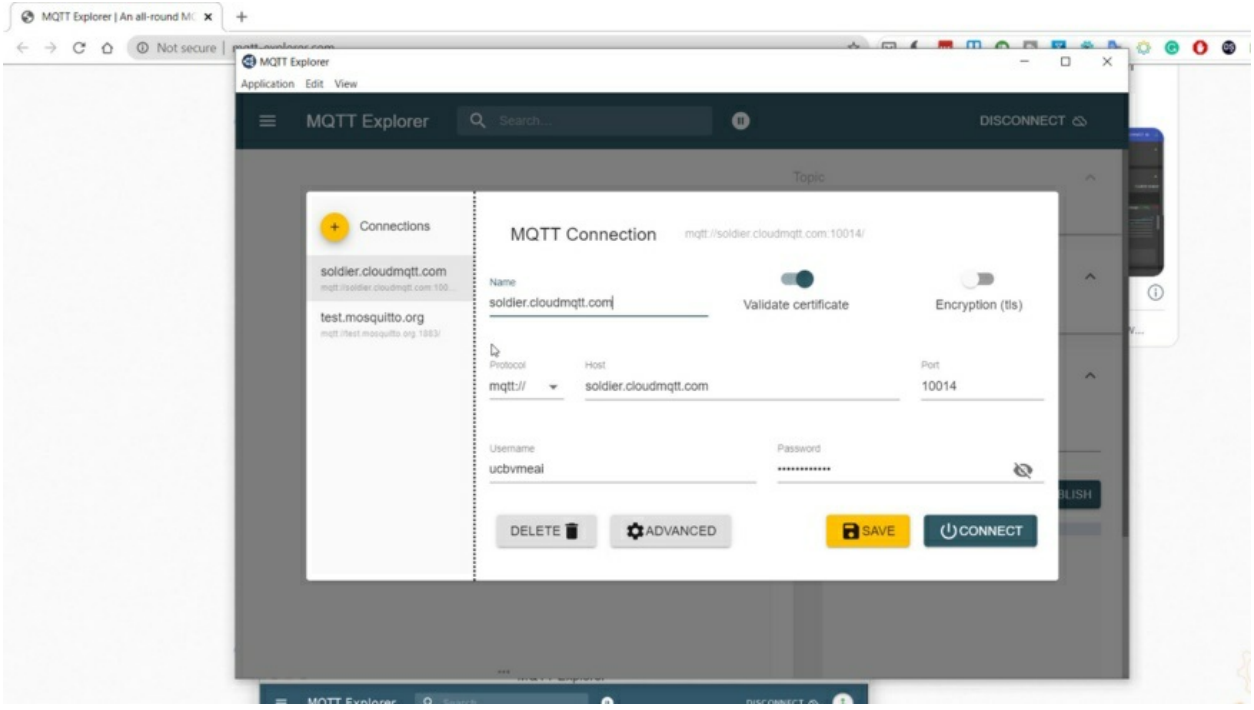
Let's get it. Let's give this one all this one. And this is a job find. As you can see. Now to test these out.



You need to dload a software called NQ, TTC Explorer and the software will help you connect you to your MQ Tweetie or Cloud and Kutty T before moving forward with entering Alerta inside your E. S. P court. Now choose your platform. Dload the software for four Windows.

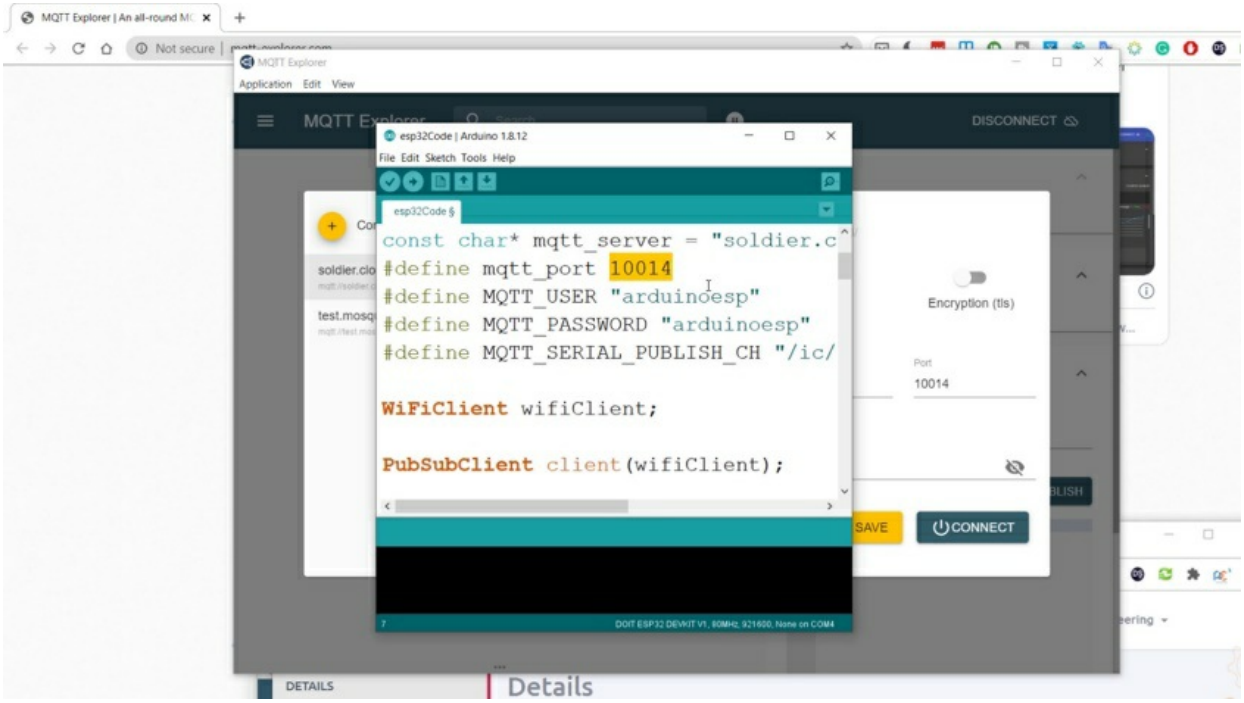


And once you dload the software, you can easily. Can make two or mq tde. And as you can see, I have connected here. Let's let's this connect. Okay.

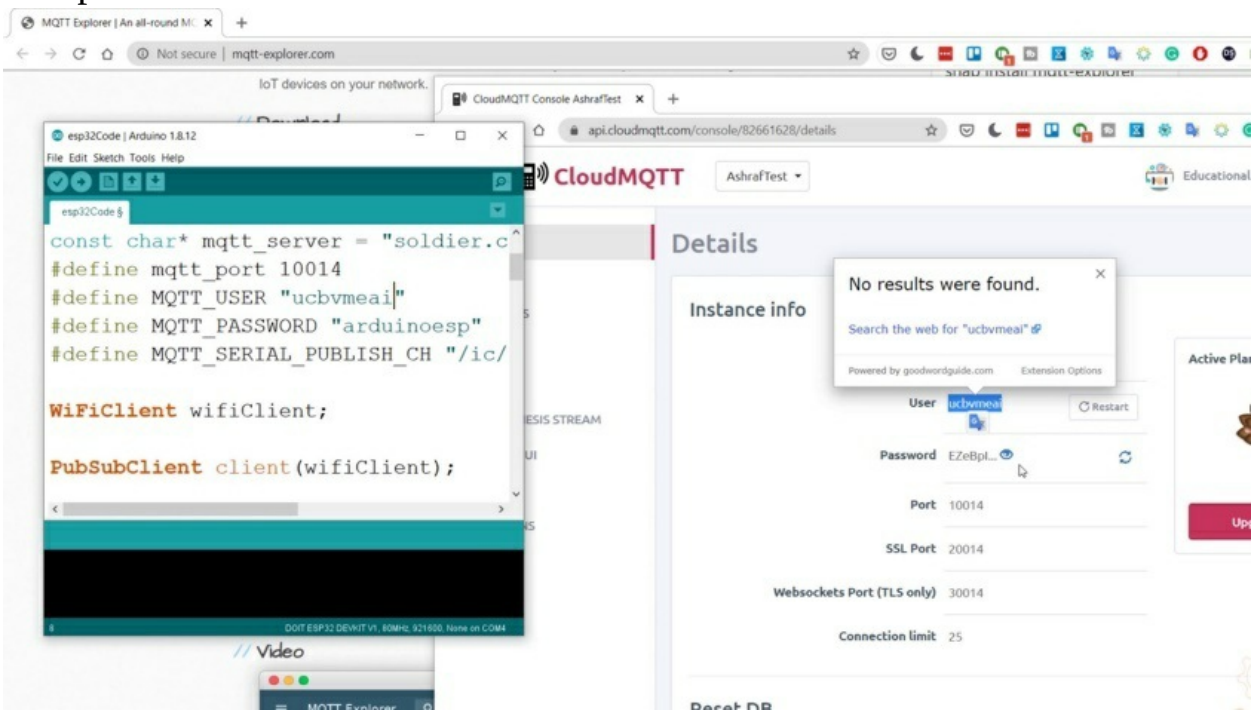


Now, this is. Our cloud in cutey, you can go to the dload or the tales page Kobie the name or the server and based tier on the host code with a port. And here we have the board ten thousand fourteen and based at T, it could be the

user name from Hangar's place tiers and could be the port. The password is unplaced here. These are the same details that we need here. Now our code.

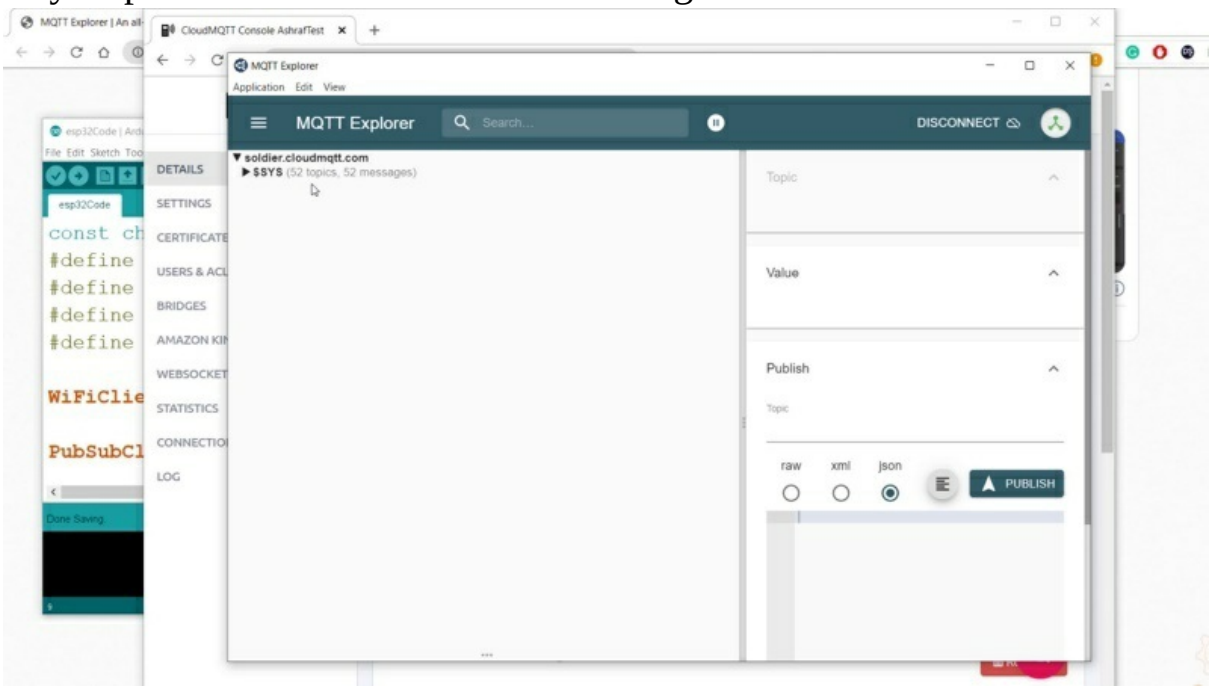


As you can see, this is the port. This is that. You are. This is a used-up name. We need to change it. Who was this user name from this page? A single. Of the space.

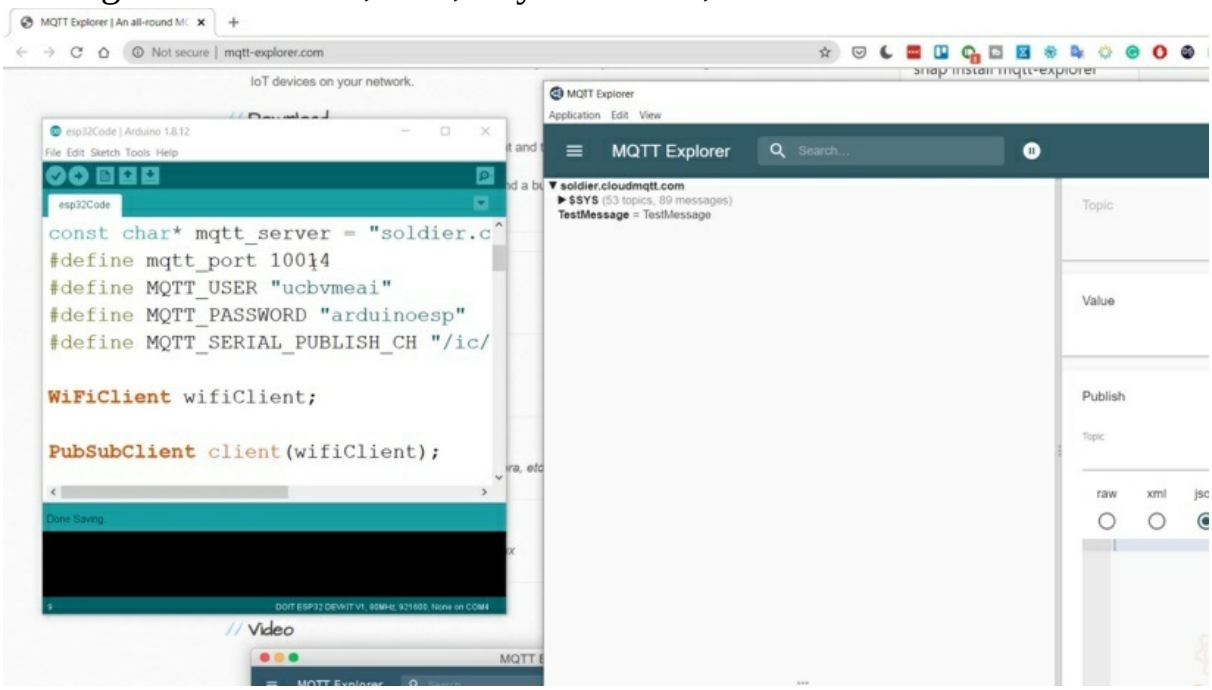


As you can see here with those of Maine Lobach, two of code best teams

with a password. From here on based to tier, then savior of court. Now, to make sure that our cloud computing is working, once you can name it MQ Tity Explorer. Click Connect after entering the details.

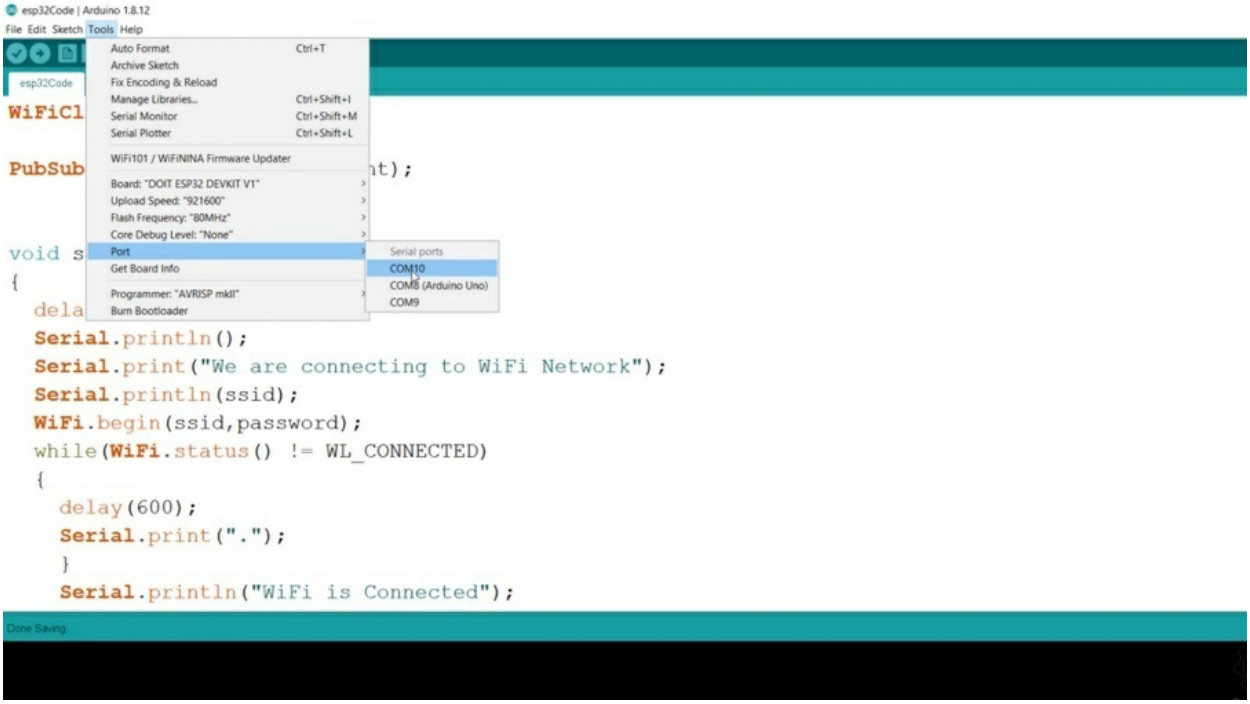


And now, as you can see, it's connected. Go back here and select Web Socket. You I. We are going to try to send a message. Let's call it a text message. And Glikson, now, as you can see, we have a T-shirt.

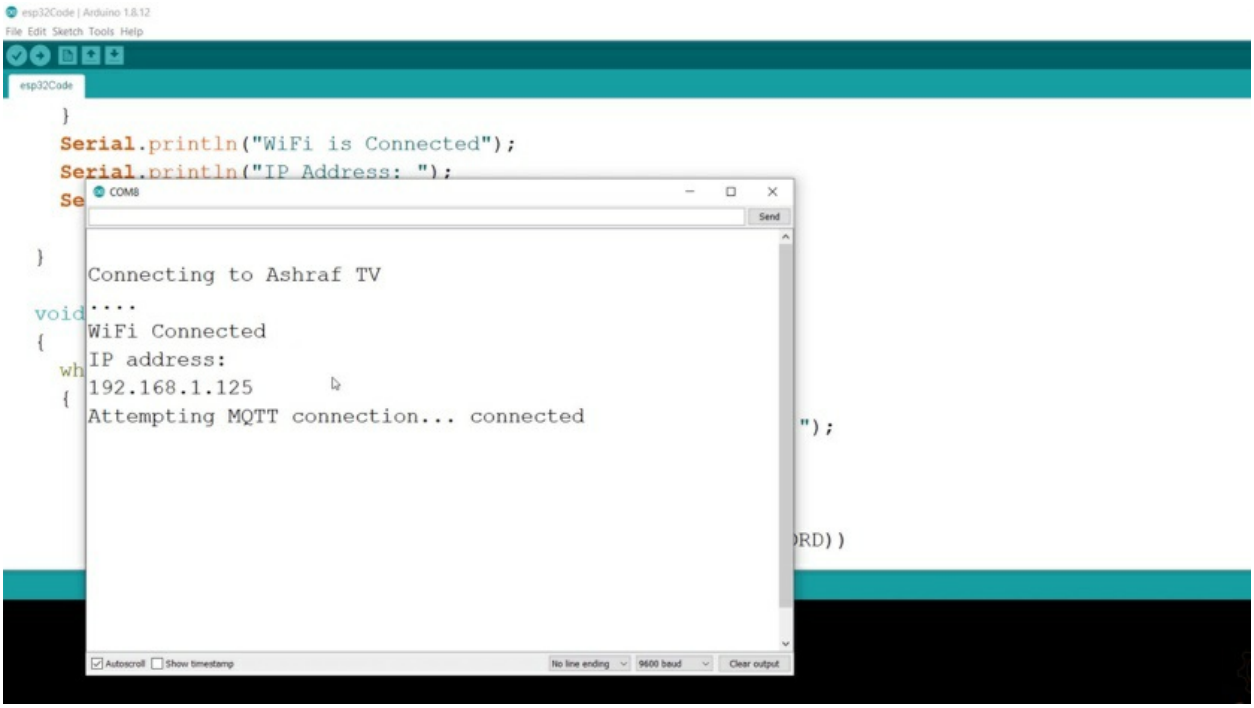


Now, as you can see, we also have it here, an M Kutty explorer, which means

the information that we did, in turn, are E. S. P. Is correct. Make sure to choose the right and Kutty selfish link port username and password to make sure that you are connected. Let's minimize this. Now, this is our code. I will base my password on two years. That's it. Now, once you have all it is, your Wi-Fi network tells it incorrectly and you'll impute tea server details incorrectly. What you need to do at this point is go to the tools, make sure that you are selecting the right board E. S. P 32 Dev Kit, variant one, and make sure that you are selecting the right compost. And simply click. Upload.



That's it. Now, once that was uploaded. Just open up the serial monitor.

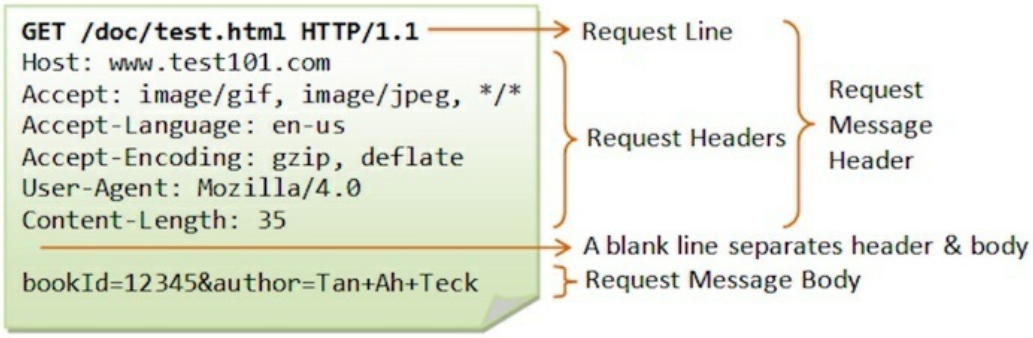


And if you'll give this message between that you are now connected to the MQ server after being connected to the Internet using your Wi-Fi network. And you should see your IP address. This is how you'll make sure that everything is working just fine and just as planned for the U. S. people. The next step is connecting these two balls so that they can start exchanging information. And you will get to read all data from the online imputed t cell.

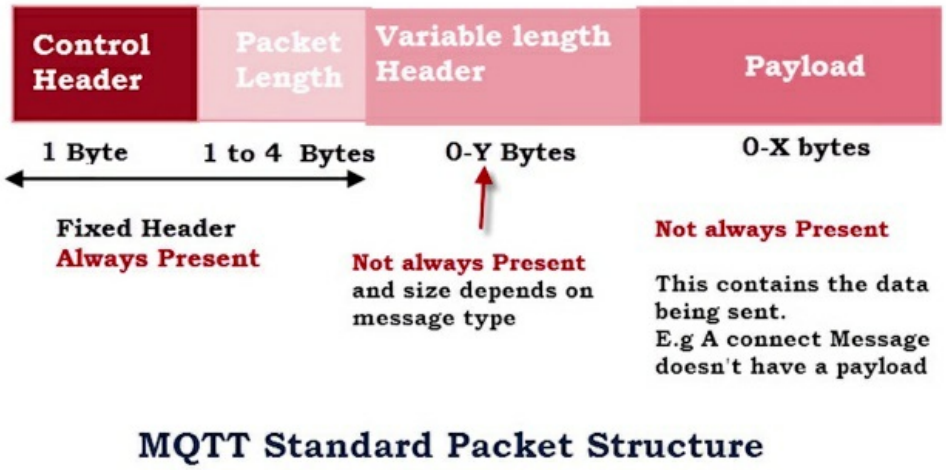
INTRODUCTION TO MQTT

We will learn the following What is MQTT and how it compares to HTTP How MQTT works You know about HTTP or Hypertext Transfer Protocol on which the web operates Now let's look at a protocol specially designed for IoT applications MQTT or Message Queuing Telemetry Transport MQTT is primarily employed for constrained internet of things devices It is ideal for Low bandwidth, high latency, and unreliable networks You know about HTTP or Hypertext Transfer Protocol on which the web operates Now let's look at a protocol specially designed for IoT applications MQTT or Message Queuing Telemetry Transport MQTT is primarily employed for constrained internet of things devices It is ideal for Low bandwidth, - But

why even prefer MQTT over HTTP? Well first off, MQTT is a lightweight protocol when compared to HTTP This means that the MQTT message packet size is much lower than HTTP At its lightest, the minimum MQTT packet size is just 2 bytes

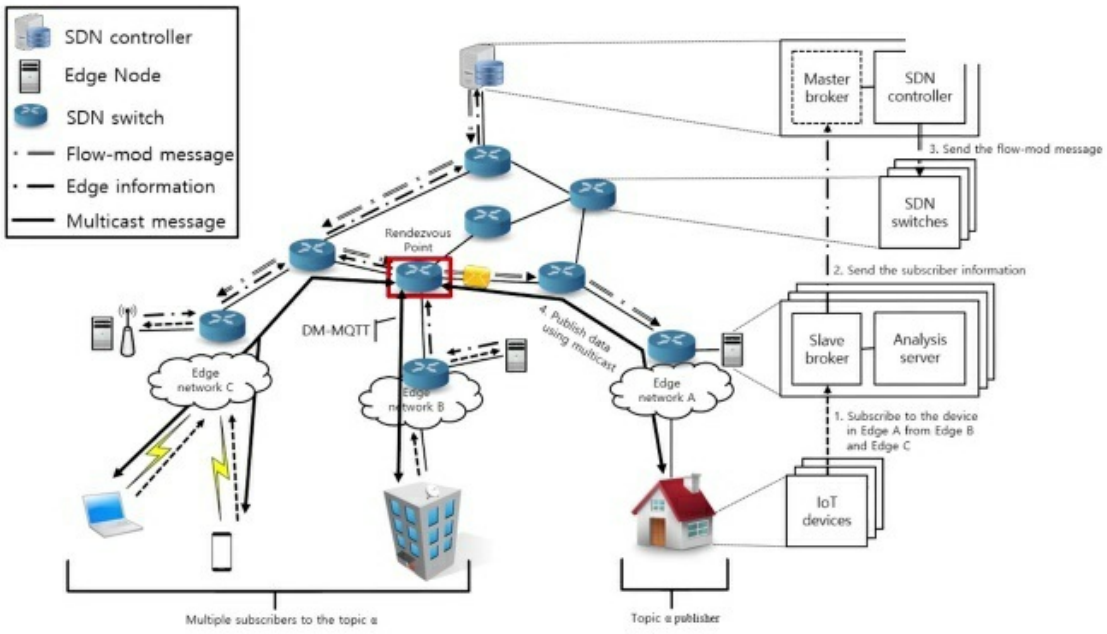


This is because unlike HTTP which transfers messages in a document like structure,

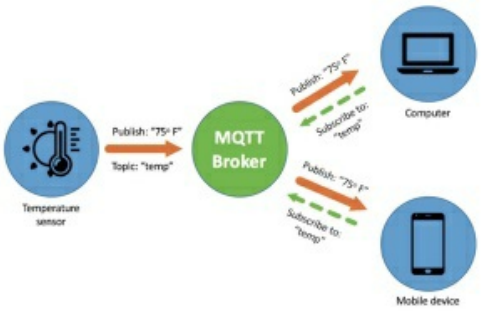


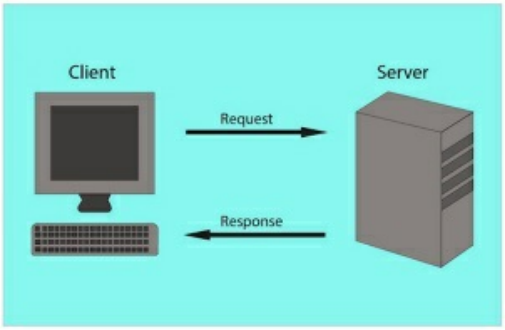
MQTT Standard Packet Structure

MQTT messages are transferred as a byte array

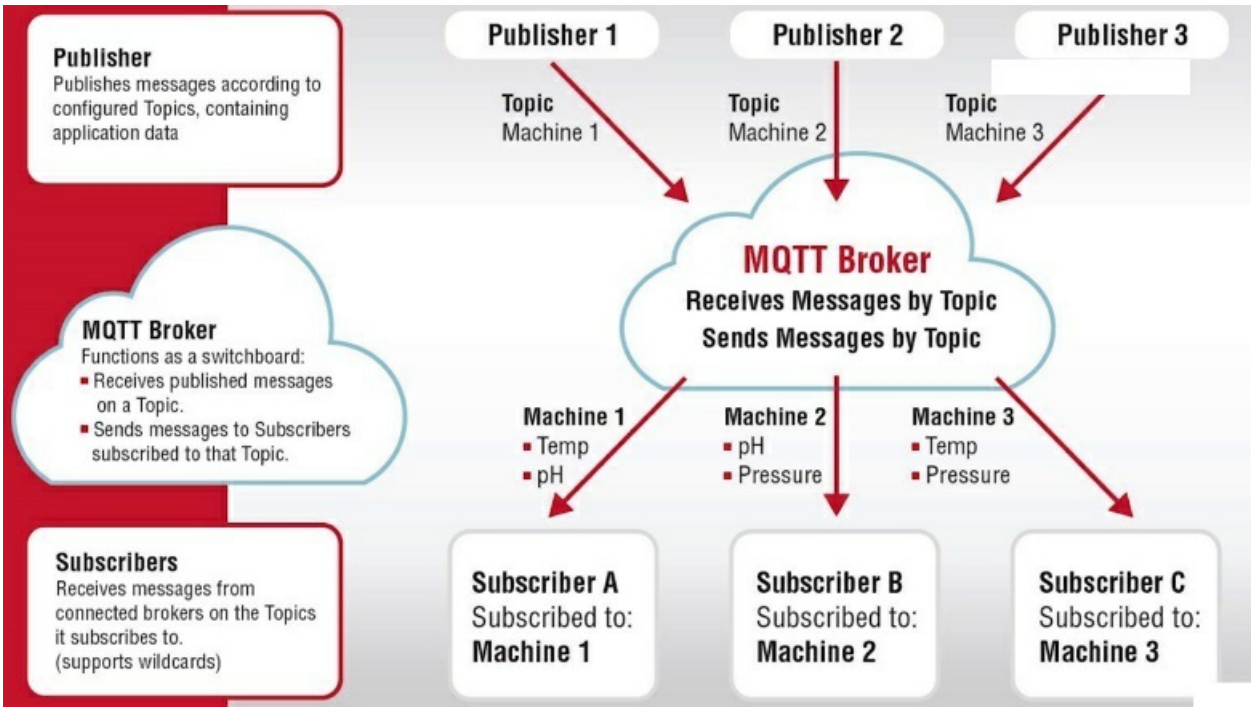


This makes MQTT great for low bandwidth networks, as it lowers the amount of data transferred over a constrained network

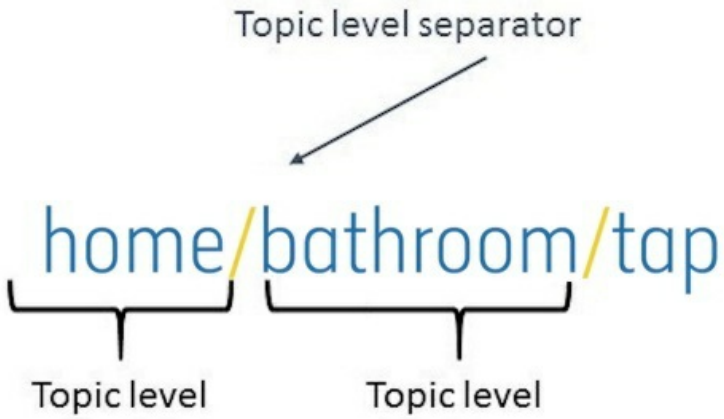




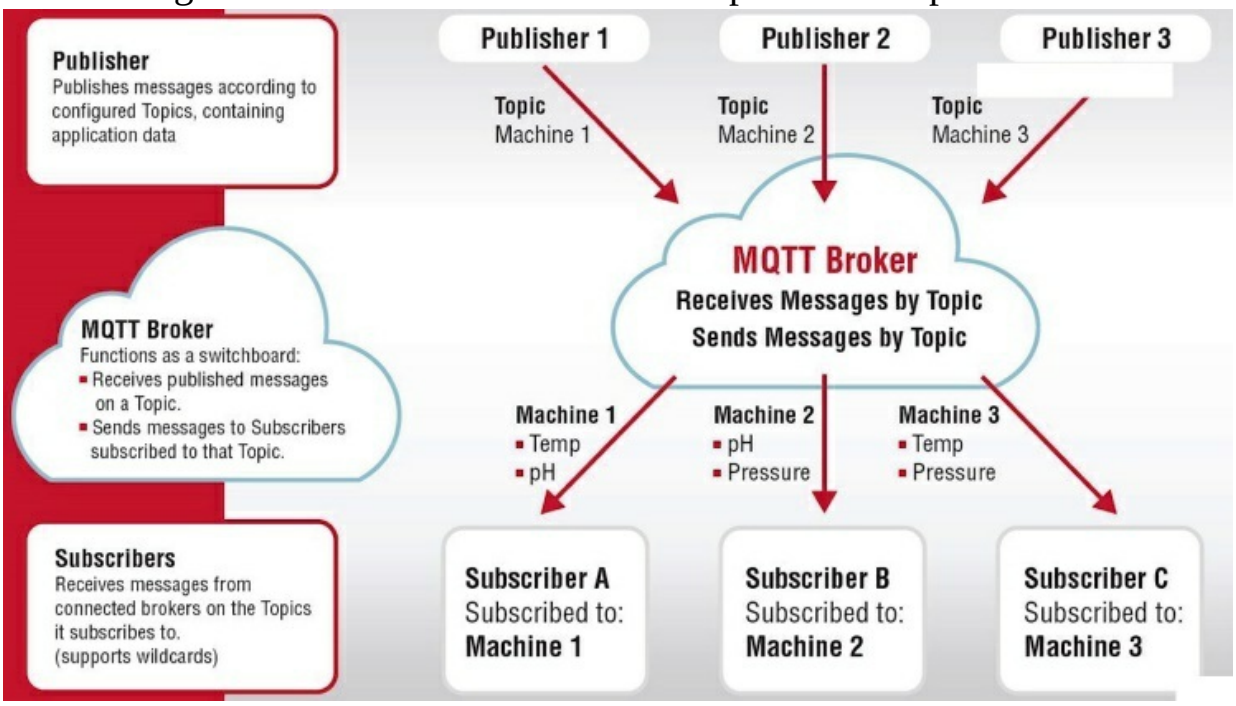
Now MQTT works on the Publish/Subscribe model, whereas HTTP works on a request/response model



So you may ask, what's the advantage of using such a model? To understand this let's look at how the publish/subscribe model works A client device can publish a message to a topic The message is what contains the actual data, which can be sensor data, commands, etc Now topics are a way to specify where you want to publish your message and to which device, For example, to open a tap in your bathroom you would need to publish a message to Open or Close the Tap



But the tap could be in your living room too, so to differentiate between which tap you need to open you would need to create topic levels Then you can exactly specify that you want to turn on the Tap in your bathroom and not your living room Topic levels are separated by forwarding slash Topics itself are in string format and each forward slash represents a topic level



Now if another client device wants to receive the published message, it would

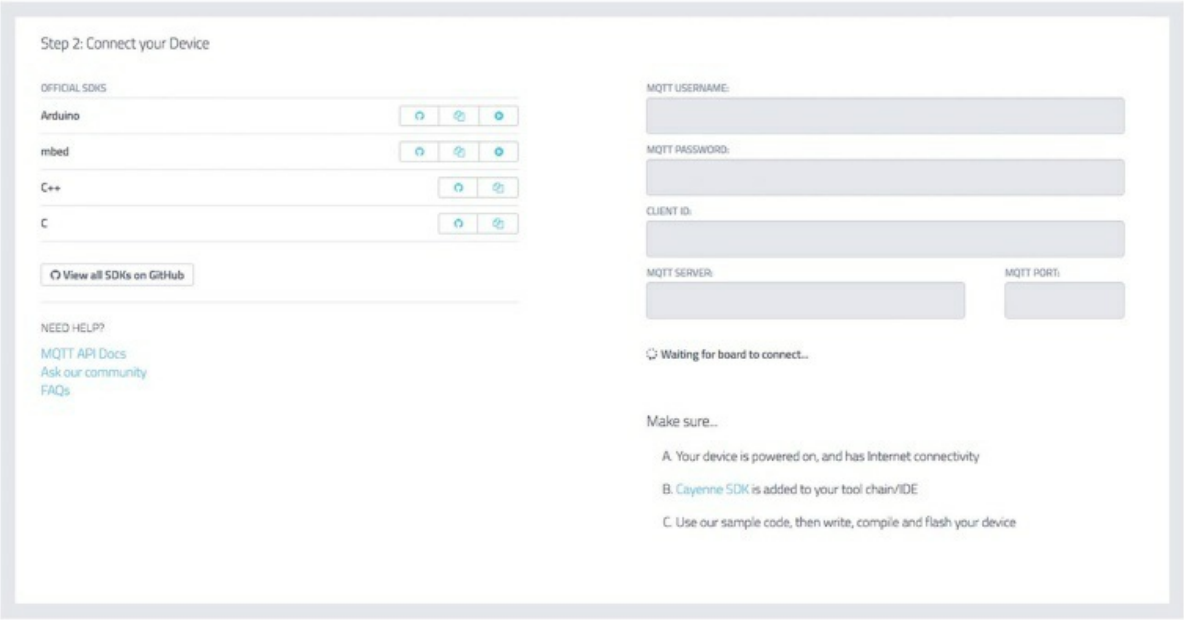
need to subscribe to the particular topic on which the message was published We will discuss it in in detail Now to manage all the topics and messages being delivered to the subscribed clients, we need to use an MQTT broker The MQTT broker is responsible for receiving all the published messages on various topics, filtering them, and publishing it to the respective subscribed clients Multiple devices can be subscribed to the same topic which means you can control multiple devices together without establishing one to one connection with each one of them

This is again one of the features which make MQTT a go-to protocol for IoT applications When messages get published to topics to which no client gets subscribed, the messages automatically get discarded

Now there are various MQTT brokers available, and some can be run locally on your computer's operating systems, like the mosquito broker But wouldn't it be great if you could send your sensor data to the cloud, and access it remotely? Or remotely control actuators from the comforts of your home?

All of this is possible using Cayenne, a cloud-based IoT drag and drop project builder Now, the Cayenne cloud acts as the MQTT broker, to which

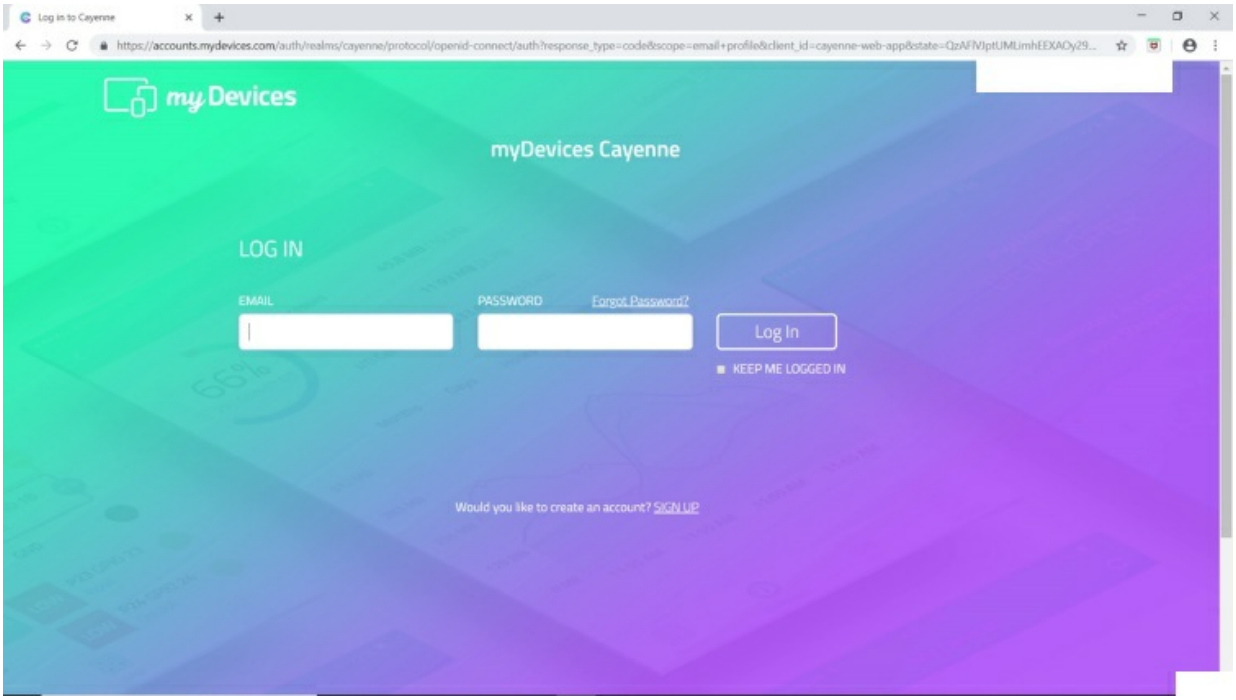
all client devices can send data to and receive commands from the Cayenne dashboard



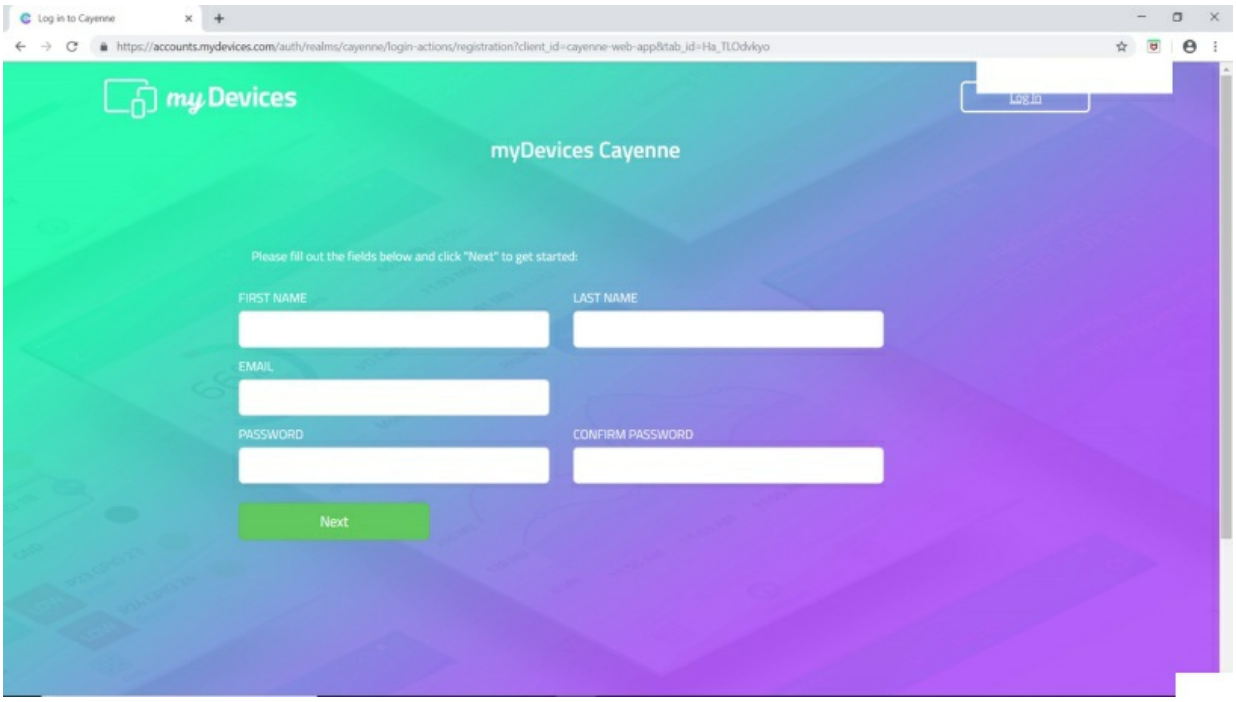
Cayenne has an MQTT API which facilitates this connection The Thing here will be the Client device that will be establishing an MQTT connection with the Cayenne broker Now the easiest way to connect the Thing to the Cayenne Cloud is through the Arduino IDE The Cayenne Arduino MQTT library bundles the Paho Eclipse MQTT client which easily allows you to interact with the Cayenne MQTT broker we learned what MQTT protocol is and how it compares to HTTP We also learned about how MQTT works In you will learn about interfacing the Thing to the Cayenne API

INTERFACING THE THING TO CAYENNE

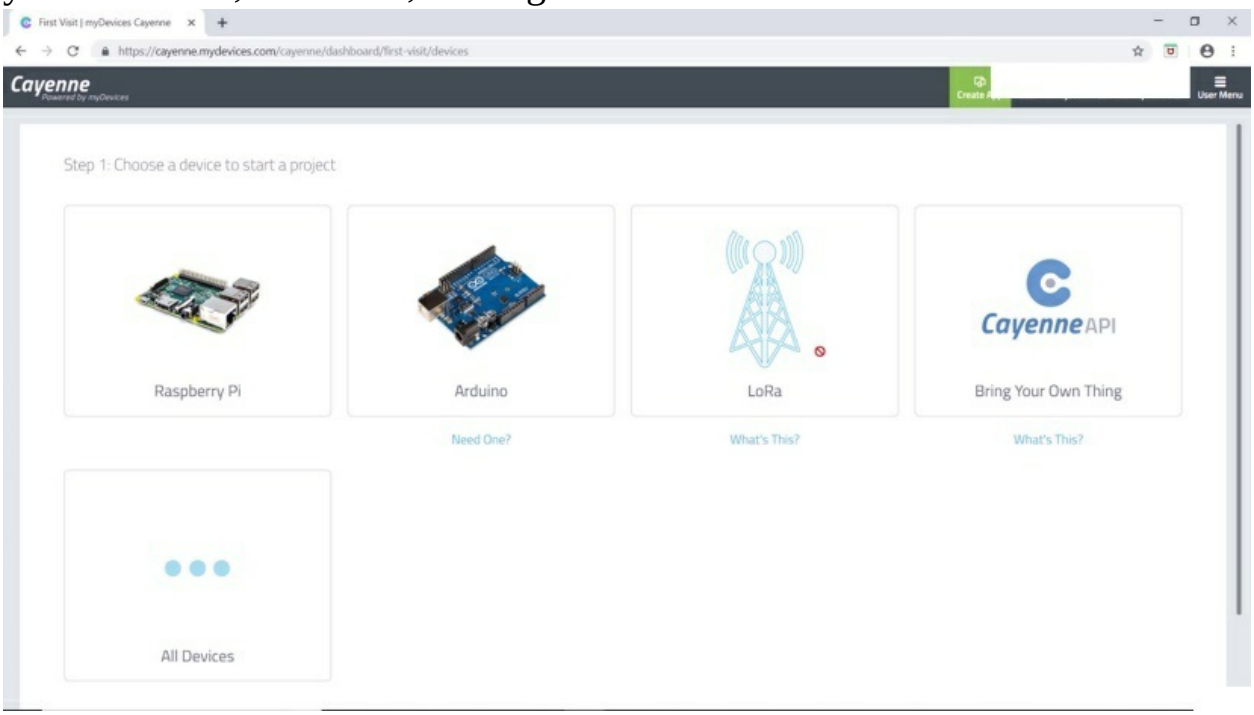
We will learn the following How to add the Thing to cayenne using the Cayenne MQTT API How to write data to a Channel on the Cayenne Dashboard



Let's go to the Cayenne website now The link for the same is given in the Resources section Here you will be asked to enter your Email ID and Password You first need to create an account with Cayenne

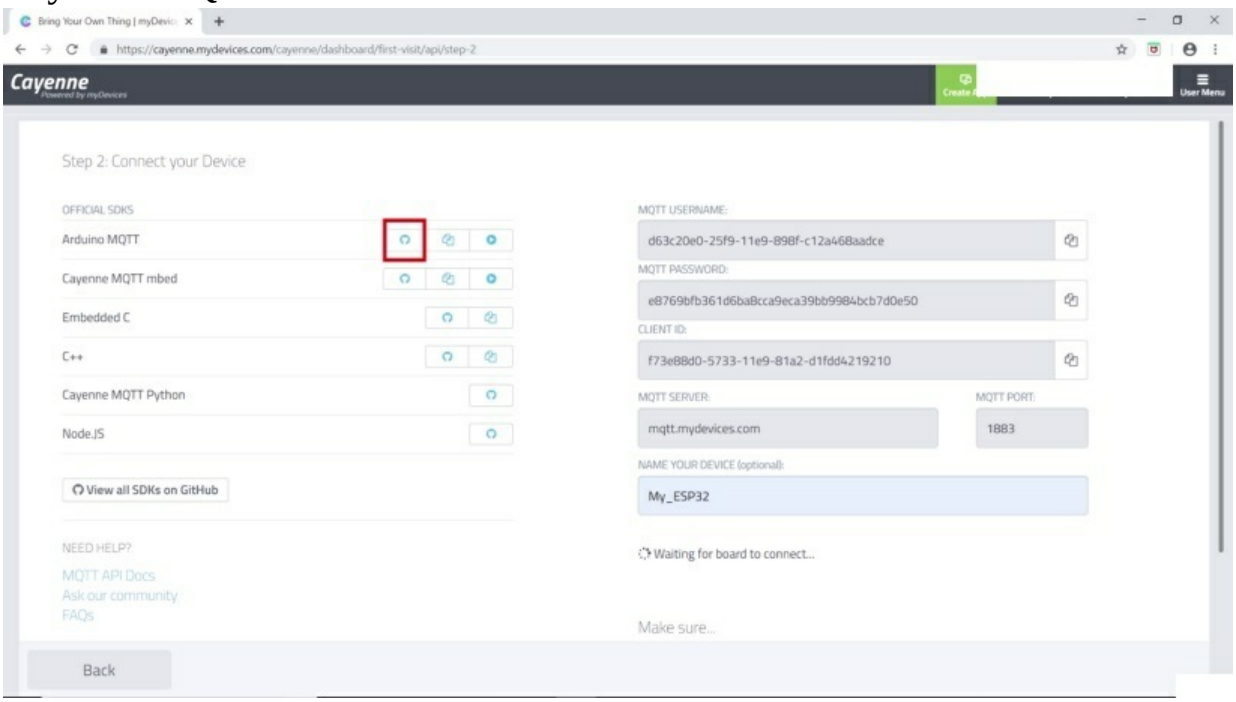


Click here to sign up Now enter all your details and note d your Email ID and Password which you will be needing to Login Once you have created your account, you need to go to the myDevices Cayenne login page Now enter your Email ID, Password, and login

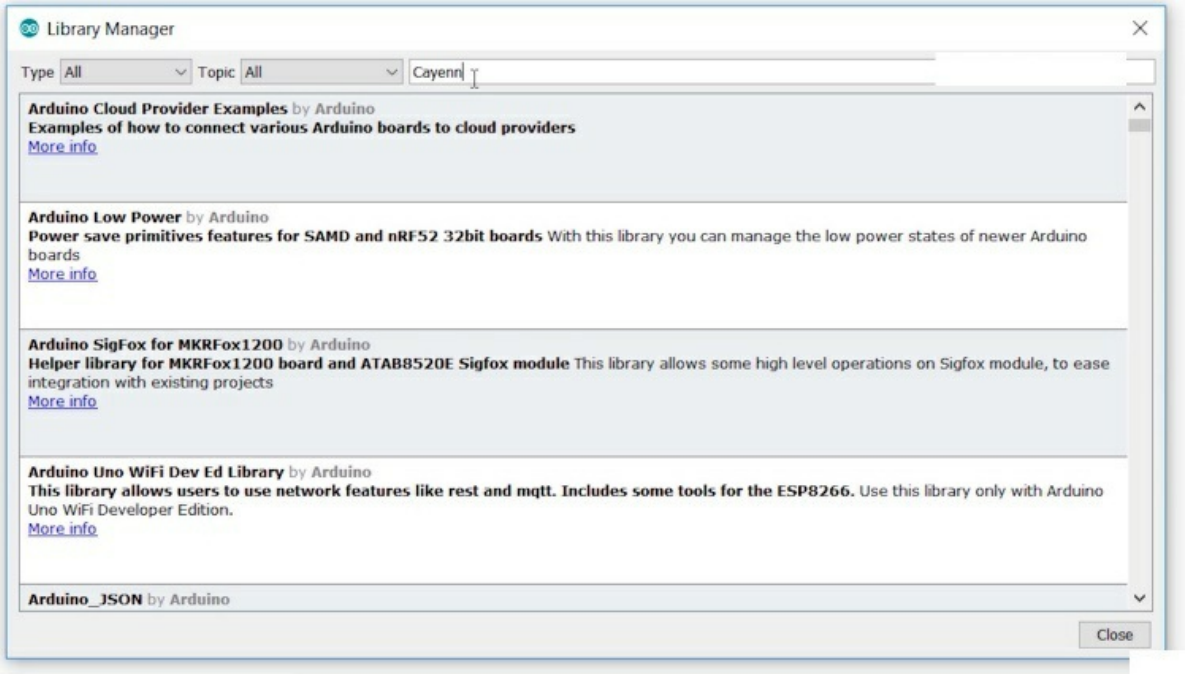


Firstly, you will be asked to choose your device which you want to interface and work on As you can see, there is already prebuilt support for the

Raspberry Pi, Arduino, and LoRA But you will not find the Sparkfun ESP32 Thing here You will need to manually add the Thing as a device using the Cayenne MQTT API



Now click here Now as you can see, there are several official SDKs available for MQTT support The Arduino MQTT SDK is one of them The Cayenne Arduino MQTT library contains functions and code and which would help you to easily connect to the Cayenne IOT dashboard It has support for Arduino microcontroller boards, as well as other ESP8266 and ESP32 based development boards You can click here to access the Github page for the same Now let's go back to the Arduino IDE and see how we can install the Cayenne ESP MQTT library

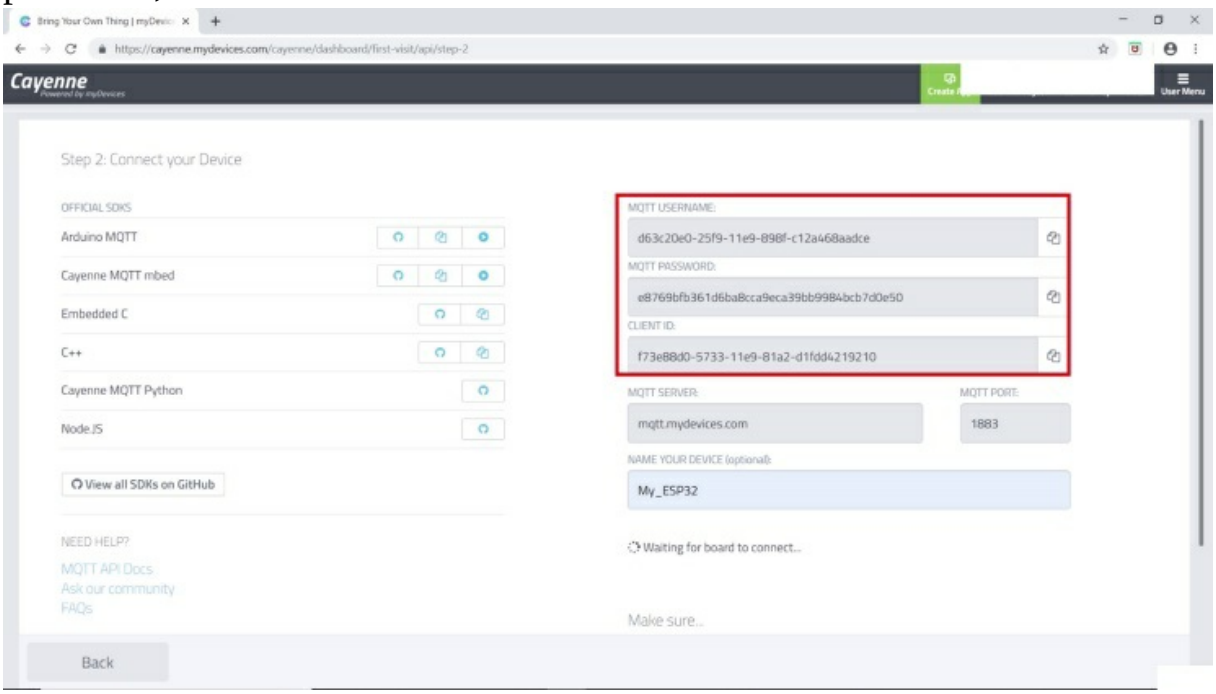


First, you need to go to tools and Manage libraries This will open up the Library Manager Now search for cayenne, you will find the Cayenne MQTT library Now install the latest version of the library Once installed you are now ready to program the Thing to connect to Cayenne

```
4. Set the network name and password.
5. Compile and upload the sketch.
6. A temporary widget will be automatically generated in the Cayen... Dashboard. To
*/

#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "ad3ce7c0-5690-11e9-81a2-d1fdd4219210";


void setup() {
```

Now let's look at the code to connect the Thing to cayenne This here is used to printout the debug messages like connection status, topic number and

channel number of messages being published to and the actual data itself Now this here is used to printout the general WiFi credentials and the server with the port number to which the client is connected to You can uncomment both of these here you don't want to print the debug messages and general WiFi status Now here we import the Cayenne MQTT library for the ESP32 It contains functions which would enable you to send data to Cayenne as well as receive data Now here you need to enter your WiFi credentials Now here you need to enter the Cayenne authentication information like the Username, password, and client ID



You can copy the same from the Cayenne dashboard and paste it in the respective fields The Client ID is used to distinguish between different clients connected to Cayenne Now here we initialize the Serial baud rate at 9600 Now this is needed to connect to cayenne with the Cayenne username, password client ID, WiFi SSID, and WiFI password You do not need the WiFi to begin function here to connect to the WiFi as the Cayenne begin function takes care of connecting to the WiFi too

```
// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "ad3ce7c0-5690-11e9-81a2-d1fdd4219210";


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();
}
```

Now we include the Cayenne loop function here to execute all the cayenne functions repeatedly

```
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();
}

// Default function for sending sensor data at intervals to Cayenne.
// You can also use functions for specific channels, e.g CAYENNE_OUT(1) for sending channel 1 data.
CAYENNE_OUT_DEFAULT()
{
  // Write data to Cayenne here. This example just sends the current uptime in milliseconds on virtual channel 0.
  Cayenne.virtualWrite(0, millis());
  // Some examples of other functions you can use to send data.
  //Cayenne.celsiusWrite(1, 22.0);
  //Cayenne.luxWrite(2, 700);
  //Cayenne.virtualWrite(3, 50, TYPE_PROXIMITY, UNIT_CENTIMETER);
}

// Default function for processing actuator commands from the Cayenne Dashboard.
// You can also use functions for specific channels, e.g CAYENNE_IN(1) for channel 1 commands.
CAYENNE_IN_DEFAULT()
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  //Process message here. If there is an error set an error message using getValue.setError(), e.g getValue.setErro
}
```

Now the Cayenne out default function is where you write data to the client It is used to publish data in the form of messages to the Cayenne broker and update the Channel In the Cayenne dashboard, the channel is where your actual data gets displayed Now the virtualWrite function is used write data to

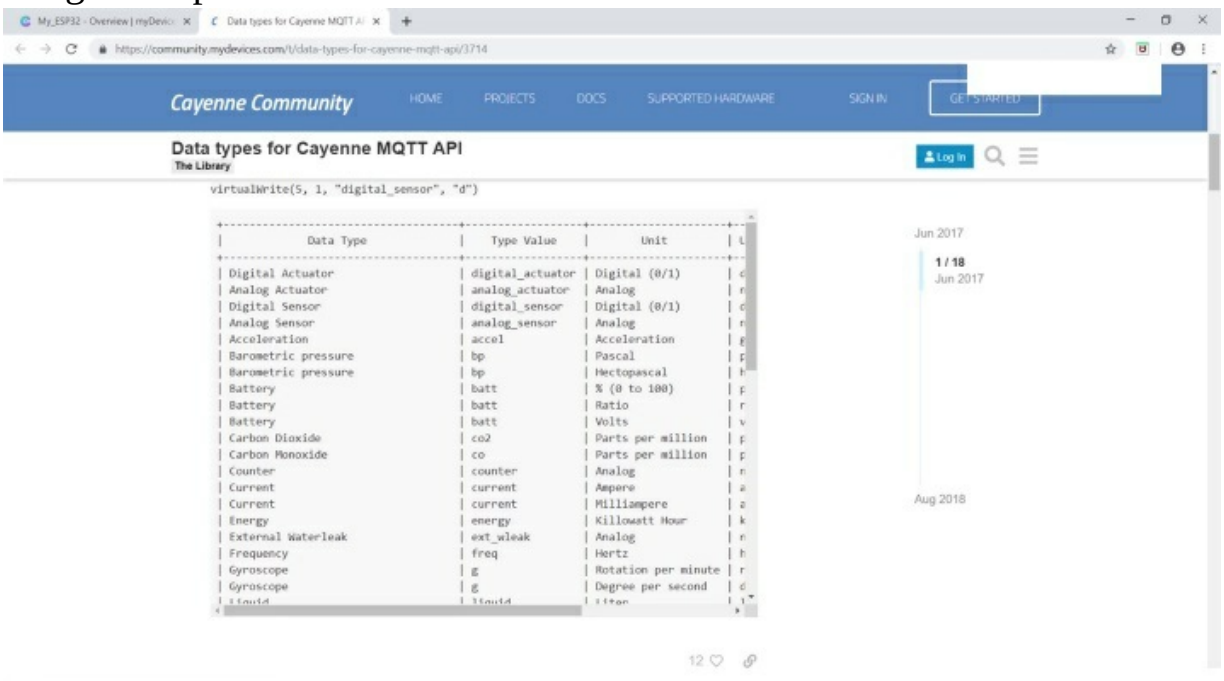
the particular channel, in this case, being 0 and the actual data to be sent Here we are just sending the time elapsed since the program started running This is given by the millis function, which returns this time in milliseconds

```
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();
}

// Default function for sending sensor data at intervals to Cayenne.
// You can also use functions for specific channels, e.g CAYENNE_OUT(1) for sending channel 1 data.
CAYENNE_OUT_DEFAULT()
{
  // Write data to Cayenne here. This example just sends the current uptime in milliseconds on virtual channel 0.
  Cayenne.virtualWrite(0, millis());
  // Some examples of other functions you can use to send data.
  //Cayenne.celsiusWrite(1, 22.0);
  //Cayenne.luxWrite(2, 700);
  //Cayenne.virtualWrite(3, 50, TYPE_PROXIMITY, UNIT_CENTIMETER);
}
```

virtualWrite(mqtt_channel_number, value, "type value", "unit value")

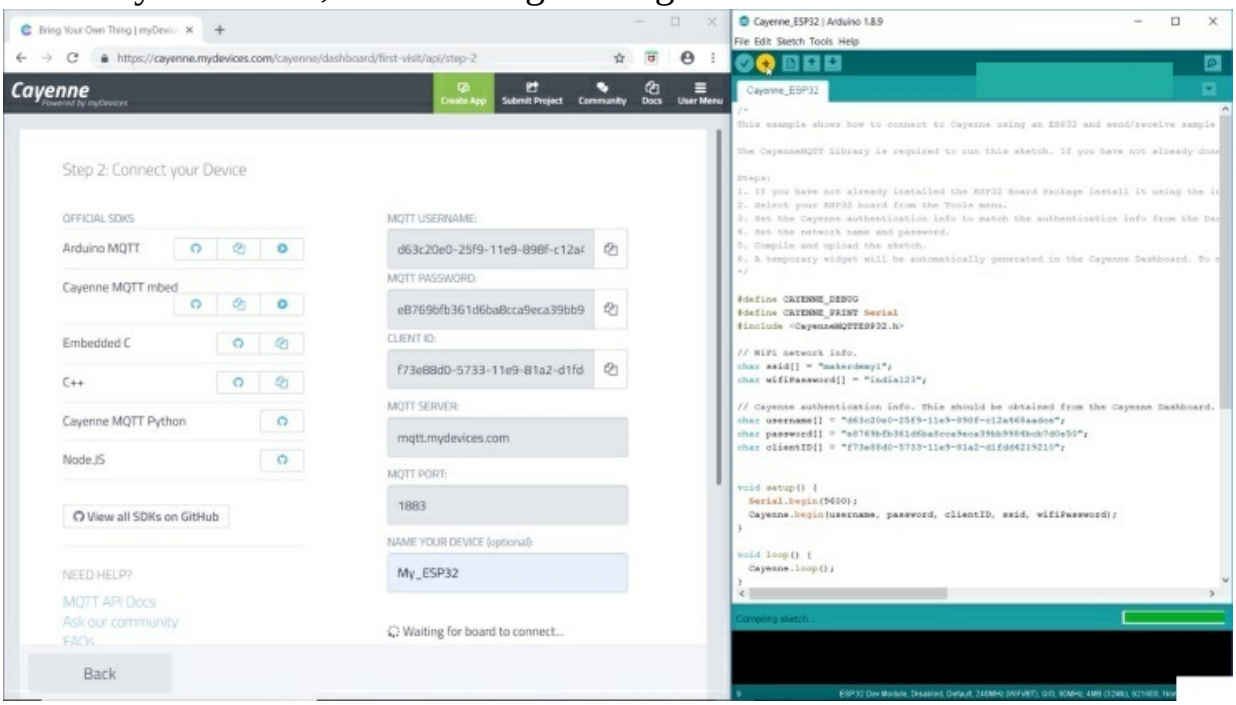
```
// Default function for processing actuator commands from the Cayenne Dashboard.
// You can also use functions for specific channels, e.g CAYENNE_IN(1) for channel 1 commands.
CAYENNE_IN_DEFAULT()
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  //Process message here. If there is an error set an error message using getValue.setError(), e.g getValue.setErro
}
```

The general parameter format for the virtual write function is this

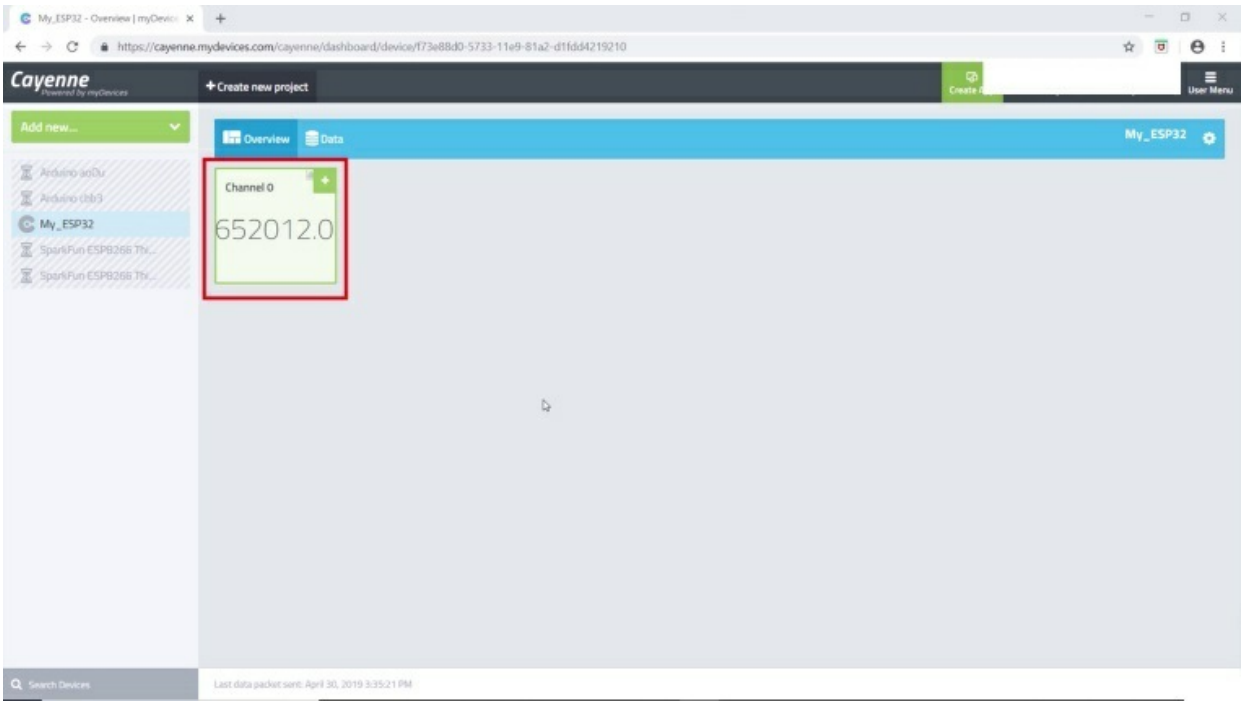


If you need the full list of data types supported by the Cayenne MQTT API, you can refer to this Cayenne community page the link to which is in the

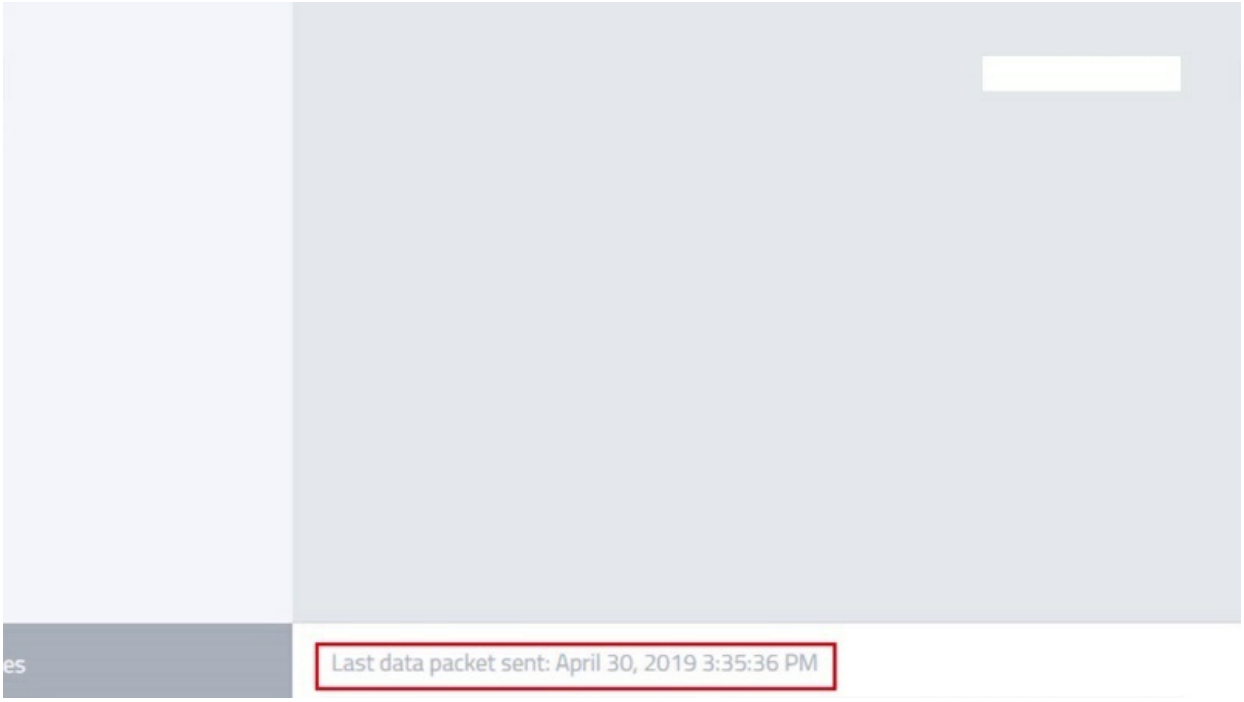
resources section Now, to read commands from the Cayenne dashboard you would need to use the Cayenne In Default function We will look at how to use this in Now let's go back to the Cayenne MQTT API menu Here you can name the Thing to whatever you want this will be displayed in the devices list Now as you can see, it is showing waiting for the board to connect



Now let's go back to the code and upload it Once uploaded you can see that the MQTT connection to Cayenne has been successfully established This is the Cayenne dashboard

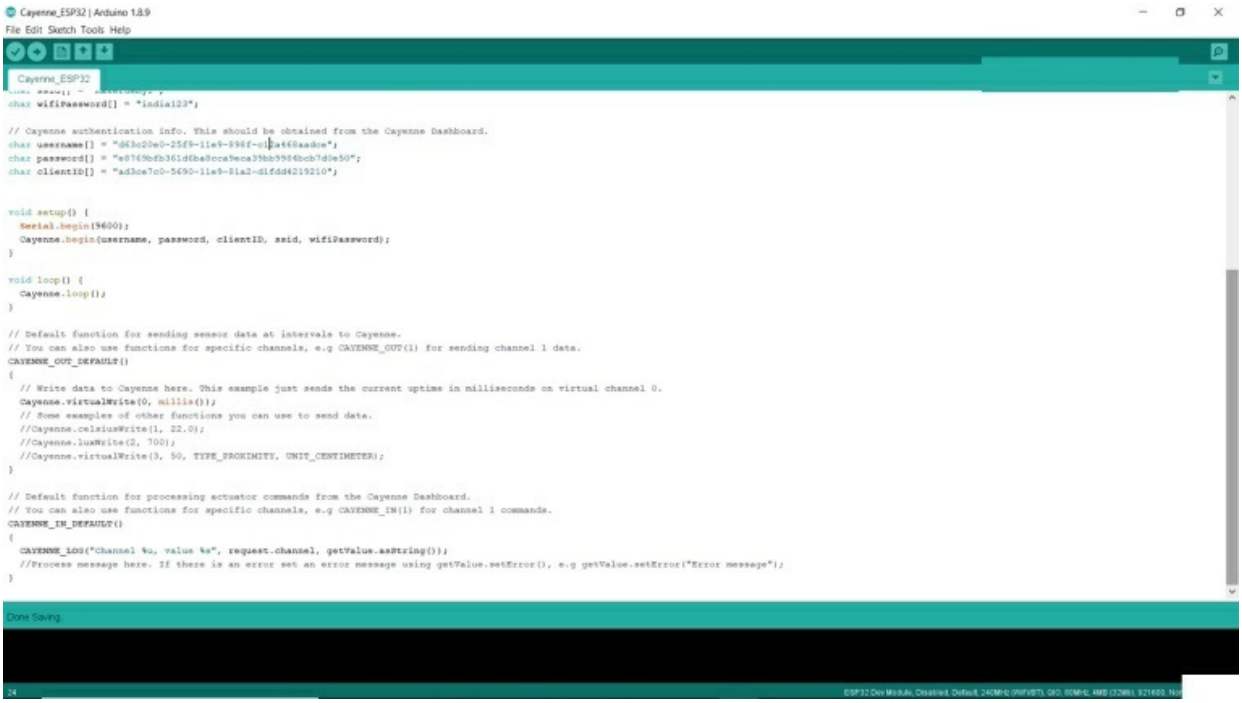


Now in the devices list, you can see that our Thing, named My ESP32 is now available



Here you can see that Channel 0 is showing the time elapsed since the program started Now if you look closely, you will see this message at the bottom It shows the date and the time at which the last data packet was received Now you might notice one thing, the cayenne channel receives the

data packet about every 15 seconds The reason is that the Cayenne out default function is designed to run every 15 seconds;



hence the MQTT messages are published every 15 seconds to Cayenne Currently, cayenne MQTT messages are rate limited to 60 messages per minute, which means a maximum rate of 1 message per second If the client code tries to publish messages above this rate, it may cause messages to be dropped or client to get disconnected.

SETTING THE MESSAGE RATE AND CREATING A CUSTOM WIDGET

We will learn the following How to set your MQTT message publishing rate How to create a custom widget Now previously we saw that the rate at which the MQTT messages were published was capped at a maximum rate of 60 messages per minute Now let's see how we can adjust this rate to suit our needs Now, you don't need to use the Cayenne out default function to write data to the Cayenne channel as it is capped at a default rate of 1 message every 15 seconds Lets see how we can change this rate to 1 message per second

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "12abfec0-6bc2-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;

void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();

  if(millis() - lastMillis > 1000) {
```

The initial code and setup code is the same as seen in the Except here we are initializing an unsigned long integer last millis as 0 We will be using this later in the code

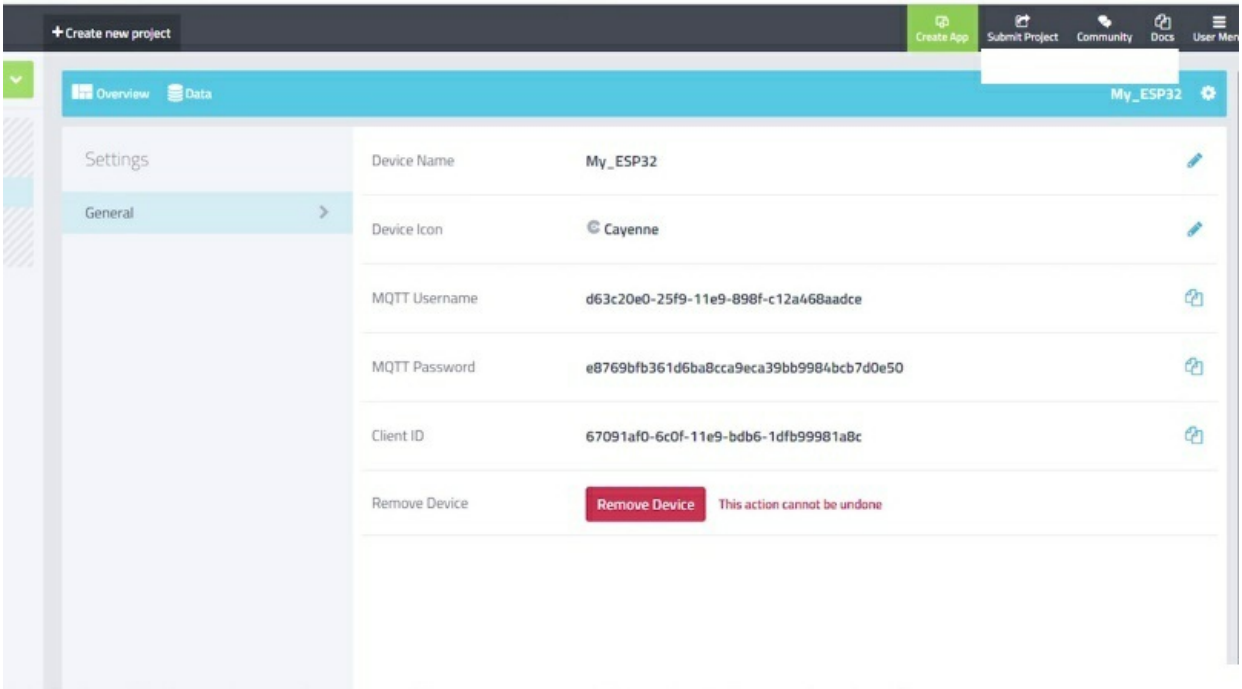
```
// Default function for sending sensor data at intervals to Cayenne.
// You can also use functions for specific channels, e.g CAYENNE_O             ding
CAYENNE_OUT_DEFAULT()
{
  // Write data to Cayenne here. This example just sends the current uptime in mill
  //Cayenne.virtualWrite(0, millis());
  // Some examples of other functions you can use to send data.
  //Cayenne.celsiusWrite(1, 22.0);
  //Cayenne.luxWrite(2, 700);
  //Cayenne.virtualWrite(3, 50, TYPE_PROXIMITY, UNIT_CENTIMETER);
}

// Default function for processing actuator commands from the Cayenne Dashboard.
// You can also use functions for specific channels, e.g CAYENNE_IN(1) for channel
CAYENNE_IN_DEFAULT()
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  //Process message here. If there is an error set an error message using getValue.
}
```

Since we will not be using the Cayenne out default Cayenne In default functions Cayenne In default functions Now let's look at the loop code Here we are using the millions function which will return the time elapsed since the code ran We are also using the last millis, which was initialized to zero earlier in the code Now we compare, if the current elapsed time in subtracted by the last millis variable, at the first run set to zero, is greater than 1000 then the following code will execute Since the millis function returns the time in milliseconds hence here we are checking if 1000 milliseconds or 1 second has elapsed Now the current time elapsed will be stored in the last millis variable

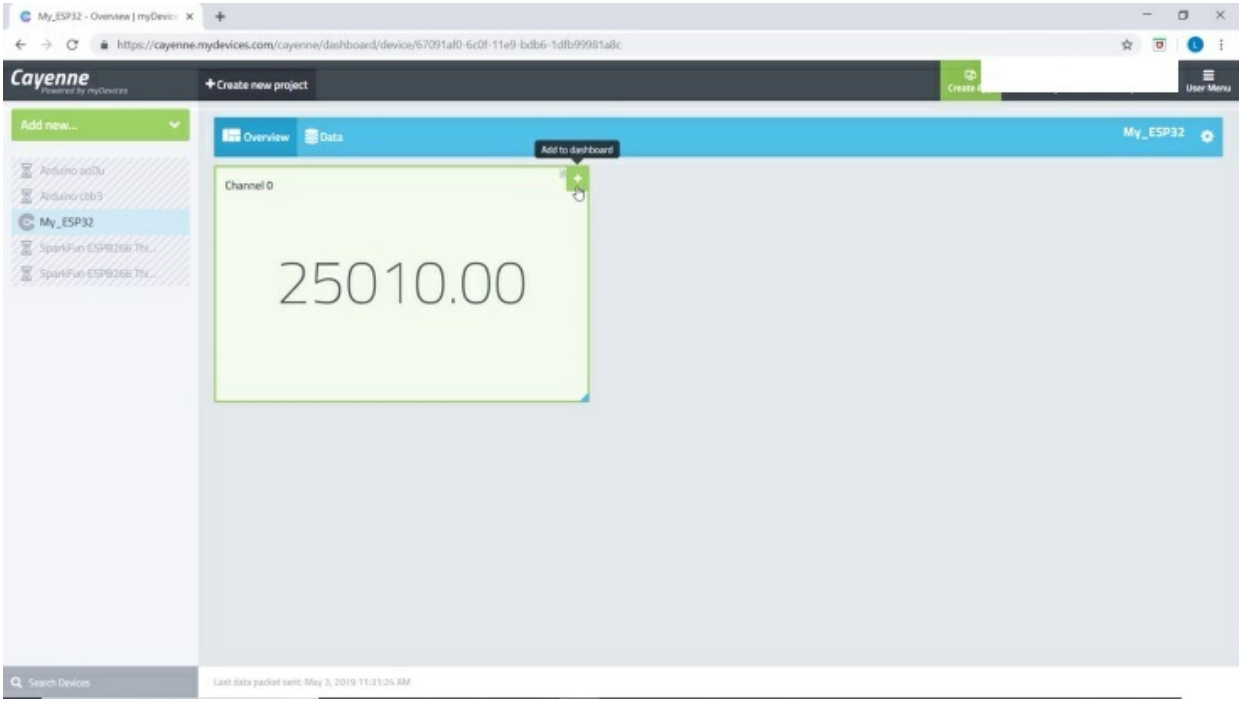
Cayenne_rate

```
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "12abfec0-6bc2-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;

void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();

  if(millis() - lastMillis > 1000) {
      lastMillis = millis();
      Cayenne.virtualWrite(0, lastMillis);
   }
}
```

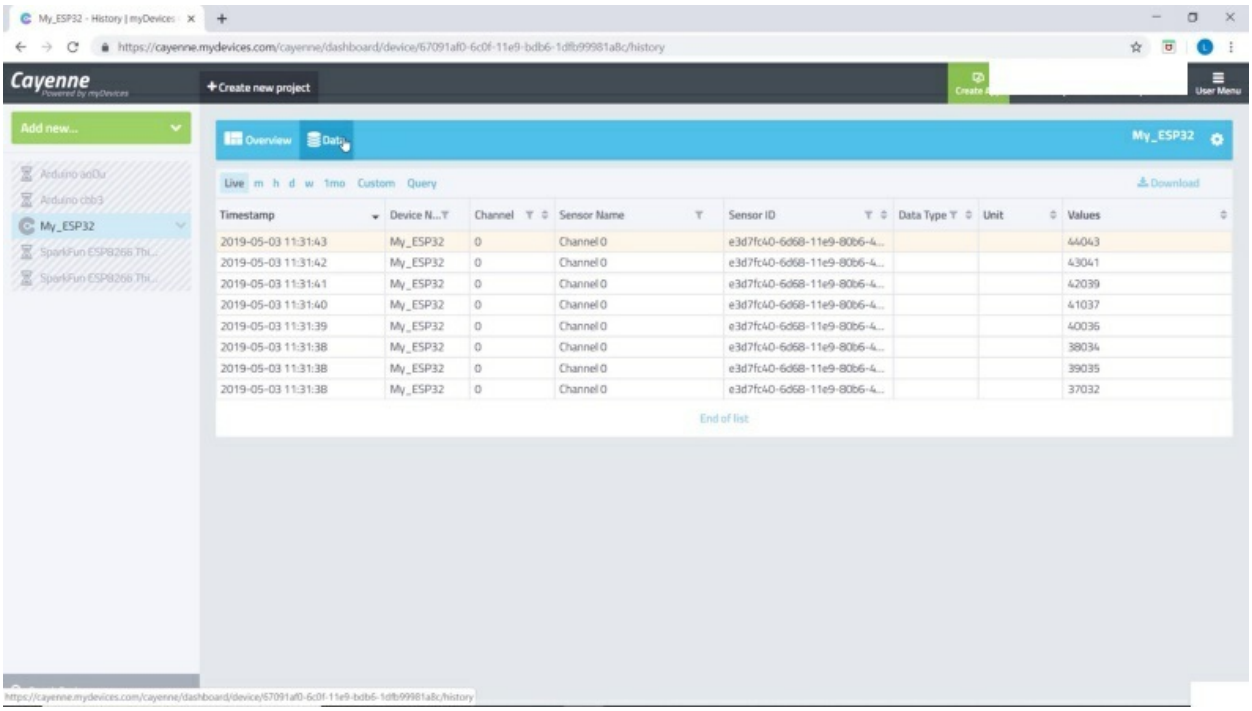
We will also include the virtual write function to write data to the channel 0 with the current time elapsed data So when the loop runs again, this, if condition will check if 1 second has elapsed from the last elapsed time, noted Now before uploading and running the code go to your Cayenne dashboard and in the devices list, click on your cayenne device Now go to device settings here and click on configure



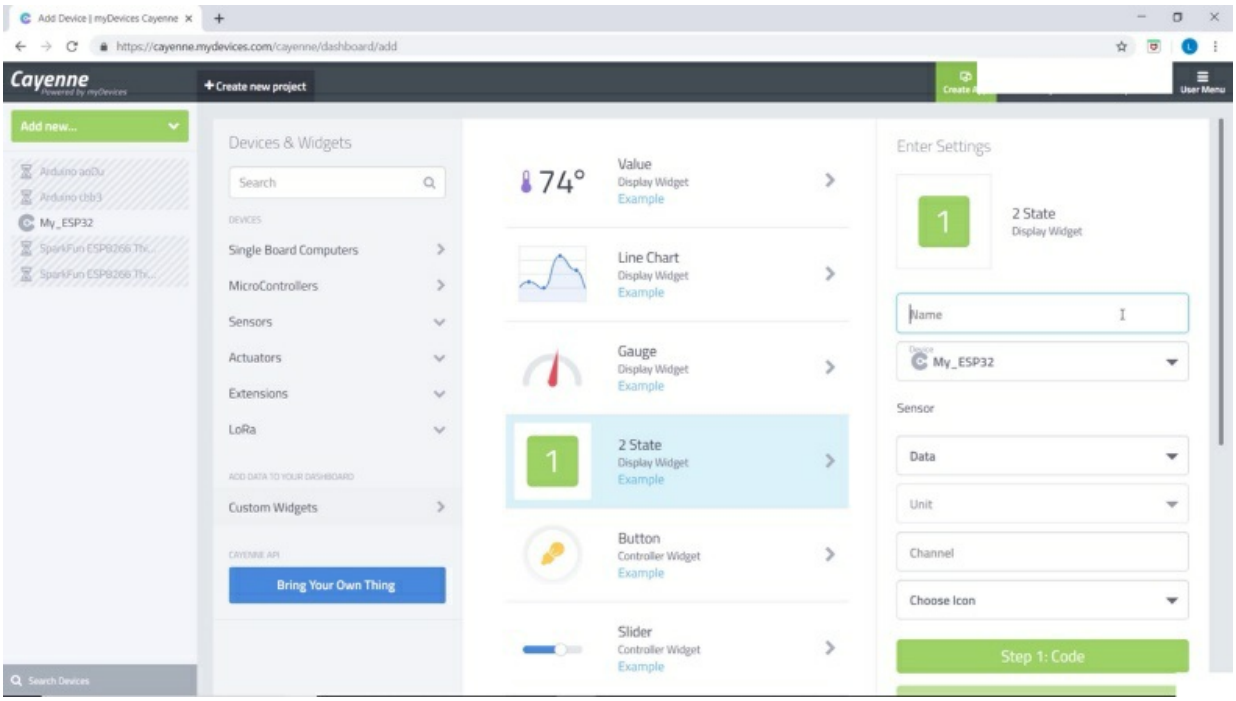
here you will find the device name, device icons which you can choose from, the MQTT authentication information, and the device client ID Now copy the Cayenne MQTT username, password, and client ID to your code Now upload the code



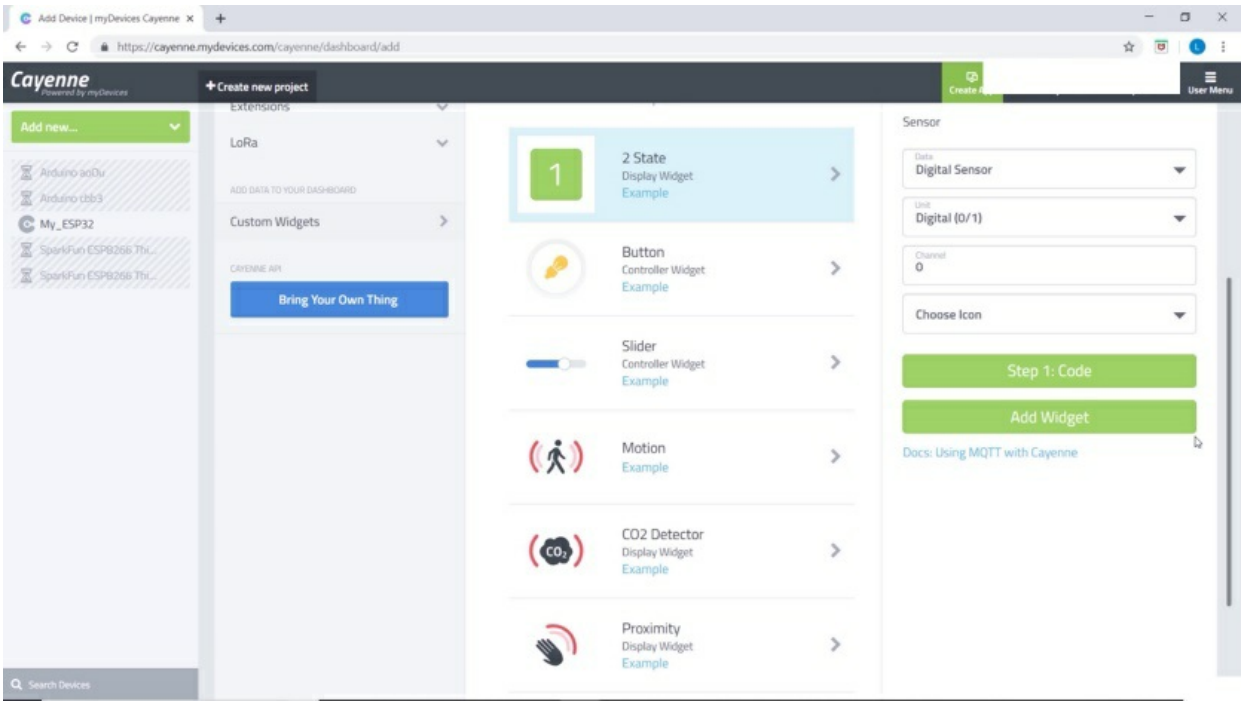
You can see that the messages are received every second as we had set in the code on channel 0 Now to add this channel to your dashboard you can click here



Now go to the data tab here Here you can see the MQTT messages being received Here you can get the information regarding the time of receiving the message, the device name from which the data is received channel number, sensor name, sensor ID, data type, unit and Actual value of the data contained in the message As you can see, we are getting real-time data here Now let's look at how to create custom widgets on Cayenne We will be detecting touch and if touch is detected, we will display 1 else we will display 0 On the Cayenne dashboard, go to Add new and then Device/Widget Now go to Custom Widgets



Here you can find a variety of Widgets Now let's select the 2 state widget In the widget settings, you can name the widget anything you want Here we are naming it to touch Here we can select the device, to which we need to add the custom widget Here we can select the Type Value Since we have the state value as either 1 or 0 hence we choose digital sensor here This has to be the same which we specified in the virtual Write function Now here we can select the Unit as digital 0 or 1 Now in the channel field we need to specify the channel number for the custom widget Since we are publishing the data to Channel 0, hence we specify channel 0 here



Now in the icon menu, you can select a variety of icons for various sensors and actuators Its not mandatory though Now click on Add Widget here The custom widget will be added to the dashboard Now let's look at the code Here we assign PIN15 which is the touch sensor pin to the variable touch

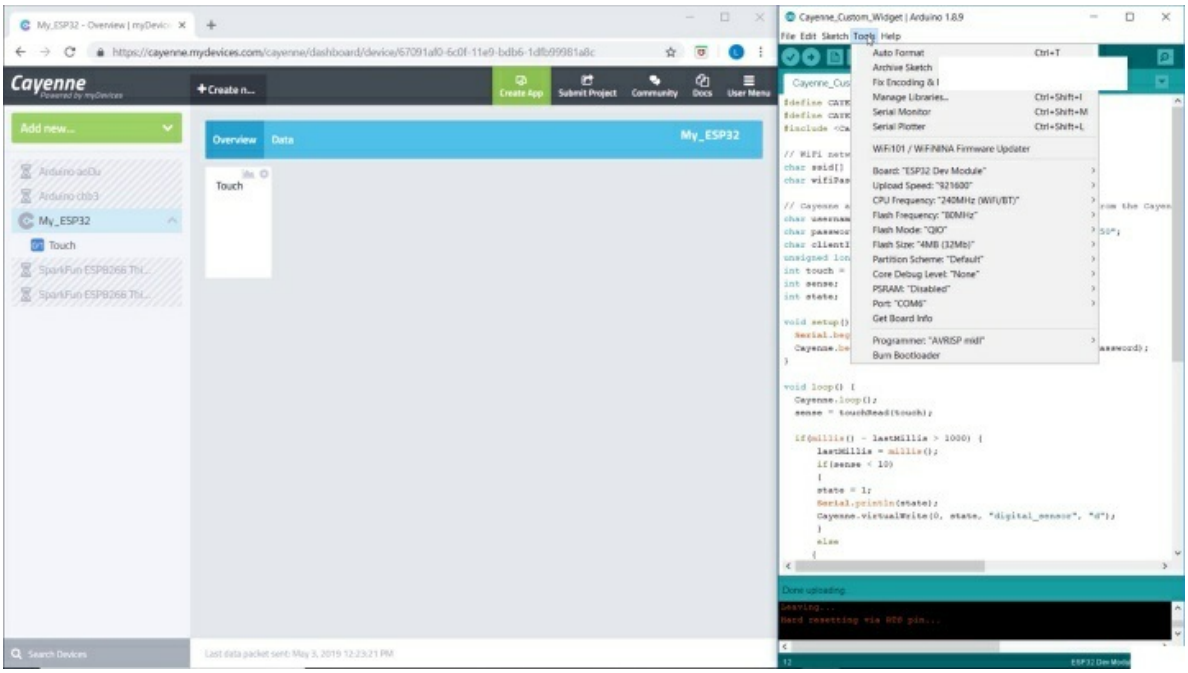
```
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "c2f2a2a0-6bd8-11e9-af96-af70e905ba69";
unsigned long lastMillis = 0;
int touch = 15;
int sense;
int state;

void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
}

void loop() {
  Cayenne.loop();
  sense = touchRead(touch);

  if(millis() - lastMillis > 1000) {
      lastMillis = millis();
      if(sense < 10)
      {
      state = 1;
      Serial.println(state);
      Cayenne.virtualWrite(0, state, "digital_sensor", "d");
      }
```

Then we initialize these variables which we will be using later in the code Now in the loop code, we are using the function touch Read to read analog

values from pin 15 and assign it to the variable sense

```
if(millis() - lastMillis > 1000) {
    lastMillis = millis();
    if(sense < 10)
    {
    state = 1;
    Serial.println(state);
    Cayenne.virtualWrite(0, state, "digital_sensor", "d");
    }
    else
   {
     state = 0;
     Serial.println(0);
     Cayenne.virtualWrite(0, state, "digital_sensor", "d");
   }
 }
```

Now inside the if condition for publishing messages every second, we are checking, if the touch reading drops below 10 the state is assigned 1 Next we print this state value to the Serial Monitor In the virtual Write function we specify the channel number as 0 the state in the value field, the type value as digital sensor and the unit value as d Again, you can refer to the Data types for the Cayenne MQTT API for the list of type values and Unit Values If the touch pin reading is above 10, it means that no touch was detected and the state is assigned as 0 and similarly printed to the Serial monitor and written to the Channel 0

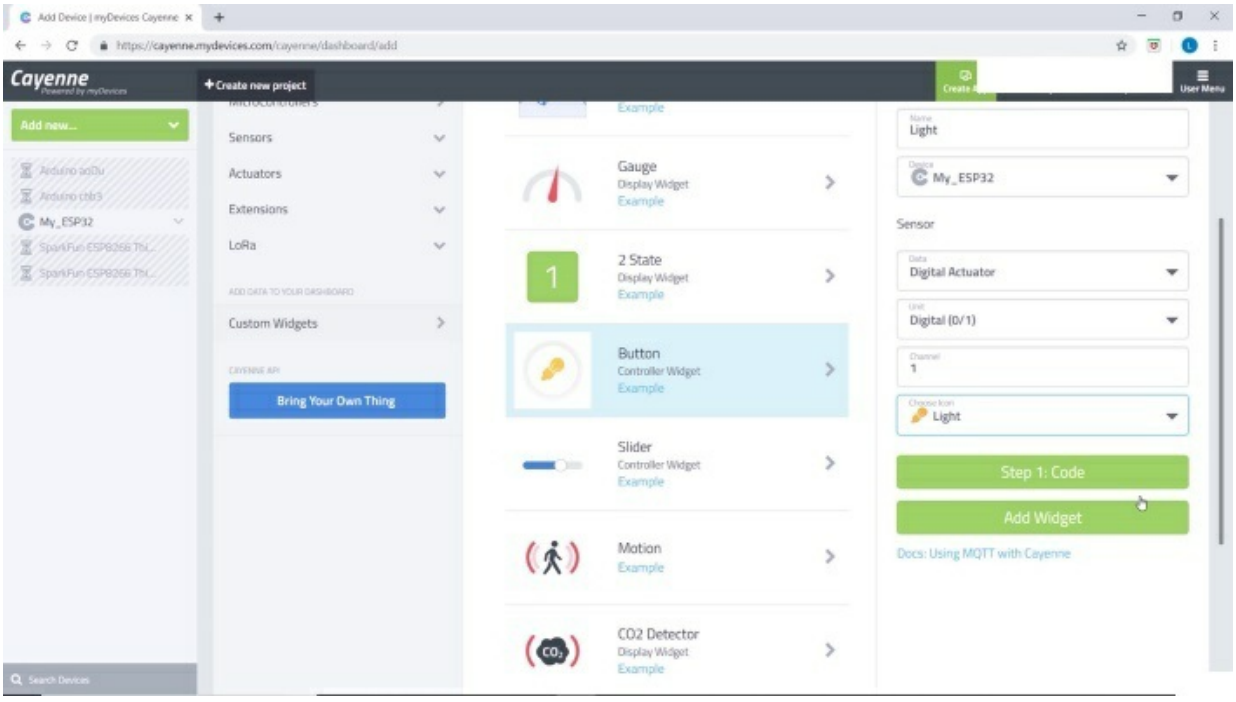


Now upload the code and open up the Serial Monitor You will see the state value and the Debug messages

Now if you touch PIN15 on the Thing, you will be able to see the state change to 1 In this, we learned how to set our MQTT message publishing rate.

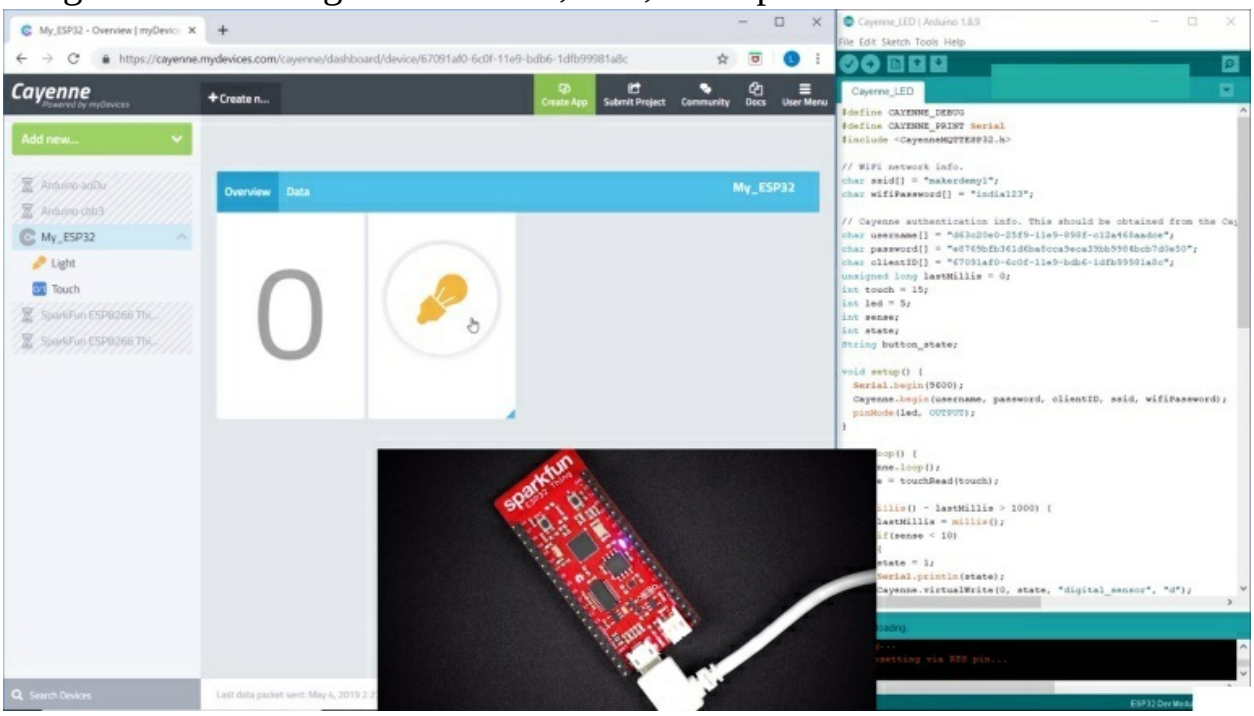
ACTUATING THE ONBOARD LED AND USING TRIGGERS

So now that we have learned how to create a custom widget and read data from the onboard touch sensor pin, let's look at how we can control an actuator from the cayenne dashboard First, let's go to the Cayenne dashboard Now go to Add new and Device/Widget Currently, only the Arduino, ESP8266 Thing and the Generic ESP8266 are supported in Cayenne by default and not the Spark fun ESP32 Thing Hence we can only use the custom widgets, However, the custom widgets offer the same functionality as the default ones



Now in the custom widgets lets select the button widget It is a controller type widget All controller type widgets can be used for actuating devices Now in the settings lets enter the name Now for the data type there is only one option which is the digital actuator type This is because the button will output 1 when switched on and 0 when switched off The unit value will be digital 0/1 Now enter the channel number as 1 This is because we had already used channel 0 for the touch sensor widget Now in choose icon, lets select light

This will be displayed on the widgets list under your device Now add the widget Before diving into the code, first, let's upload it



Now in the cayenne dashboard click on the light widget You will see the onboard LED on the Thing light up If you click again, the onboard LED will turn off Now let's look at the code to understand how to process the actuator commands from the button widget and turn the LED ON and OFF The initial code is pretty much the same

Cayenne_LED

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "67091af0-6c0f-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;
int touch = 15;
int led = 5;
int sense;
int state;
String button_state;
```

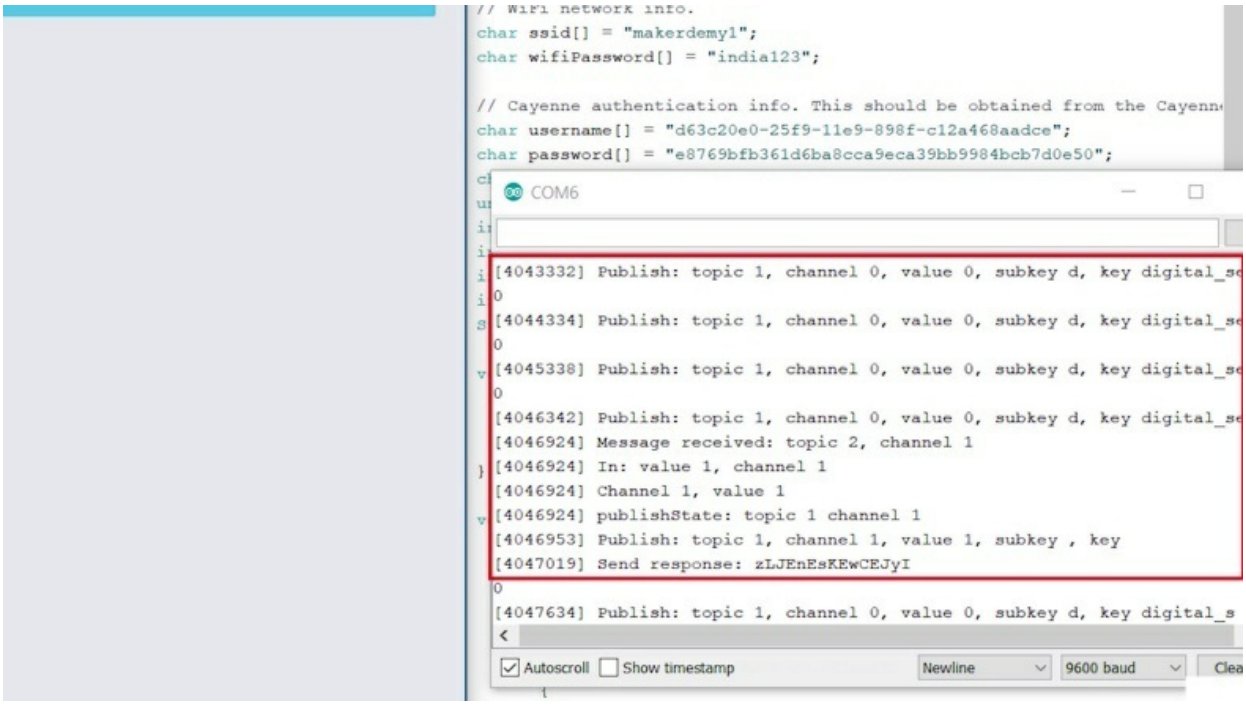
However, we need to add some new variables Here since we are using the onboard LED, we assign the LED variable as Pin 5 Also, we create a new string variable to store the state of the button widget Now in the setup code since we are using the onboard LED as the output, hence we specify it here

```
int led = 5;
int sense;
int state;
String button_state;

void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
  pinMode(led, OUTPUT);
}

void loop() {
  Cayenne.loop();
  sense = touchRead(touch);

  if(millis() - lastMillis > 1000) {
      lastMillis = millis();
      if(sense < 10)
      {
      state = 1;
      Serial.println(state);
      Cayenne.virtualWrite(0, state, "digital_sensor", "d");
      }
```

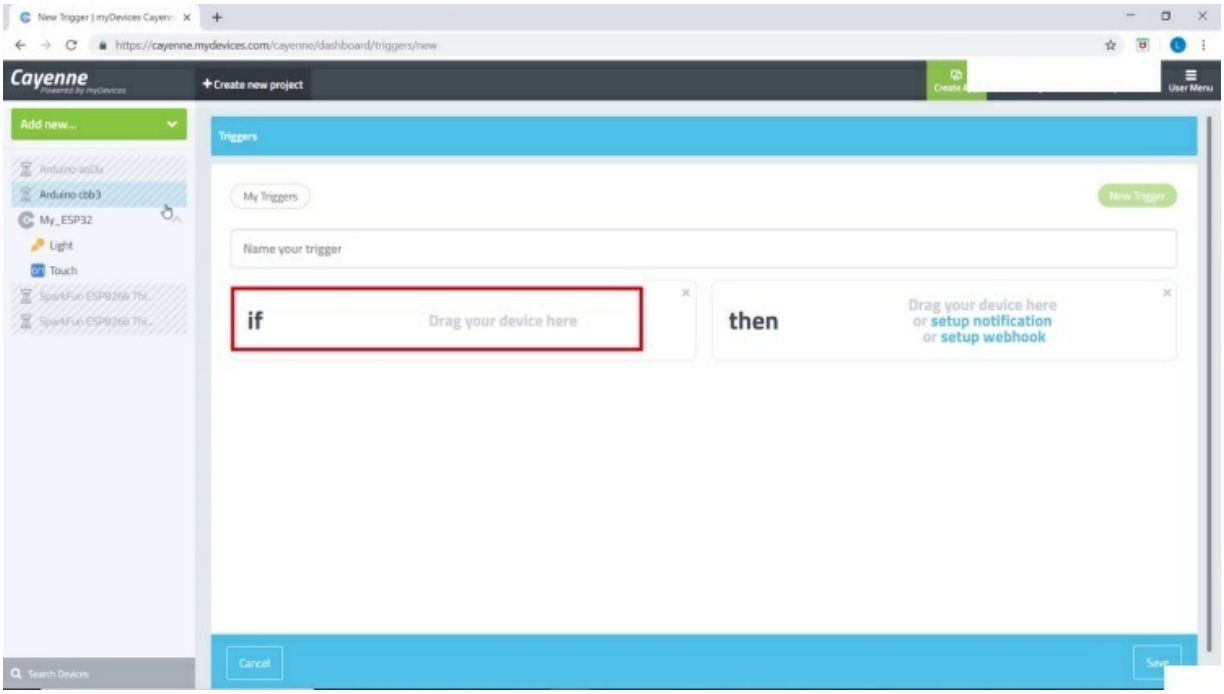
Now we do not change the loop code here as we still will be using the touch widget we had created earlier

```
  //Cayenne.virtualWrite(3, 50, TYPE_PROXIMITY, UNIT_CENTIMETER);
}

// Default function for processing actuator commands from the Cayenne Dashboard.
// You can also use functions for specific channels, e.g CAYENNE_IN(1) for channel 1 commands.
CAYENNE_IN_DEFAULT()
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
 //Process message here. If there is an error set an error message using getValue.setError(), e.g ge
}

CAYENNE_IN(1)
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  button_state = getValue.asString();
  if (button_state == "1"){
    digitalWrite(led, HIGH);
  }
  else
  {
    digitalWrite(led, LOW);
    }
 //Process message here. If there is an error set an error message using getValue.setError(), e.g ge
}
```

Now, the Cayenne in default function is where you can receive the actuator commands from the cayenne dashboard However if you want to receive actuator commands from specific channels, you can do so using the Cayenne in function You have to specify which channel number you need to receive commands from here The cayenne log function here will get the channel number and the actual value in the received actuator command and print it to the Serial Monitor Now the get value as string function will get the value in the received command and convert it to a string value This is then stored in the button state variable Now if we click on the button widget on the dashboard, the received command will be 1 If we again click on the button widget on the dashboard the received command will be 0 Here we check, if the button state value is 1 then we switch on the onboard LED Else, we switch it off This is how the actuator commands in Cayenne can be processed in code

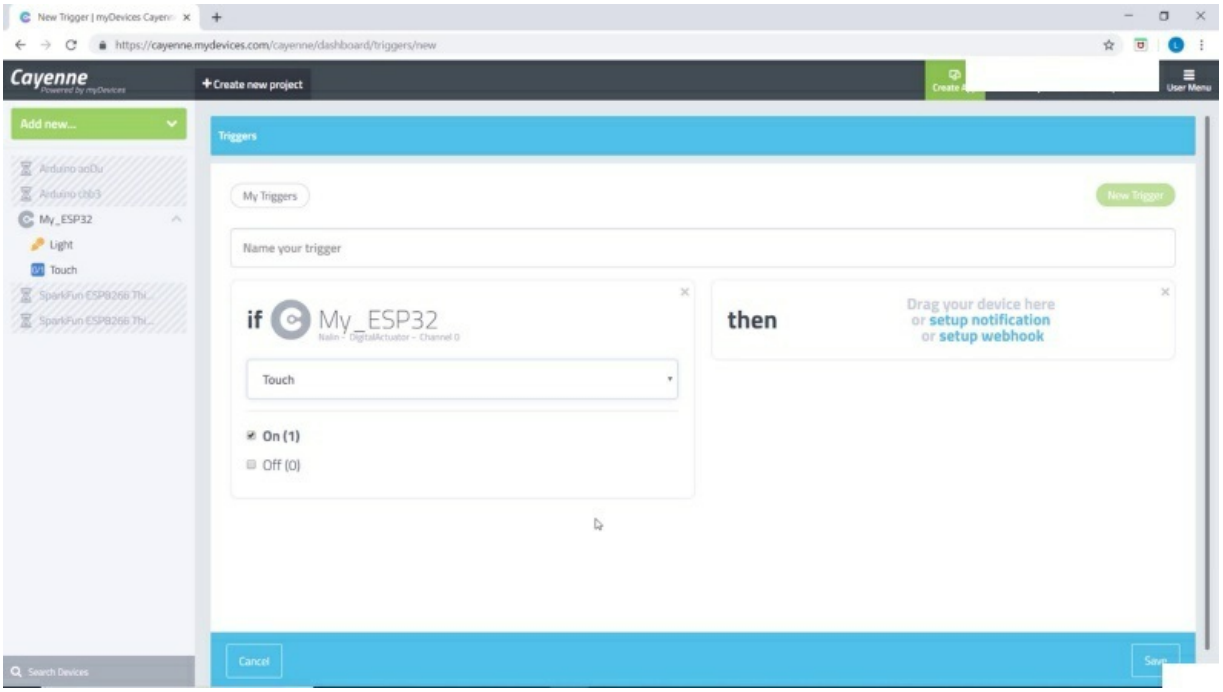


Now if you open up the Serial Monitor, and click on the widget again, you will see the Debug messages which include the Channel number, Topic, and the Value received Now let's look at the next feature that is Triggers in Cayenne Now go to Add new and select Trigger Here you can see there are two Fields if and then

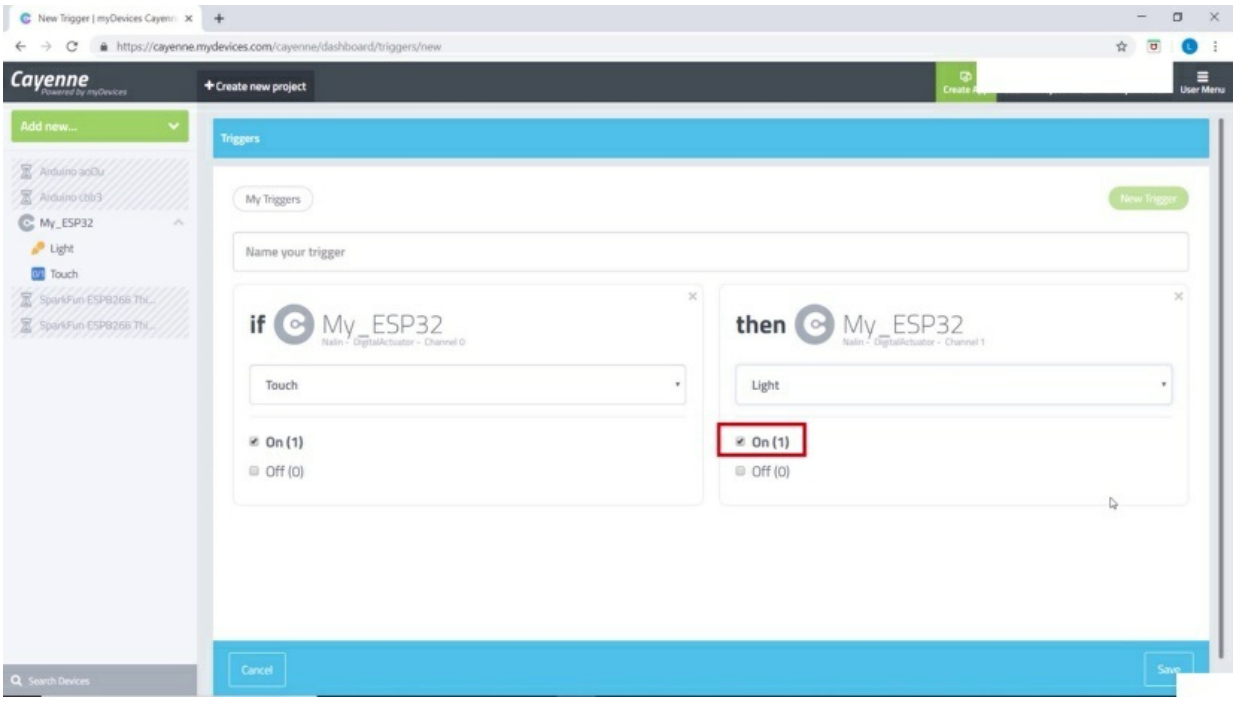


So if some condition is true here, then this will execute It can be compared to

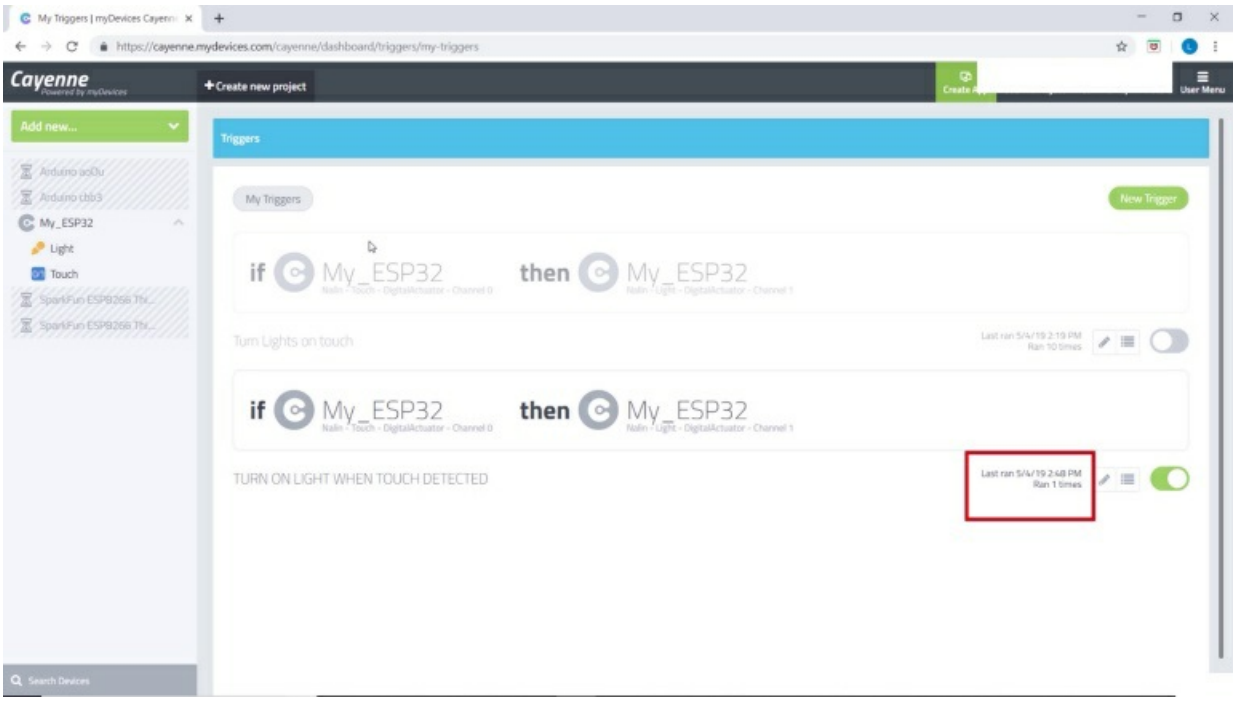
the If statement while writing code Which makes the trigger feature convenient as it essentially replaces lines of code with this simple drag and drop format Now let's drag our device, which is the Thing right here Now let select the trigger, and then the touch widget We can set the trigger as either ON or OFF, depending on whether we have the touch sensor state as 1 or 0 We select On here



So if touch is detected, then it will execute this part Now again drag the same device and select the action which you want to be executed when the touch is detected let's select the light widget here We select On here, which means that when touch is detected, the button state will become 1 and the light will turn ON



Now name the trigger anything you want Now click on save here This will save your trigger Now in the My triggers page, you can see all your created triggers You can also turn the trigger ON or OFF Now let's see the trigger in action Click on the device here Now if you touch PIN 15 on the Thing, you will see the touch state turn to 1 which will trigger the light state to turn ON and the onboard LED to glow Now go to the Triggers page and go to My Triggers

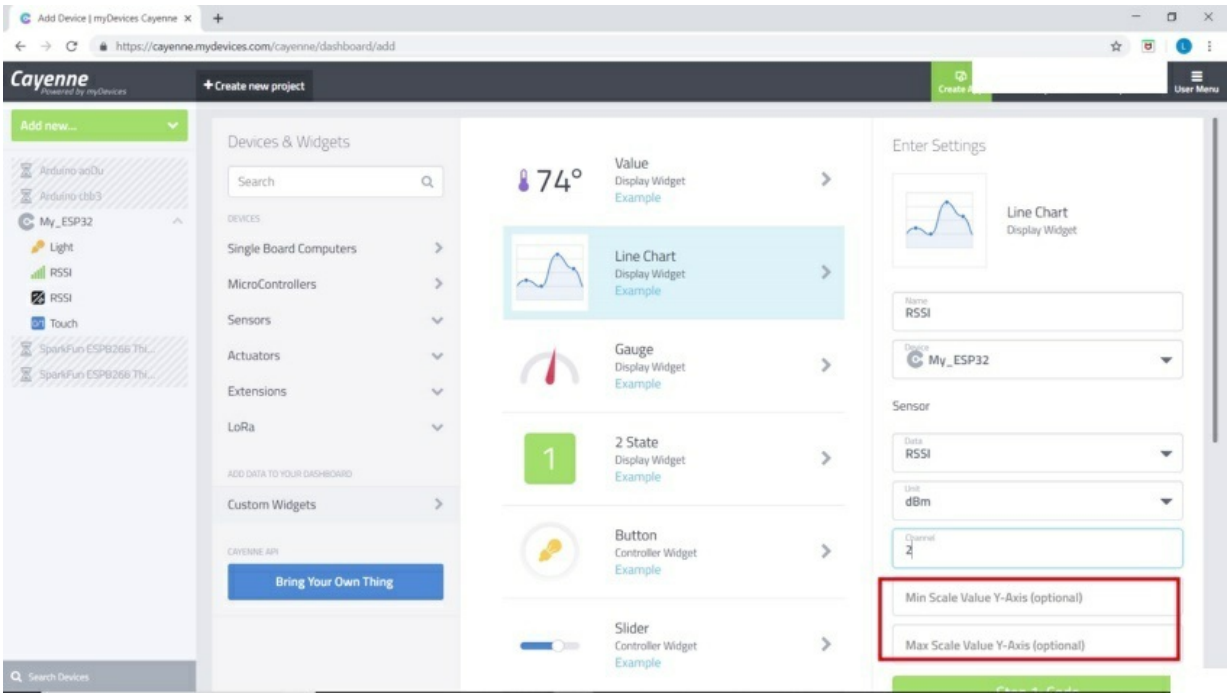


You can see here the number of times the Triggers have been executed and the time and date at which the last trigger was executed.

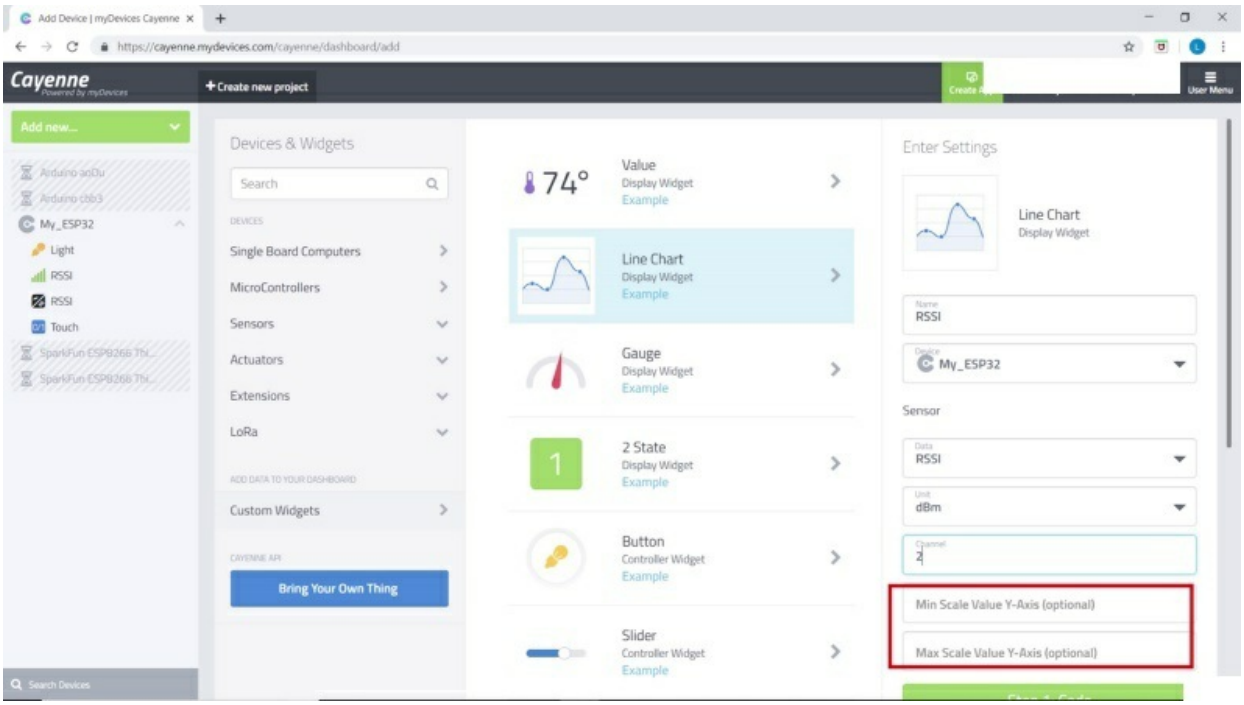
USING TRIGGER NOTIFICATION AND SCHEDULING

Creating Display widgets and using the notification trigger Scheduling Events Now lets further explore the Trigger feature Triggers can also be used to notify once a certain condition becomes true Let's look at how we can use this notification feature

We will be monitoring the signal strength with the phone's hotspot as the access point If it drops below a certain threshold, we will get an Email notifying us of the same Now go to Add new and Devices/widget In the custom widgets



let's use the display widgets are a great way to visualize your data First, we would like to have the Value of the signal strength displayed Hence we use the Value widget Now enter the name Enter the data type as RSSI since we will be receiving RSSI values as data Select the unit as dBm, channel number as 2 since we have already used channel number 0 for touch and channel number 1 for Lights Choose the icon for signal strength Now add the widget Now let's use the Gauge widget has three levels Ok, Warning and Danger We will use this for letting us know whether the signal strength is strong, medium or weak Now enter the same name, data type and channel number and add the widget



We will also use the Line chart display widget It is useful for keeping a track of the past values received You can leave these 2 fields blank It will automatically adjust the scale for the incoming data Now add this widget to your dashboard the channel numbers for all the 3 widgets must be the same that is channel number 2 since at this channel we will be receiving the RSSI values

Cayenne_RSSI

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>
#include <WiFi.h>

// WiFi network info.
char ssid[] = "FroyoPlasma";
char wifiPassword[] = "india123";

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "67091af0-6c0f-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;
int touch = 15;
int led = 5;
int sense;
int state;
String button_state;
```

Now let's jump to the code The code has some additions from the one seen in the Here we have included the Wifi library This is to use the RSSI function The initial code is the same as the previous one No change here We are using the phone's hotspot as the access point here, hence we need to enter the hotspot credentials

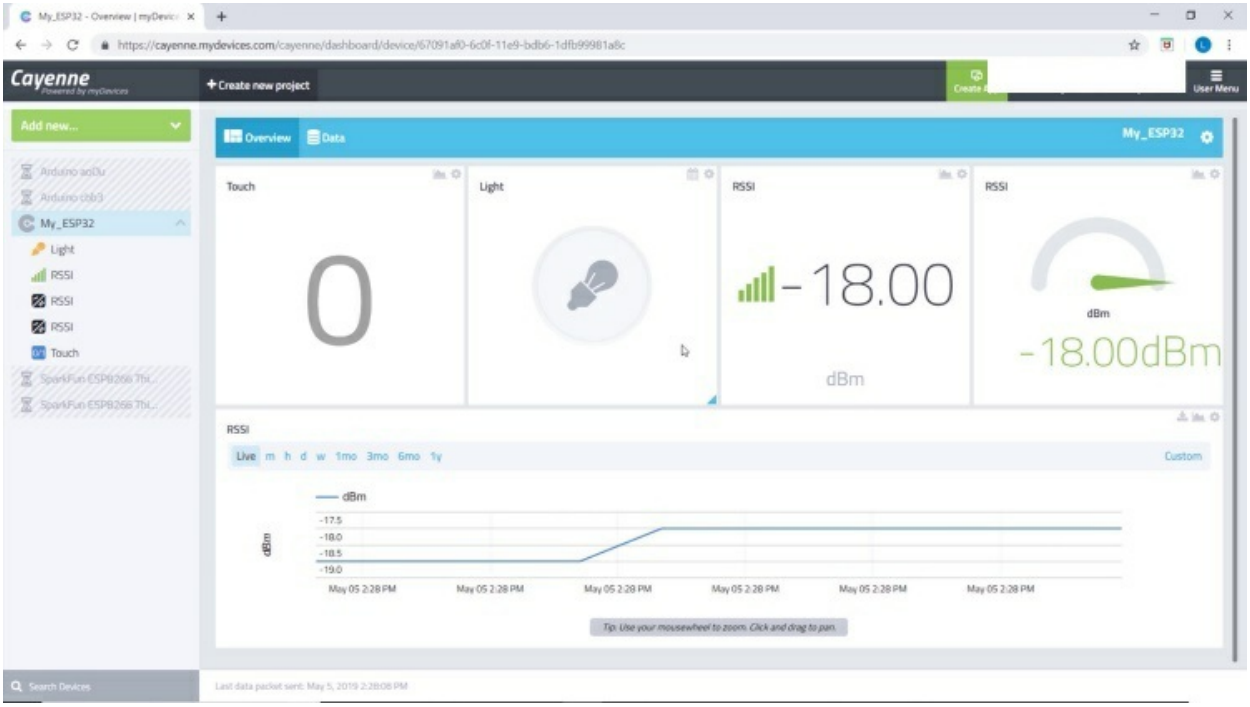
```
String button_state;

void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
  pinMode(led, OUTPUT);
}

void loop() {
  Cayenne.loop();
  long rssi = WiFi.RSSI();
  Serial.println("RSSI:");
  Serial.println(rssi);
  sense = touchRead(touch);

  if(millis() - lastMillis > 1500) {
      lastMillis = millis();
      if(sense < 10)
      {
```

here In the loop code, we are first reading the RSSI value using the Wifi RSSI function and store it in this variable Now we print the RSSI values to the Serial monitor

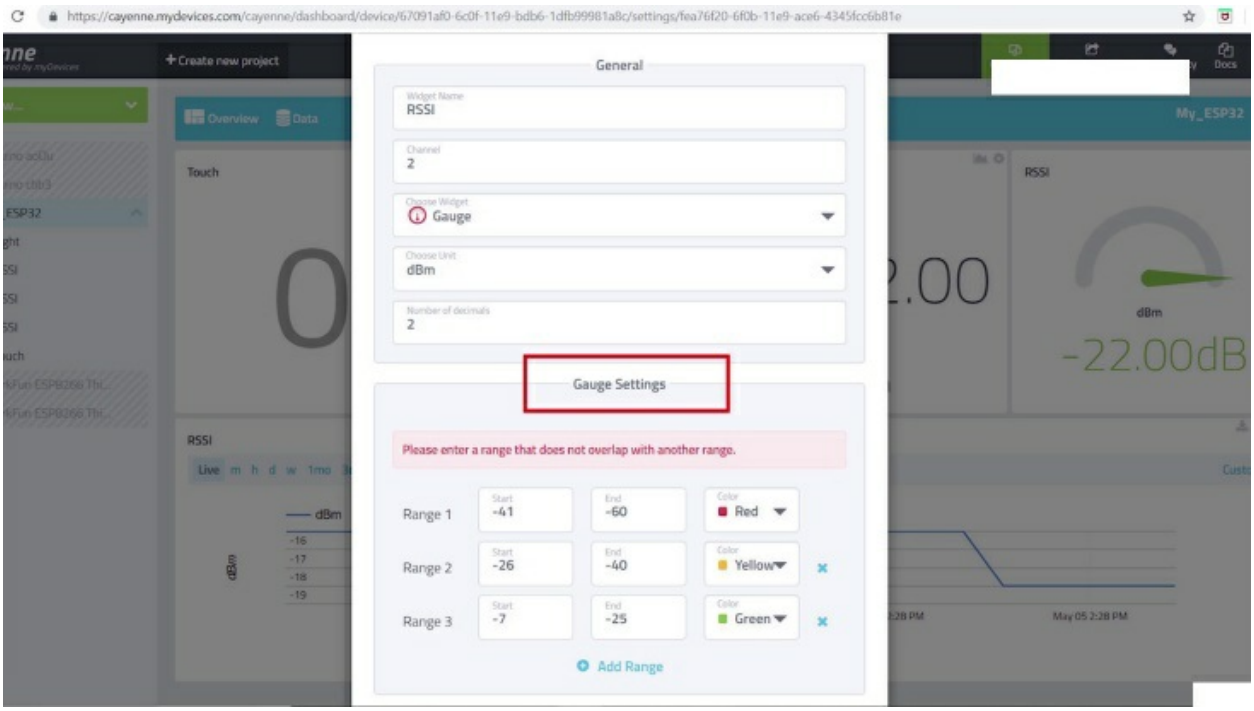
However since we need to write this RSSI data to the Cayenne channel, we use the virtualWrite function here We specify the same channel number as 2, the Value as RSSI, the data type as RSSI, and unit as dBm

This virtualWrite function is used inside the custom message rate loop we had created earlier Now here we have set the message rate as 1 message every 1 5 seconds We have increased the delay time from 1 second to 1 5

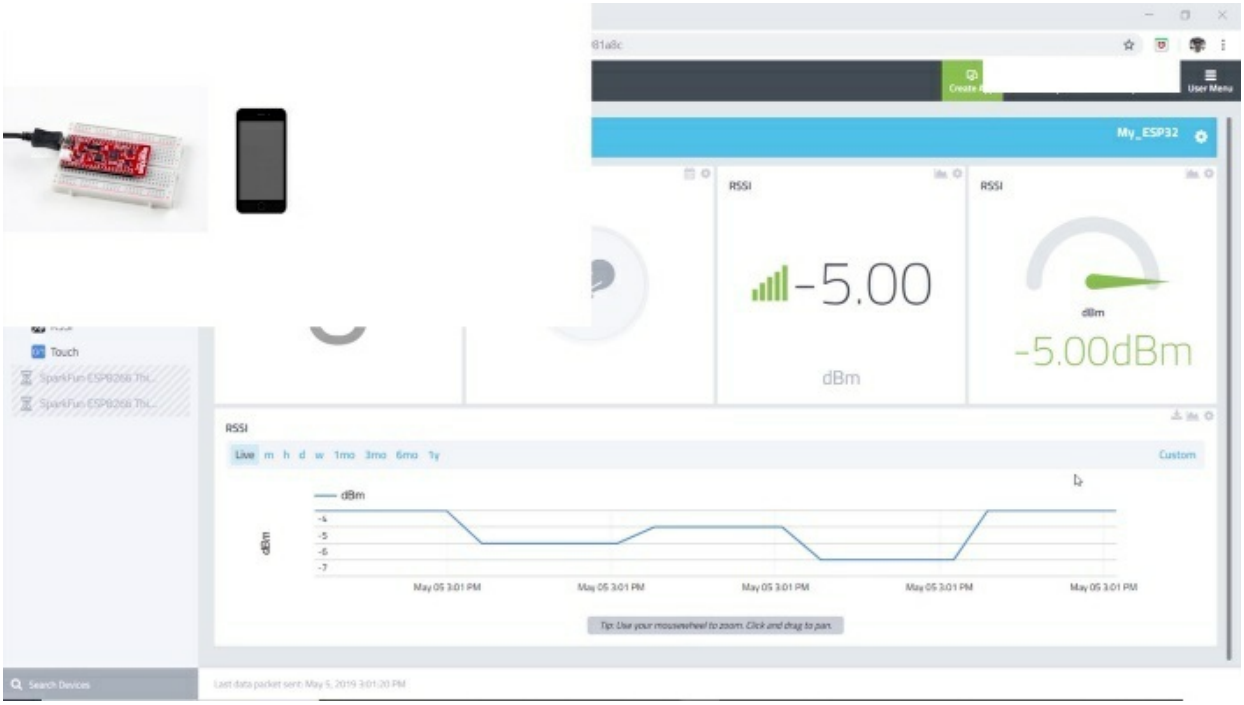
seconds because we are sending two messages simultaneously, one for the Touch sensor pin state and one for the RSSI Hence setting the message rate lower than this will cause messages to be dropped and frequent disconnections Now the rest of the code is the same as seen in



Let's upload the code now and go to the dashboard You can see the RSSI values being displayed For the gauge widget, we need to specify the range for the three levels

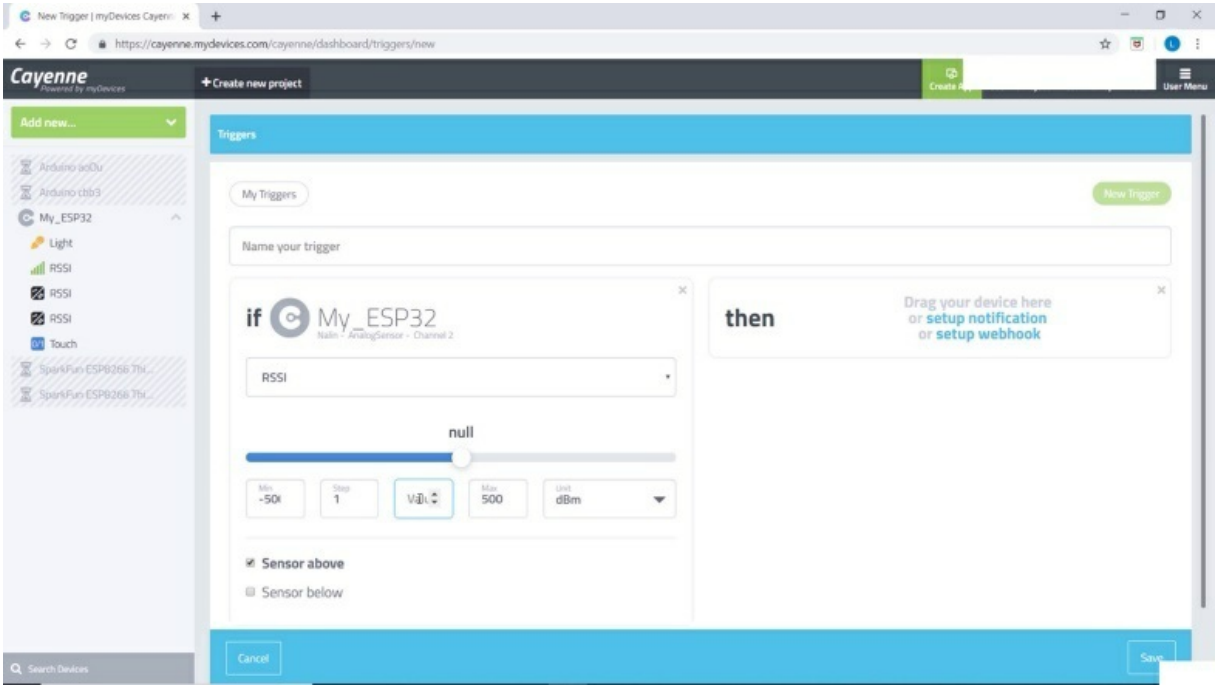


Go to the Widget settings here You can see the settings we had configured earlier while adding the widget Here you can specify the range of the values you want to be displayed as Green, red, and Yellow Since the higher the RSSI value, the stronger the signal strength, hence we have specified it in this order

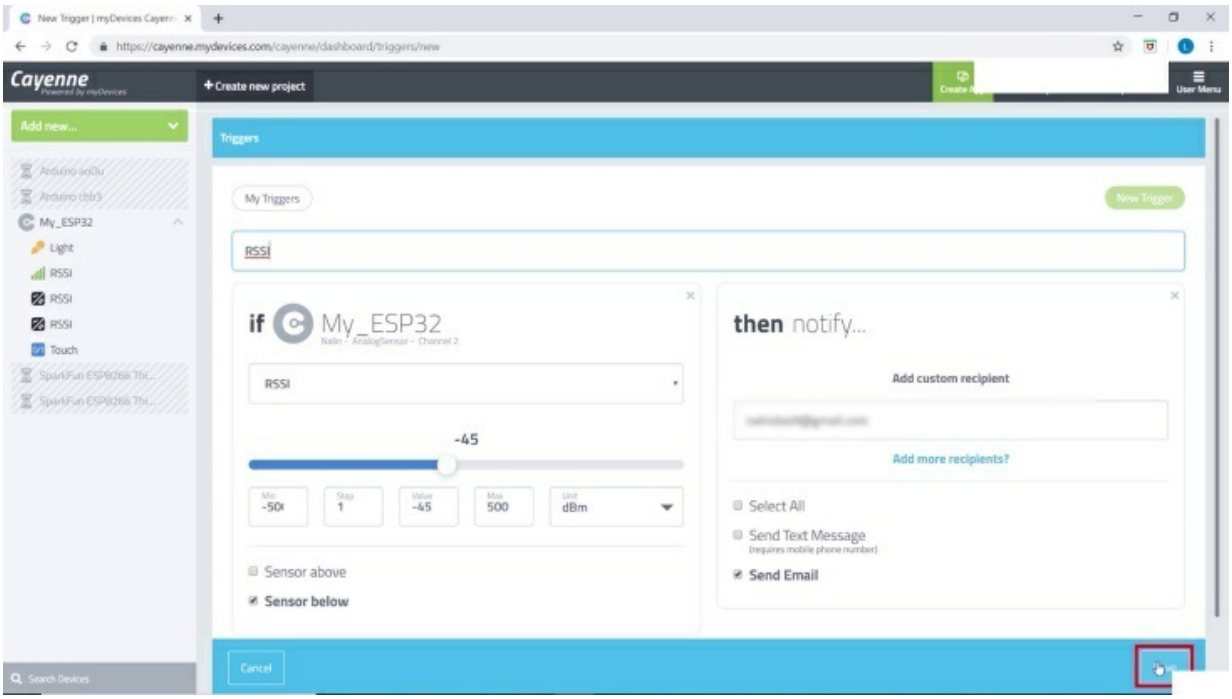


Here we can see that when the phone is near to the Thing, the RSSI value is

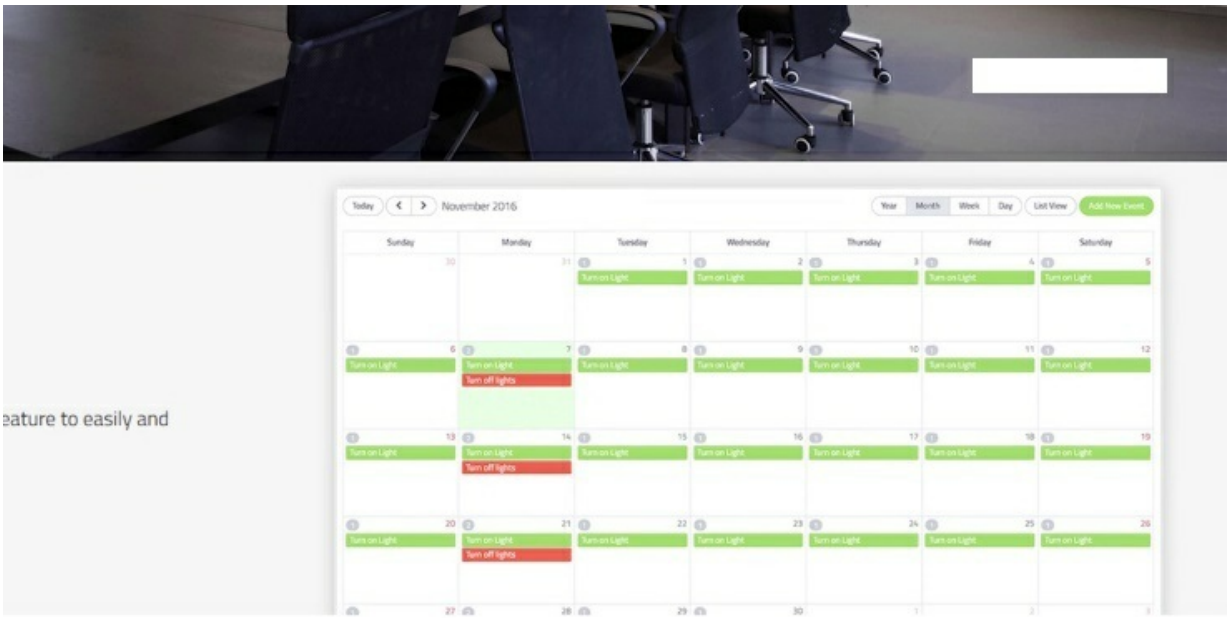
pretty good Now as we move it away, you will see the RSSI value start to fall Now we will set a notification trigger If the value drops below -45 dBm, we will send email Lets to see how we can do this First Then drag your device here and select RSSI value widget here as this is what we will be monitoring Now enter the threshold value as -45 dBm



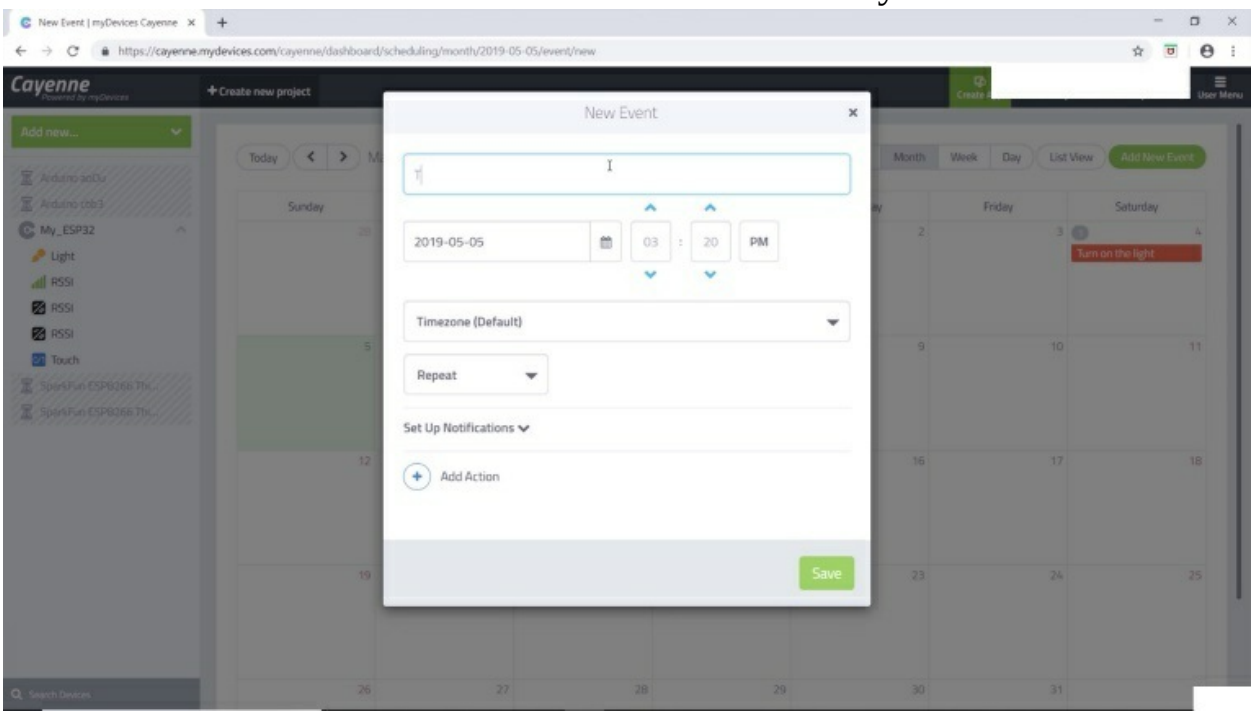
Then select the sensor below This means that when the value drops below the threshold of -45 dBm, the trigger will go off Sensor above is used if we want the trigger to go off when the value rises above the threshold



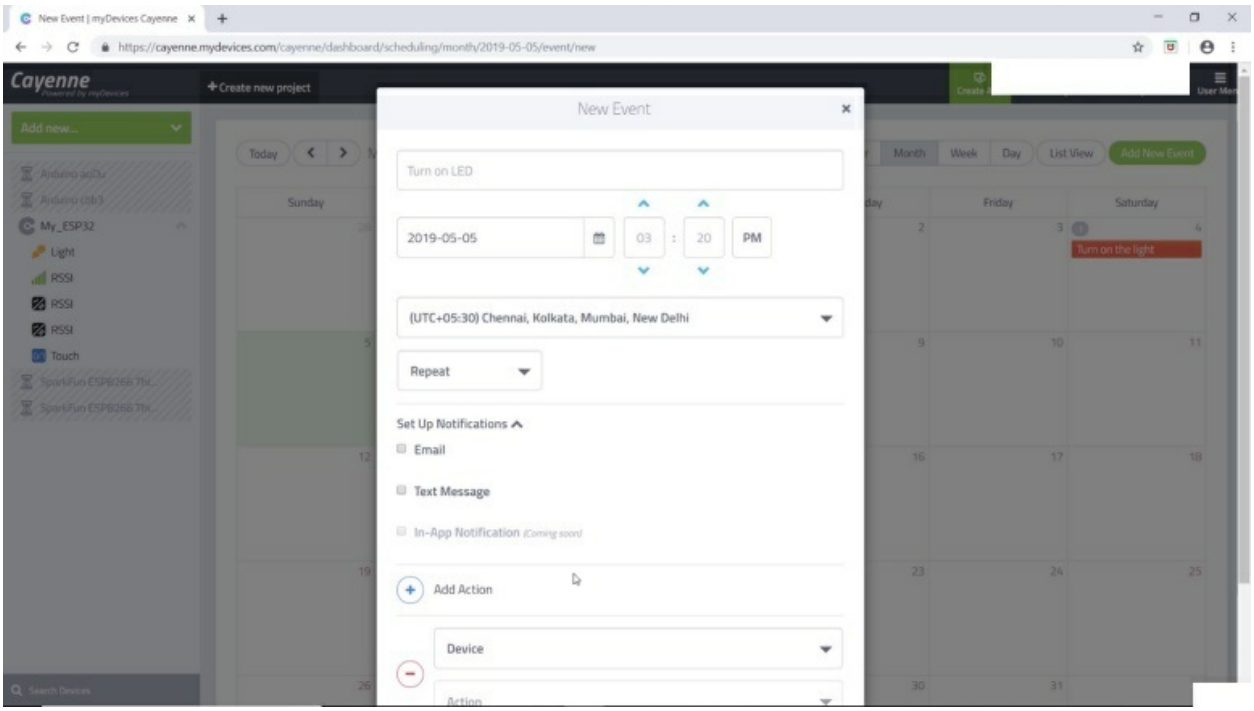
Now select setup notification in the then field and select setup notification Here you can notify text message as well as Email We will use email here Now click on add custom recipient and enter the Email ID of the recipient where you want to receive the notification email Now name the trigger and save it Now when we move the phone away so that the RSSI value drops below our set threshold of -45 dBm you will the see the notification email in your inbox



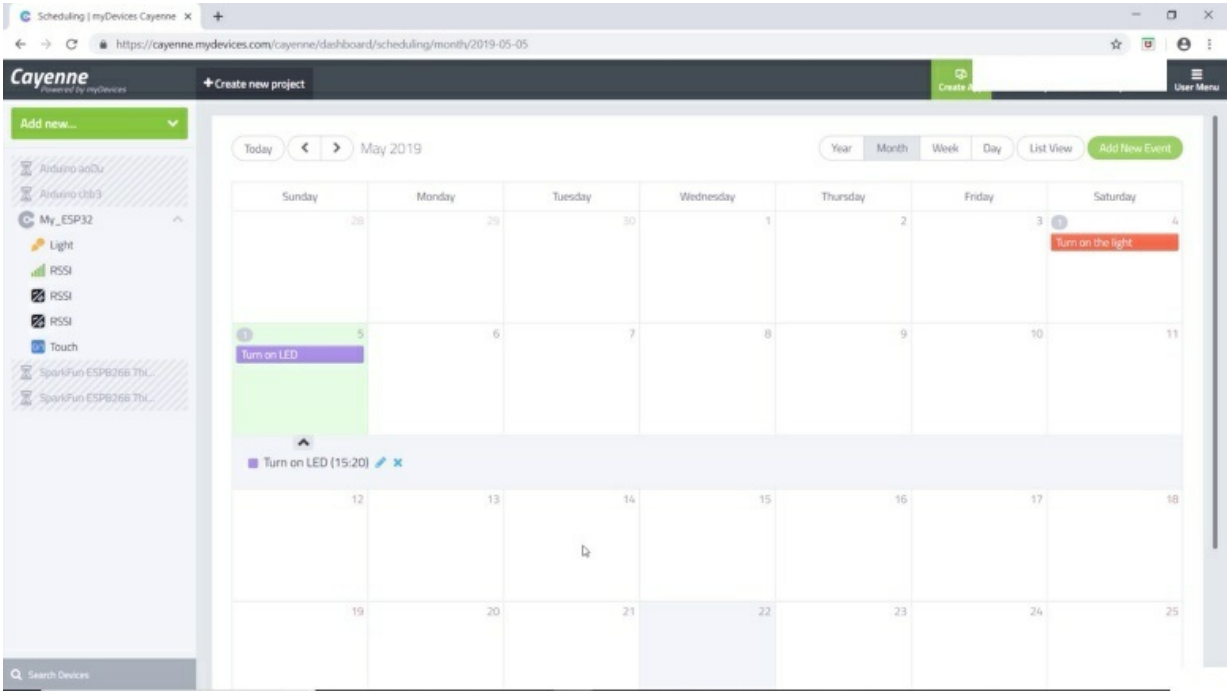
Now let's take a look at another Cayenne feature, Events Using this you can schedule an event to occur at whichever date and time you want



Let's see how we can do this Go to Add new and then Event Here you will get the Event settings Enter the Event title Then select the date at which you want the event to occur, and set the time



Here we are scheduling an event for today, 2 minutes from now Select your country's appropriate time zone You can set up notifications either send a mail or a text message on the occurrence of the event Now add the action which you want to take place Here since we want the LED to glow, select light and the action as Turn on



Now you can see that the event has been scheduled for the desired date and

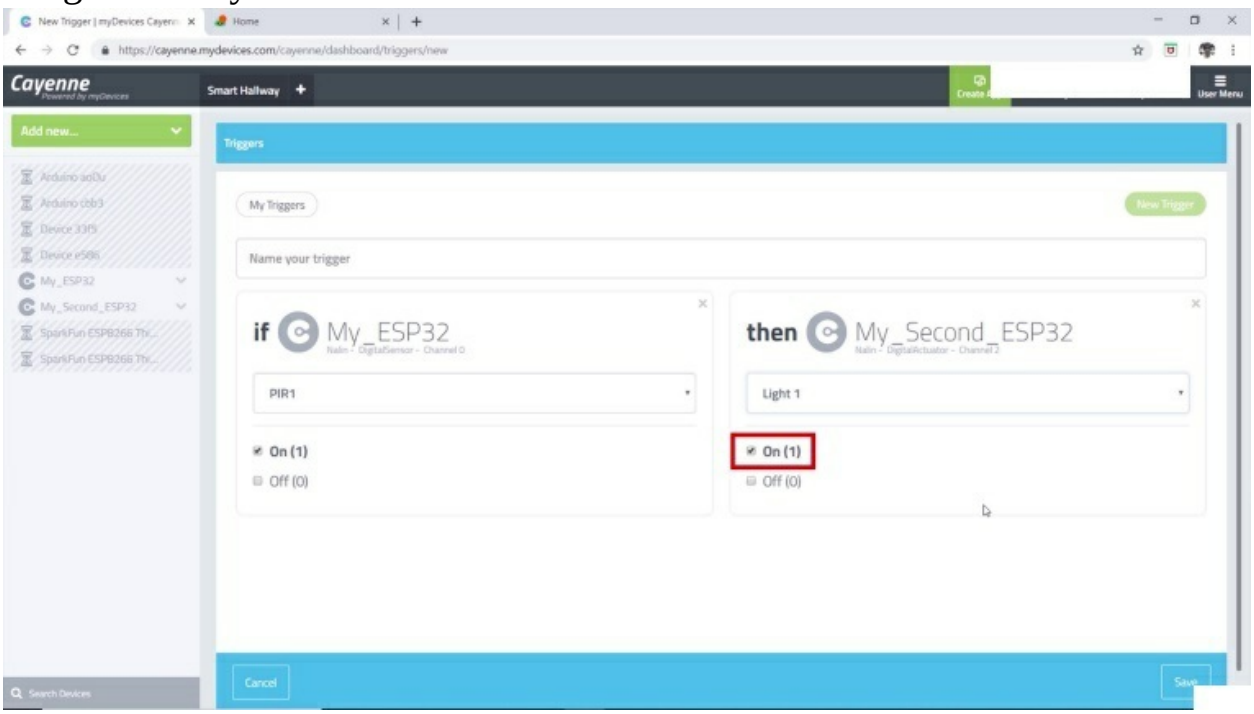
time Now if we go to the dashboard, you will see that the onboard LED will glow at the time we had set In this project, we learned how to create display widgets and use a trigger notification We also learned how to schedule an event In this section, we covered the following Introduction to MQTT Interfacing the Thing t Cayenne Setting the message rate and creating a Custom widget Actuating the Onboard LED and Using Triggers.

INTERFACING PIR SENSORS WITH THE THING

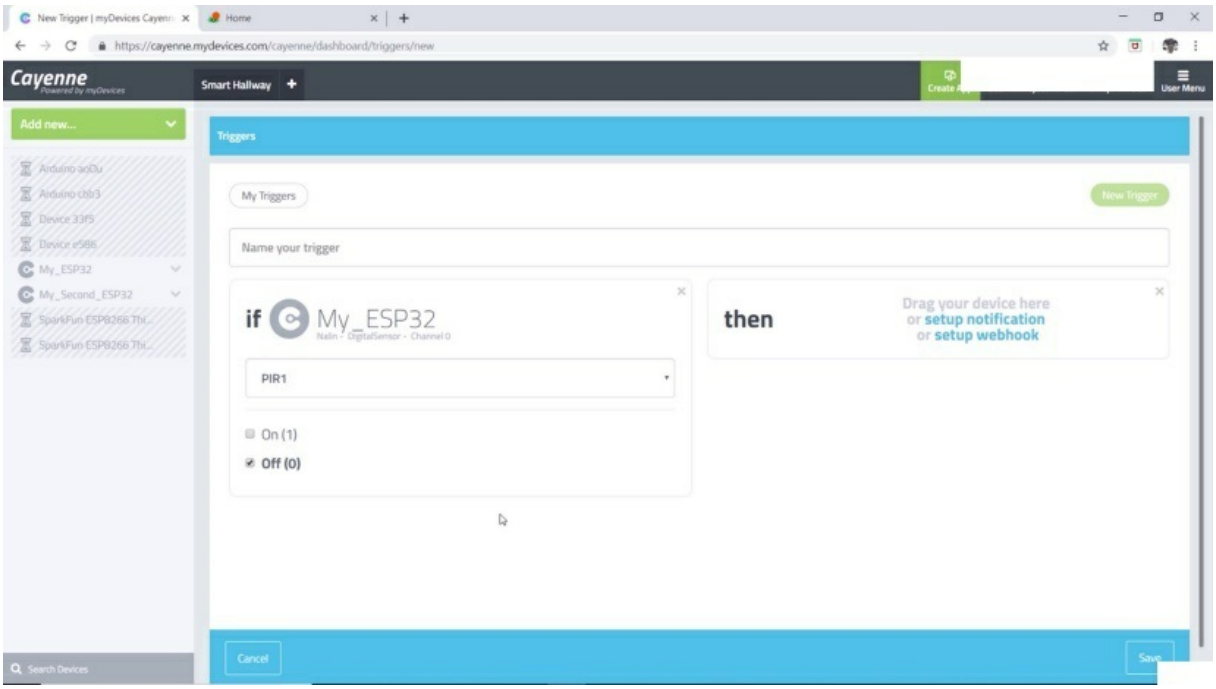
We will learn the following Setup Triggers for the two PIR sensors to actuate the bulbs Correct the PIR sensors series of 1s problem Lets now take a look at the third part of our project The Smart hallway is aimed at saving power Traditional hallways will have all the light bulbs ON at all times But with the smart hallway, the light will only be switched only when there is motion detected in the particular PIR sensors vicinity

So when a person walks in the hallway, the light bulbs will be switched ON only in the persons proximity in that project this means that when the person walks into the hallway, the first PIR sensor will detect motion and switch ON the first light bulb When the person has crossed the first PIR sensor and now enters the second PIR sensors range , the first light bulb should switch OFF and the second light bulb should switch ON Similarly, when the person crosses the range of the Second PIR sensor, the second light bulb should

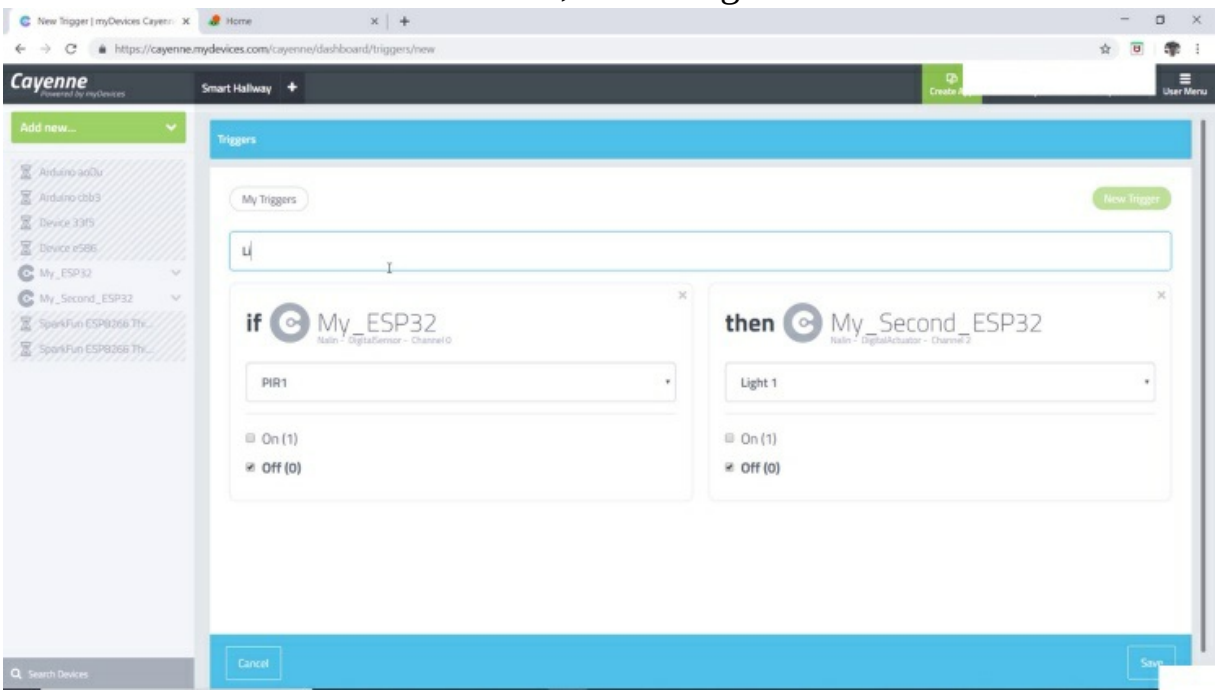
switch off Now lets look at how we can do this using Triggers on Cayenne First, go to Add new and then Trigger Once you are in the trigger menu, you need to drag the First Thing here Now select PIR1 which was the widget to detect motion for the first PIR sensor If the first PIR sensor received 1 as the value, it means it has detected motion So when it detects motion we need to trigger the 1st bulb to light up To do this, you need to drag the Second Thing here Now lets select the action you want to be triggered Select the Light 1 widget which you had created to actuate the first bulb



We need to select ON here since we want the bulb to light up Now you can name your trigger anything you want Here we will be naming it as Light 1 ON Now you can save it We need to create the next Trigger now Drag the first Thing here

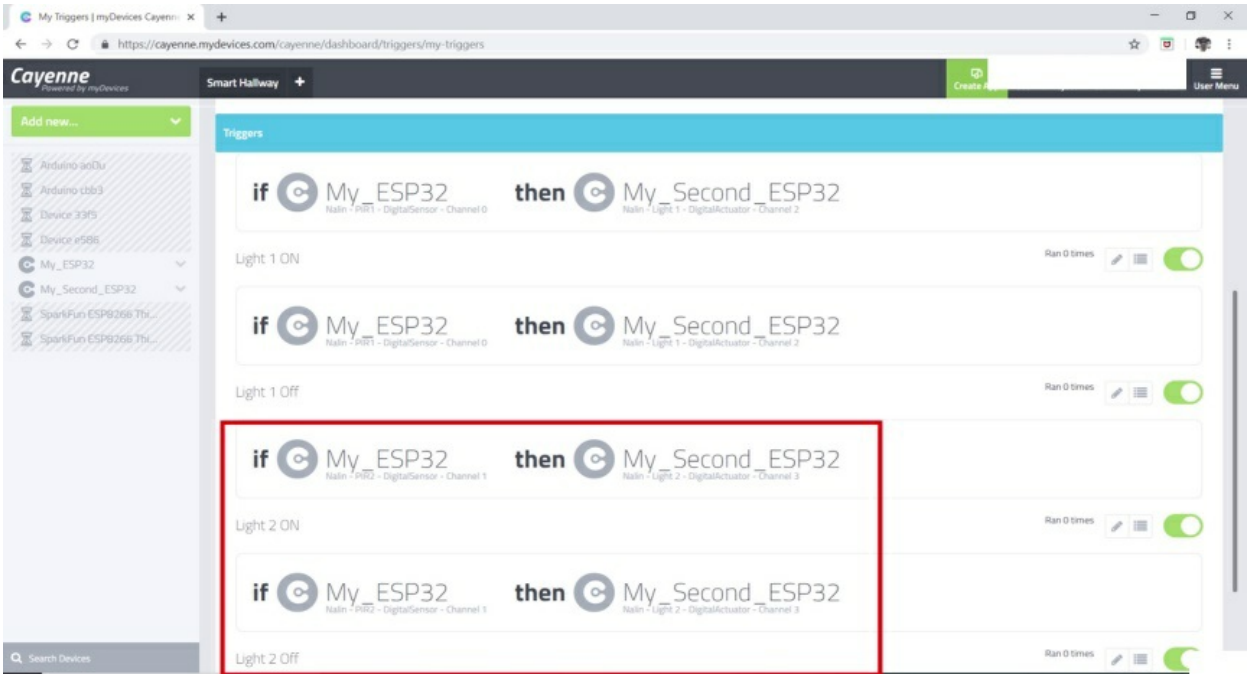


Now again, you will have to select PIR1 Now the condition to trigger will be OFF here This is because when the person has crossed, the data value received will transition from 1 to 0, indicating No motion detected



Now we need to add the action here Drag the second Thing here Now since we want to Switch OFF the first bulb, we need to select OFF here Name the Second Trigger as Light1 OFF and save it Similarly you now need to create two more triggers for the PIR2 sensor, one for the ON condition and the other

for the OFF condition



We are naming it here as Light2 ON and Light2 OFF Now that we have created our 4 triggers would think the job is done right? Not so fast You see PIR sensors are very sensitive to motion So when it detects motion, it will output a series of 1s before going back to 0 The problem with this is that on detecting motion, the PIR sensor widget will receive this series of 1s and in turn run a series of multiple unnecessary triggers To deal with this problem we have to make some changes to the PIR sensor code we had used earlier We need to alter the code so that the PIR sensor widget on detecting motion sends only a single 1 to Cayenne

Cayenne_PIR

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";
int pin1 = 25;
int pin2 = 22;
bool lastmotion1 = 0;
bool lastmotion2 = 0;
// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "67091af0-6c0f-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
   pinMode(pin1,INPUT);
  pinMode(pin2,INPUT);
```

Let's look at the code now We are initializing 2 boolean variables here lastmotion1 and lastmotion2 as 0 We will use these variables later in the code Since we do not want a series of 1s, a 1 should not be followed by another 1 A 1 should only be preceded by a 0 Hence on the loop code, we specify this condition

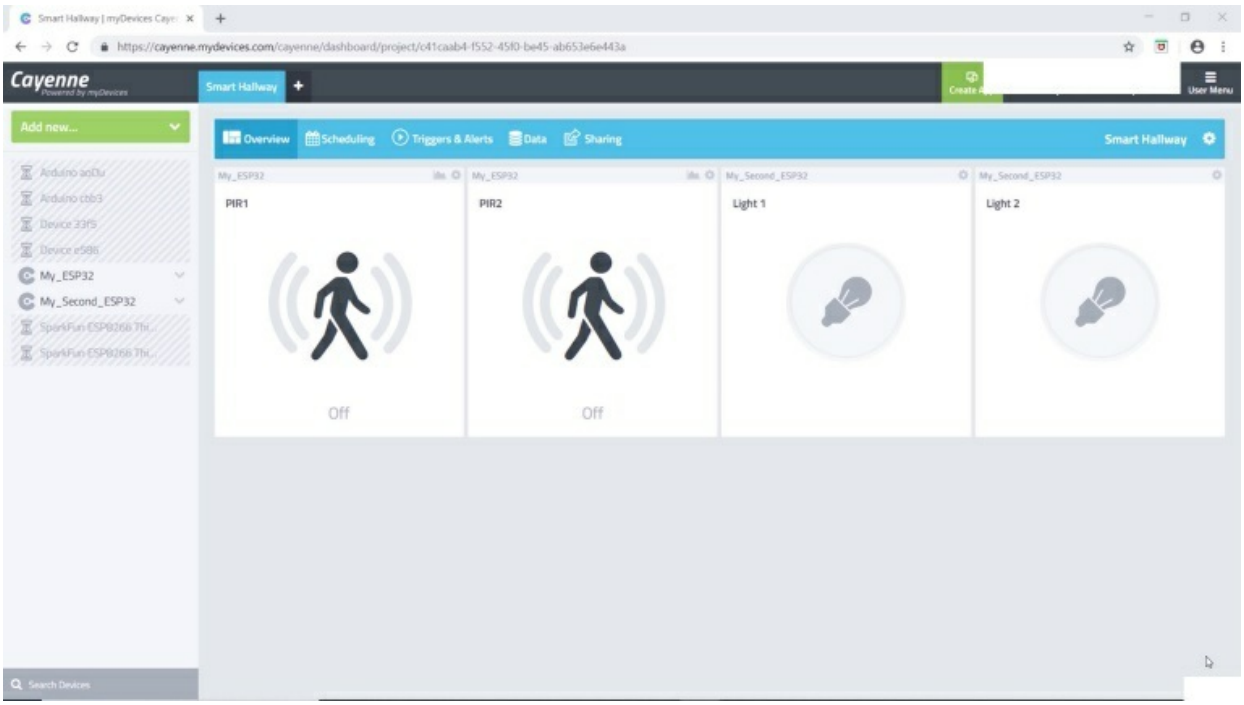
```
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "67091af0-6c0f-11e9-bdb6-1dfb99981a8c";
unsigned long lastMillis = 0;


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
   pinMode(pin1,INPUT);
  pinMode(pin2,INPUT);
}

void loop() {
  Cayenne.loop();
  bool motion1 = digitalRead(pin1);
  bool motion2= digitalRead(pin2);

  if(millis() - lastMillis > 1500) {
      lastMillis = millis();

      if((motion1 == 1) && (lastmotion1 == 0)){
            Serial.println(motion1);
            Cayenne.virtualWrite(0, motion1, "motion", "d");
      }
```

If the variables motion1 is 1 and lastmotion1 is 0, then we write our motion

value, which is 1, to the cayenne channel 0 was for the PIR1 widget if you remember The motion1 variable is used to monitor the current value of the PIR sensor 1 whereas the lastmotion1 variable is used to monitor the value read from the PIR sensor 1 just before the current one If the lastmotion1 value is 1 and the motion1 value is also 1, it means we have detected consecutive 1s and

```
    }
    else
    {
      Serial.println("0");
      Cayenne.virtualWrite(0, 0, "motion", "d");
      }
    if((motion2 == 1) && (lastmotion2 == 0)){
        Serial.println(motion2);
        Cayenne.virtualWrite(1, motion2, "motion", "d");
    }
     else
    {
      Serial.println("0");
      Cayenne.virtualWrite(1, 0, "motion", "d");
      }

     lastmotion1 = motion1;
     lastmotion2 = motion2;
}

 }

// Default function for sending sensor data at intervals to Cavenne.
```

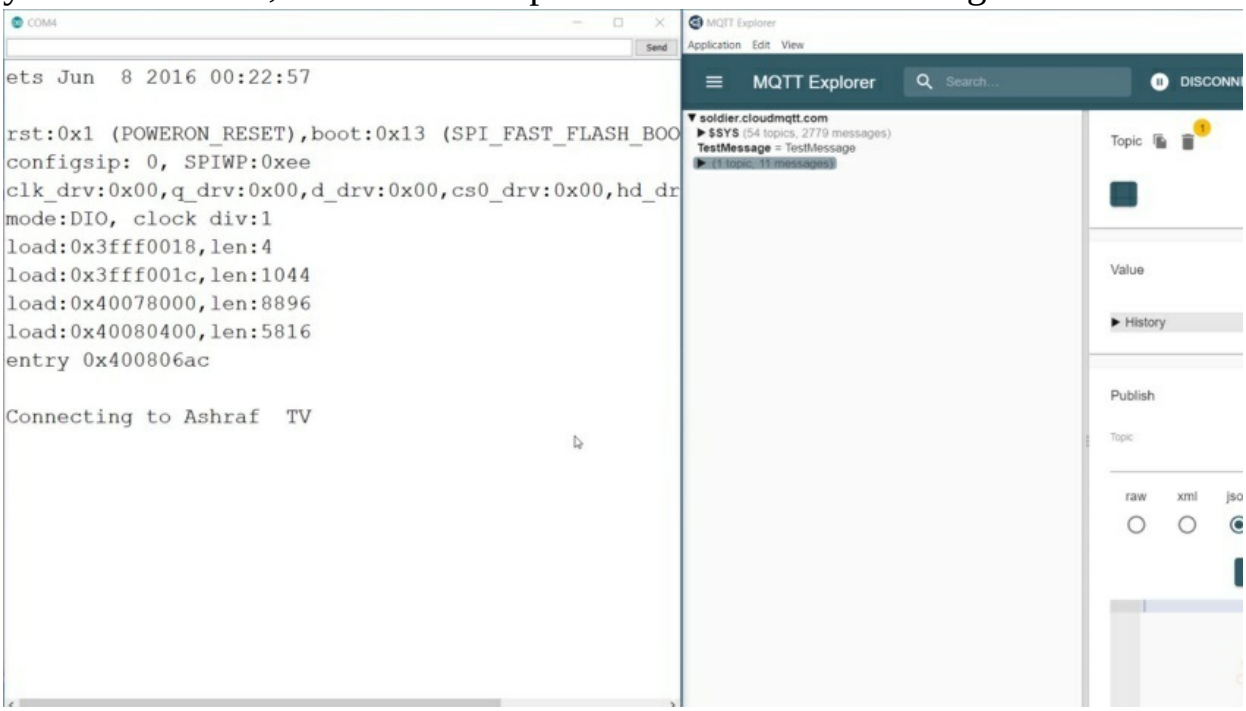
hence the else part will execute We write a zero to channel 0 in this case, meaning no motion detected We do the same for the second sensor as well Here we just store the current value from the sensors in the last motion variables so that the current value becomes the last value for the next loop iteration The rest of the code is the same as seen previously Now once you upload the code, and go to the project dashboard, you can see the setup in action



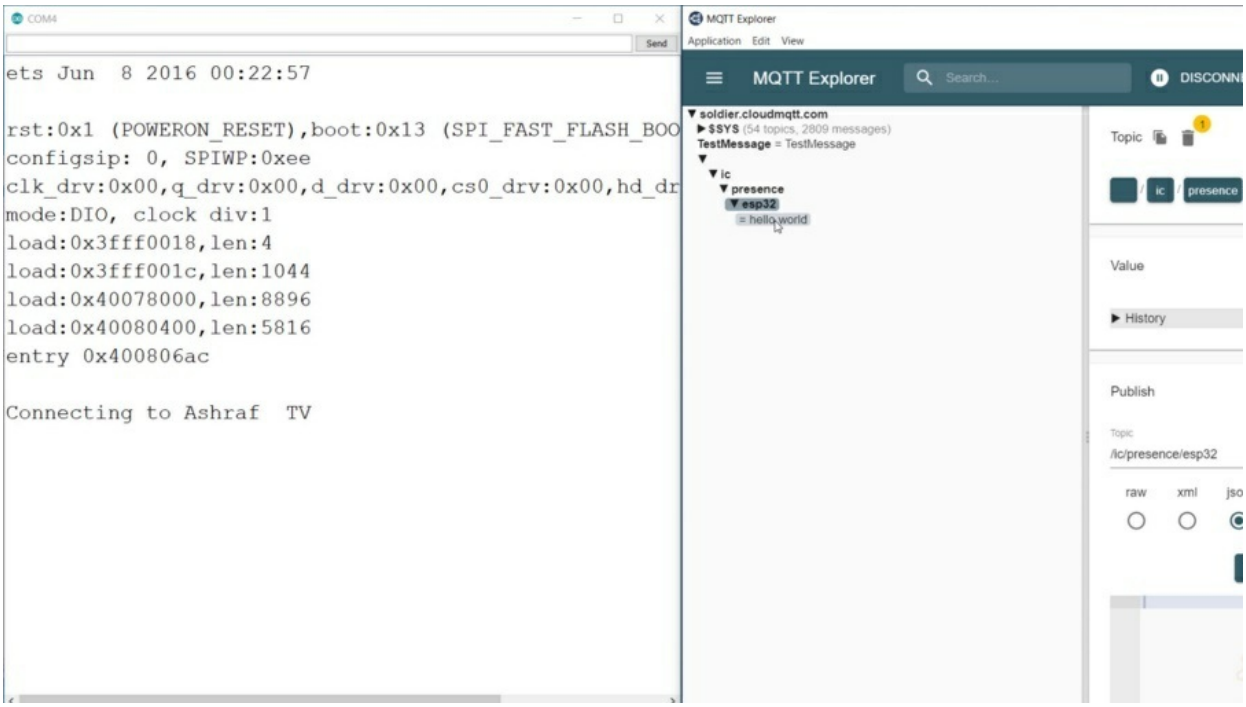
When the person walks in, the first light bulb lights up When the person crosses the first PIR sensors range, the first bulb switches OFF Similarly when the person walks into the second PIR sensors range, the second bulb lights up and then switches OFF when the person has left In this project, we learned about setting up triggers for the two PIR sensors to actuate the respective bulbs We also learned about how to correct the PIR sensors series of 1s problem.

FINAL ESP32 TESTING

Now, to make sure that our E. S. P has connected to our Cloud and Kutty server. I already mentioned that you have to dload Kutty Explorer. And as you can see here, we have one topic and about eleven messages.



And if we did open the messages, we can see that I see a presence. Yes. Thirty-two. And the message is Glow-Worm torches. The message that we did send using our code.

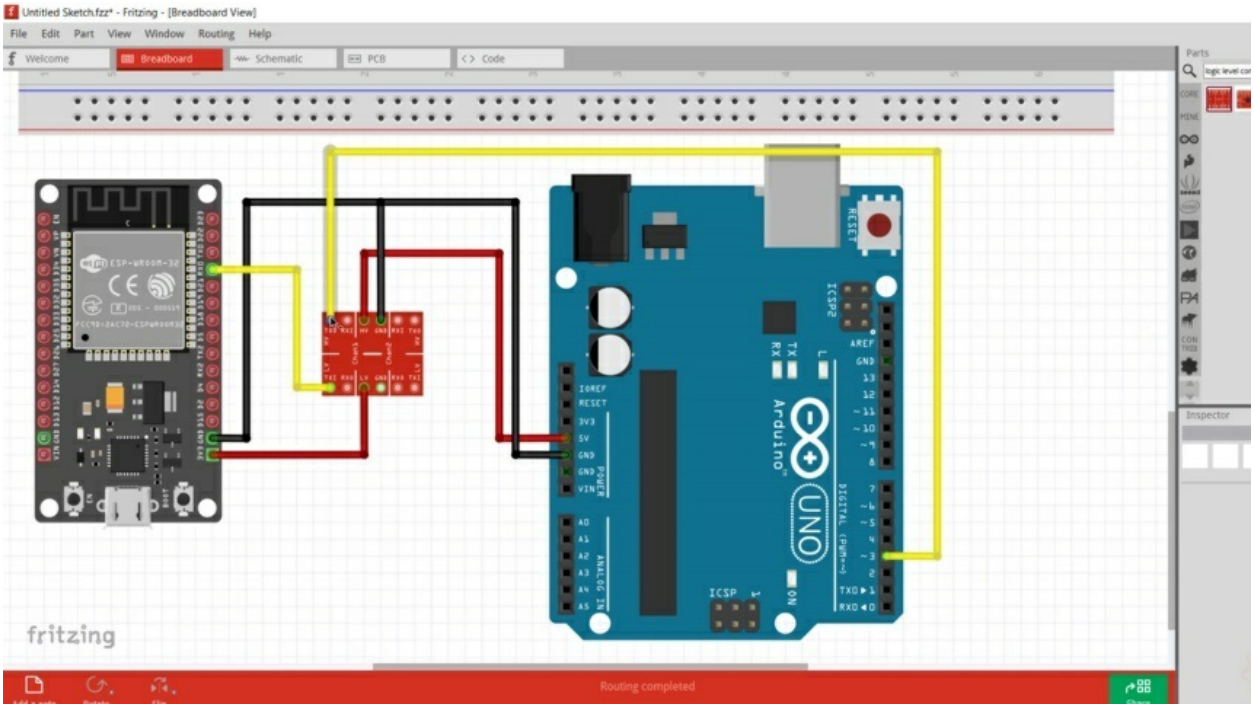


If you. Take a look at what clothing? You can see that here we have HelloWallet and we have icy Pleasant's E. S. P 32, OK. Now, if we went back, as you can see, we have about eleven messages and this is our E. S. P compost. I will visit the E. S. P board and you'll see that the level never messages will become twelve messages. Once I receive the ball, I will flick Racette Dow. As you can see now, it's 12 messages. And if I clicked this, again, it will be 13 messages. So each time we turn on our E. S. P board, it will send Hello World to our de server. And once we connected the Tor Arduino board, it will start sending census data to this topic. And again, all that you need to do now is follow the circuit design instruction and the next section to connect the Arduino board. And yes, people together to start sending census data from Arduino to E. S. P filter to and from there to that Kutty surfer.

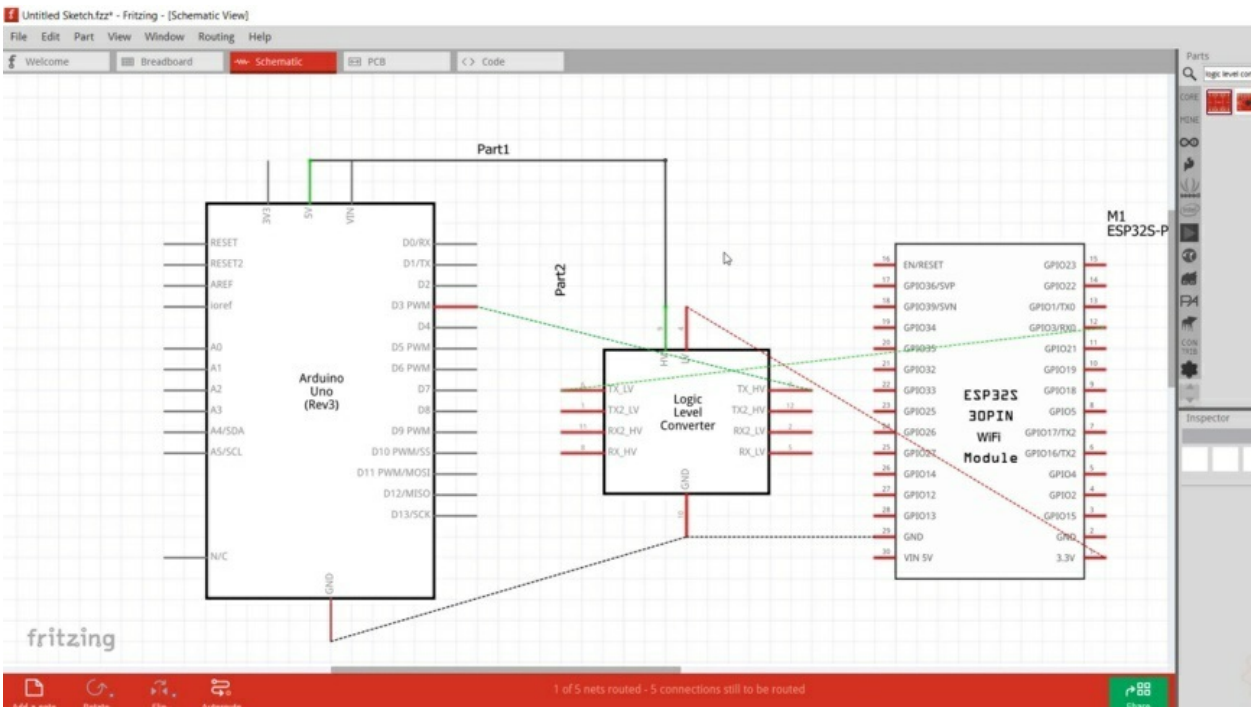
CIRCUIT CONNECTION EXPLAINED

No Child Willing to Connect E. S. P serves to board with Alvino. We are going to use flighting software. It's a very well-kn software for creating circuit design, and it's a very good software for shoaling the wiring layout for any electronic circuit. So let's go through a tablet bold mode here. Now we need to add Arduino boards so you can go here and select arguing. This is the board. Now, the next tip will be. Getting the yes Peters to board. This is it. Here we are. Yes, with his two balls. Now we need logic at Shifter. You can buy logic and they will get tons of frozen. Protostar. OK, now we have, as you can see, logic liberal, conversa, we have this one. And we have others as well. But this one will be enough for us. Now. To start connecting this, sorted this by 90 degrees, right now, we need to connect the three-points three vault from that E. S. P board, which list board to our logic level Schifter. So this is the three-point three vault. And we need to connect it to. The low voltage bin, this one, LDV. And let's make it red. Now, the next step is connecting the five volts from our old motherboard to the HP or high-level spin. So let's track this one here. That's hurt. Now. We need to connect the ground from our outside Reno with the logic of conversa with the yes people. So ground from here must be connected to the ground of the level of touch on the ground from here, must be connected with the ground from our level shift. Now, this show, the coloring to black here. I don't hear. List that I this pin. OK. Now. Moving on. Let's approach this. Now have connected the part on the ground and everything is gonna correctly. We need to connect with serial communication. Arms. And the first step is going to our door, you know, on aboard. And since we have initialized in the coding bins. Number two and three for the connection, we need to connect that. I've been number three, which is the sprint to our logical converter. And we need to make sure that we are choosing a T X ATV or high voltage. So we need to go to the T X.

HIV. And as you can see, we have more than 16. This is T x2 SUV. We need a T X SUV, which is disposed of. No. Let's move, it appears. Kerry. Now, let's show the color yellow. I have connected PINthree has no aluminum cord. We have set pins number two and 346 on our X. Now, the Artex from here, which is number three, is connected to the TX, it's V, make sure that you are choosing that item. Then you need to connect the R x d Xeroform. Yes. People. So as you can see here, we have our X zero two six zero. So now listen, go and grab the ah x pen and Caniff to that he x Elvie. Here. OK. Now, as you can see, this is the three X. And we have the Arduino pin connected to three X. This is the conversion it will take from Arduino converted and it to vote yes people without burning the board. And this is the main goal for the converter.



Now, you can see the schematic by clicking here. You can see.

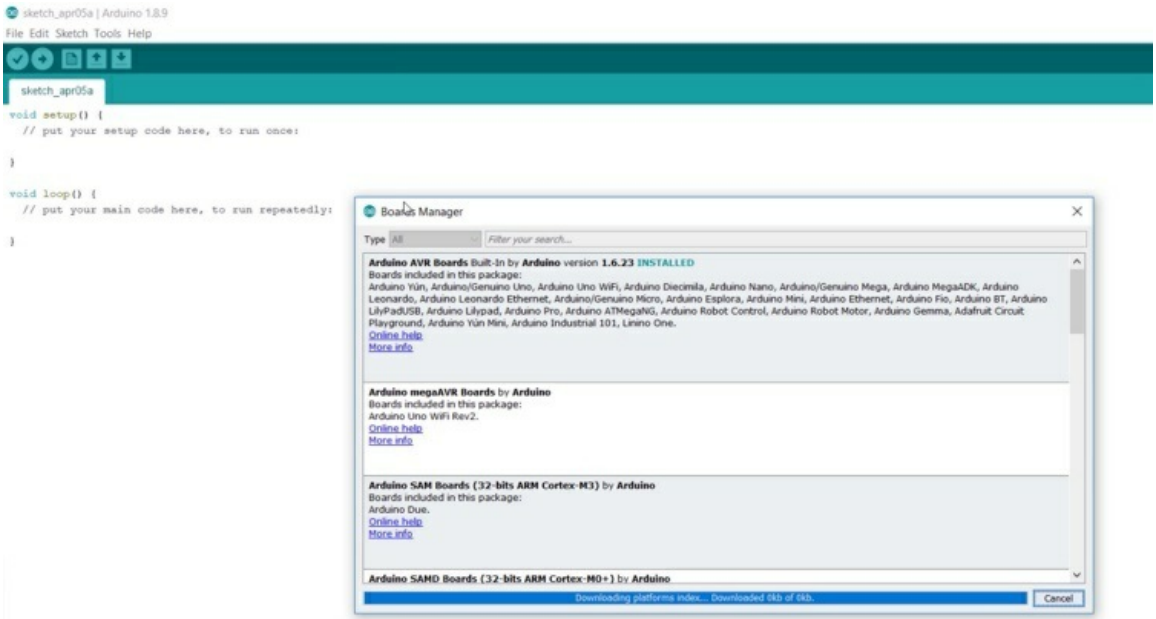


These are the two boards. You can move these balls like this and we can rotate, as you can see, 90 degrees to buy a hundred eighty degrees, okay. Now the ground is connected to these two balls. And as you can see, we have this pent-up X, and yes, people connected to the T, X, LV PINsix. And we have been DS3 from our albino connected to that. He is HIV and we have the HIV high voltage connected to the five volts and the low voltage, all the Elvie connected to the three blocks revoked. That's it. This is our connection. Make sure that you have everything connected, that we don't mix things up. Try to double-check your connection before moving forward and connecting your part, because if you have connected any way out in your way, it might damage your bonds.

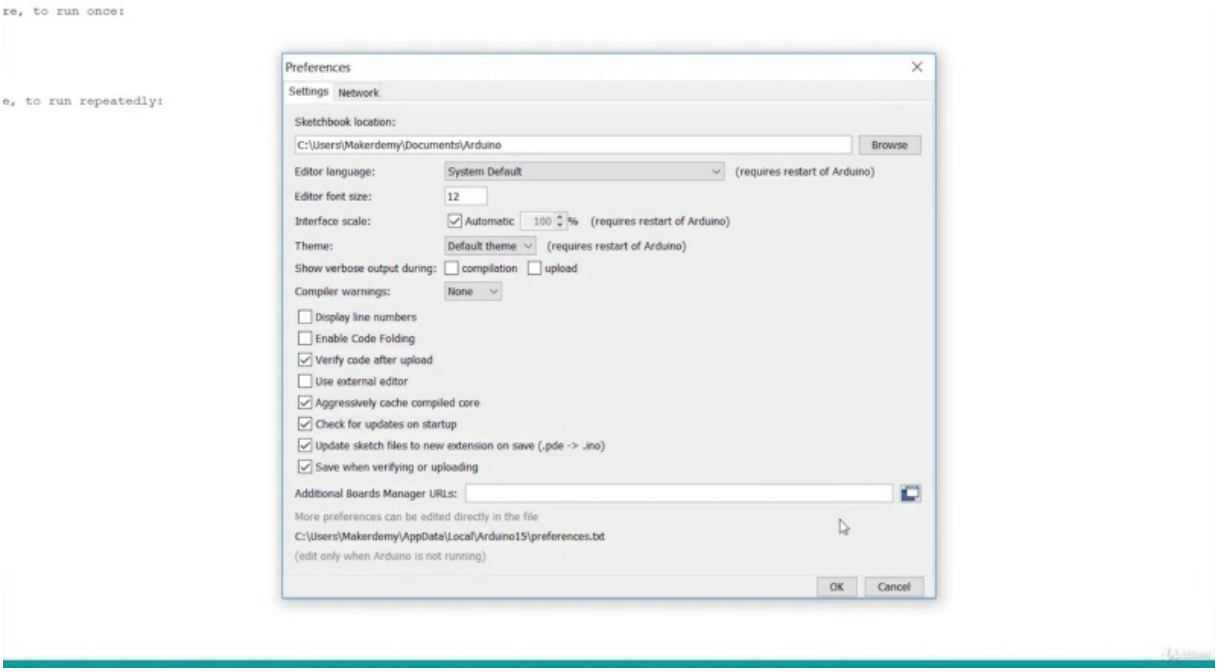
HOOKING UP THE ESP32 THING TO THE ARDUINO IDE

We will learn the following Installing the ESP32 core on the Arduino IDE Blinking an onboard LED on the Thing There are many ways of programming the Thing, but one of the easiest and most intuitive ways of programming the Thing is through the Arduino IDE. You can dload the latest version of the Arduino IDE for both Mac and Windows from the link provided in the resources section.

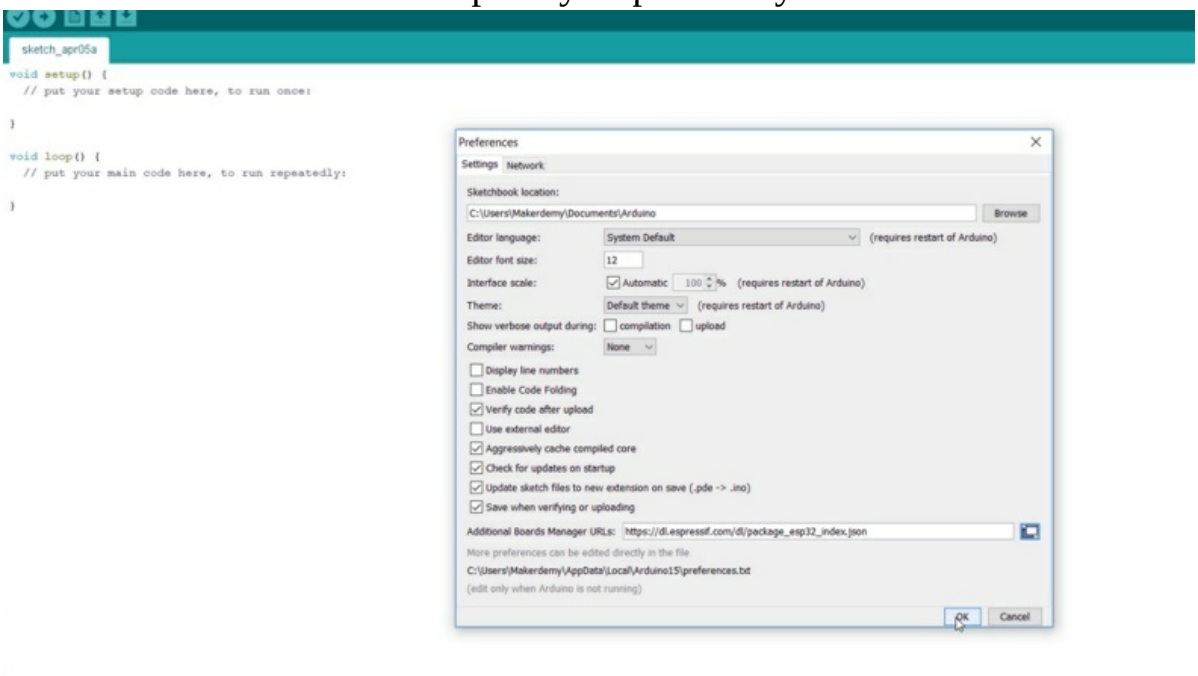
After installing the Arduino IDE, open it and you will have the development environment itself Firstly, you need to go to the tools section wherein you will find the boards section.



If you select the Arduino Boards Manager option, you will see a list of the boards which come preinstalled. From Arduino version 1. 6. 2 all Arduino AVR boards are preinstalled. But for other microcontrollers like the ESP32, you would need to install additional cores. The Arduino boards manager makes it convenient to install the Arduino core for the ESP32 Now if you search our particular board which is the ESP32, you will not find it. That's because we need to install a third-party core for the ESP32. To do this, go to file and then preferences In Mac you can find this option if you go to Arduino and then preferences.

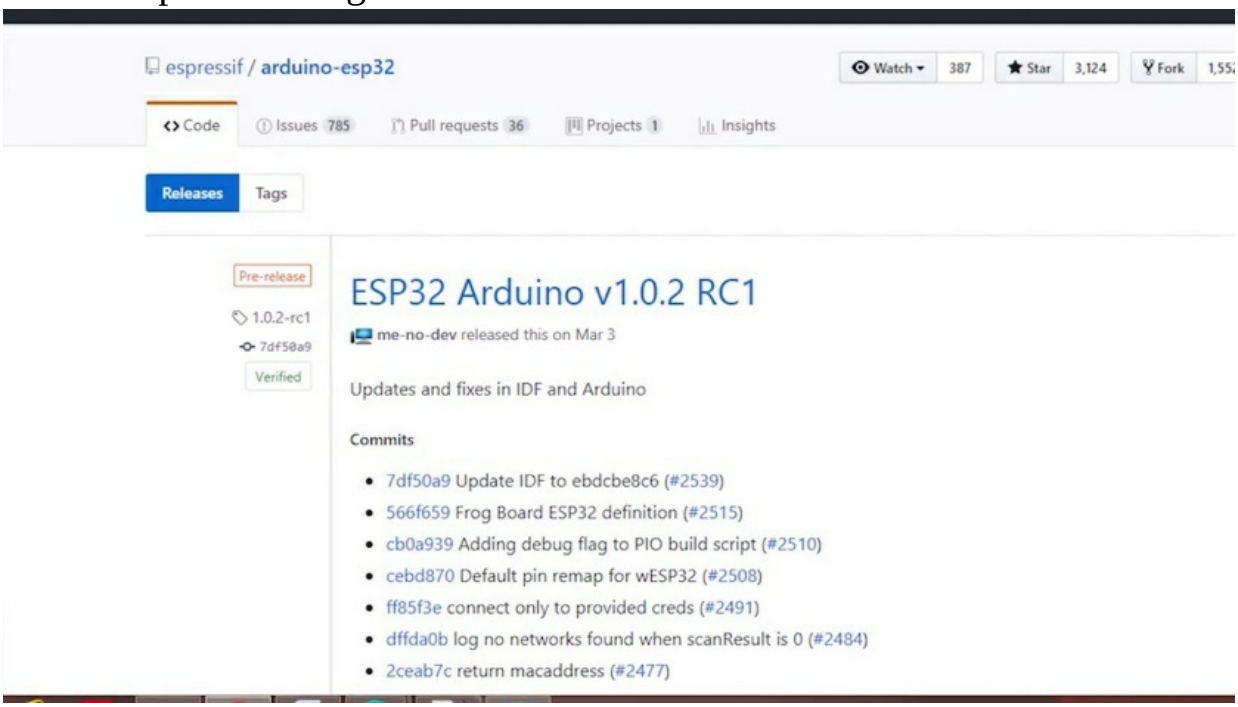


You will find a field called the Additional Board Manager URLs. The field needs a specific type of file This is a file written in JSON format which needs to be entered in the field for the additional boards manager to fetch the list of third party cores Don’t worry, you can find the link in the resources section Copy the link and paste it into the field. The additional board's manager will fetch the ESP32 cores developed by es press if systems.

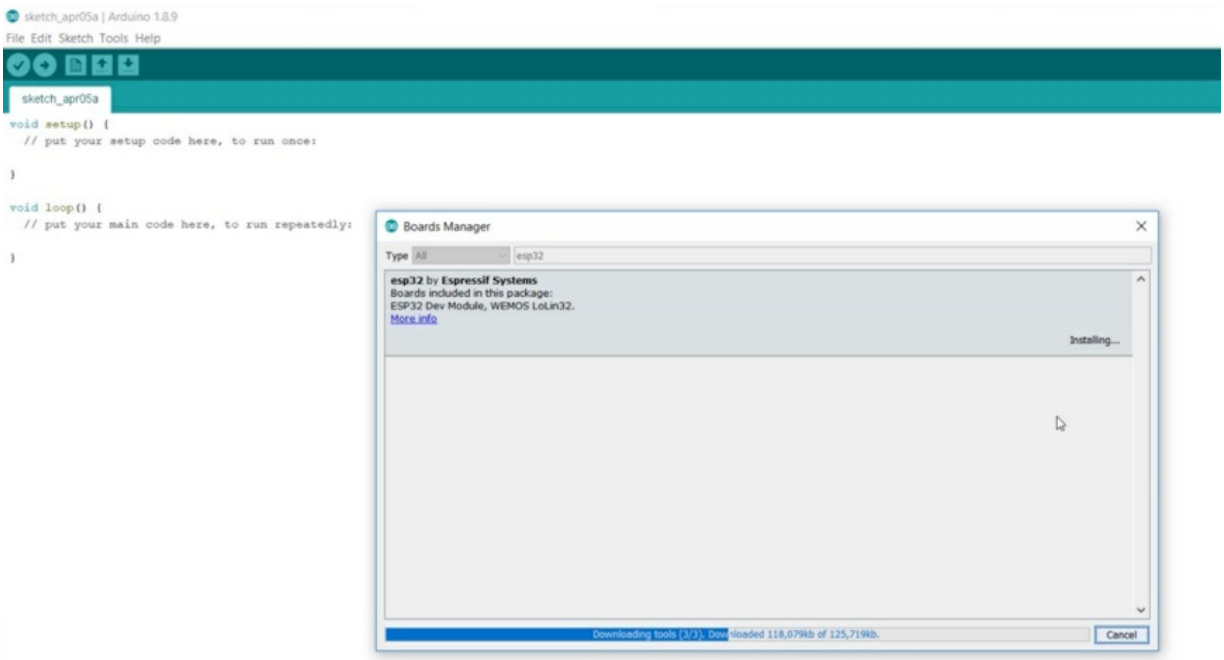


Now click here and paste the link. Now you can click on ok to exit the

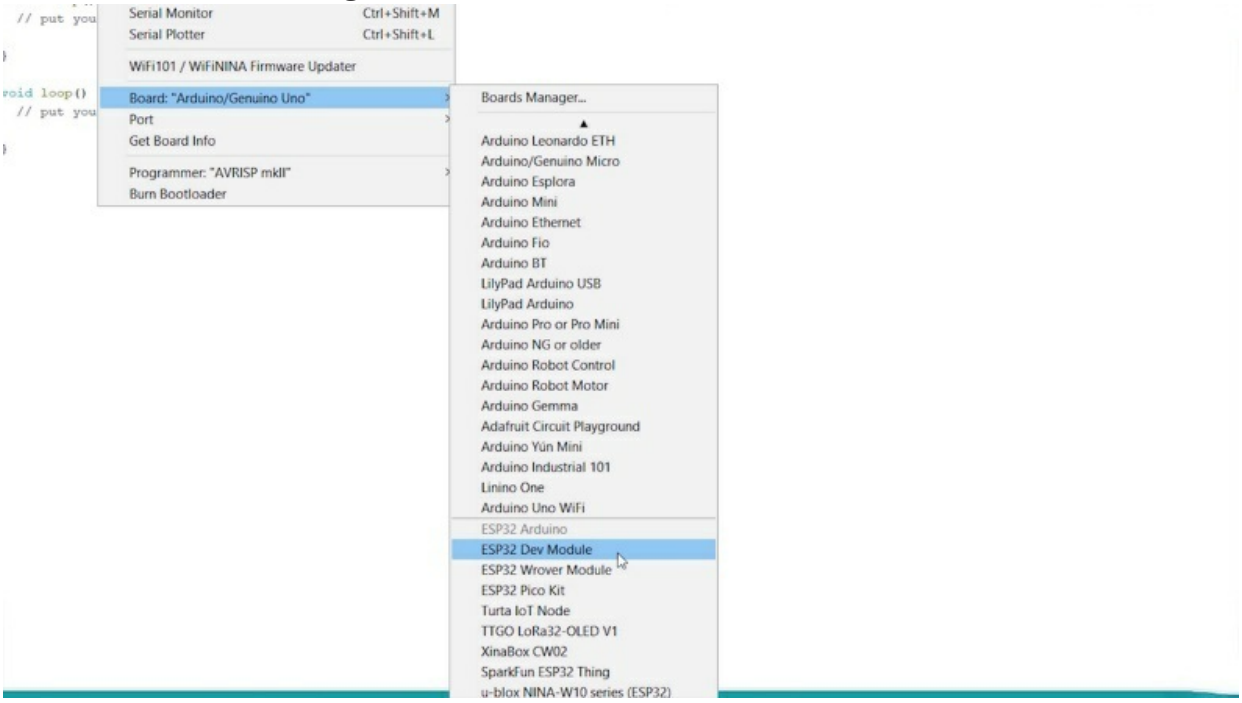
preferences window. Now if you go to the Boards Manager, and search for ESP32, you will find the list of boards. Here we can see that the package includes the ESP32 development module, which has the Arduino core for the ESP32. This will help us make the ESP32 compatible with the Arduino for using Arduino libraries and source code. You can dload any release you want, but you would want to dload the latest version as it contains updates and fixes previous bugs.



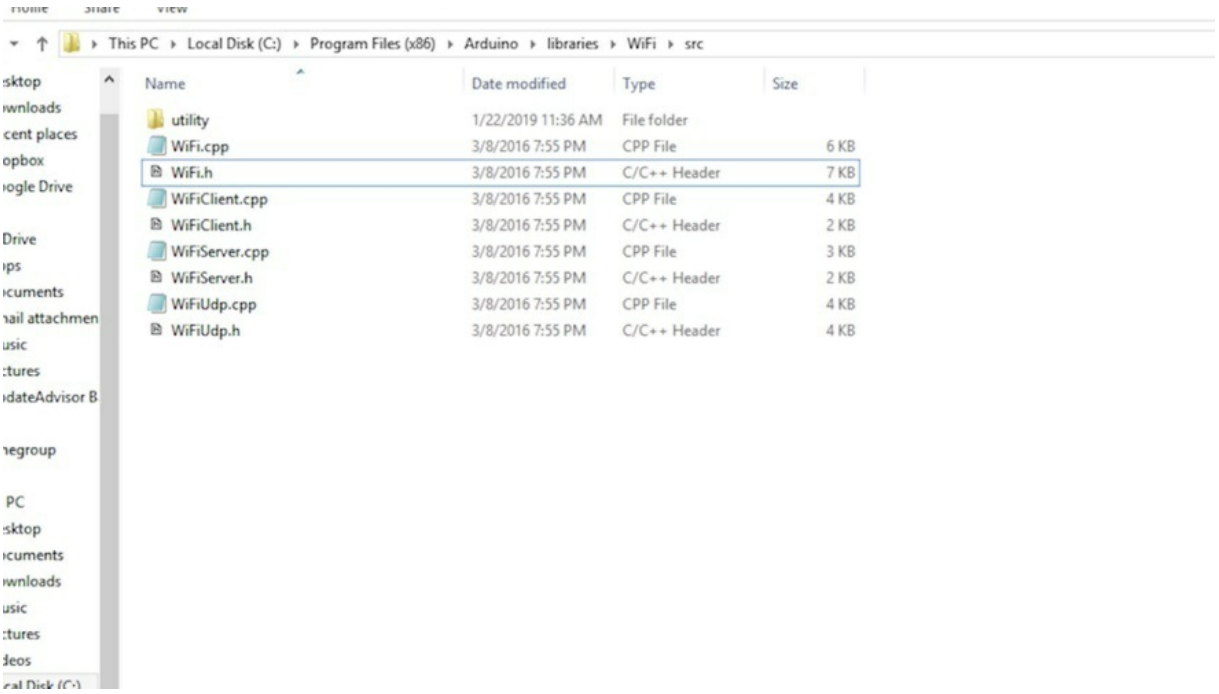
To check all the releases for the Arduino ESP32 core, you can find the link to the Github express if repository for the ESP32 in the resources section. We will be using Version 1. 0. 1 as of this date throughout the project.



Now install and wait for it to dload and install the packages Once installed, close the board managers window.

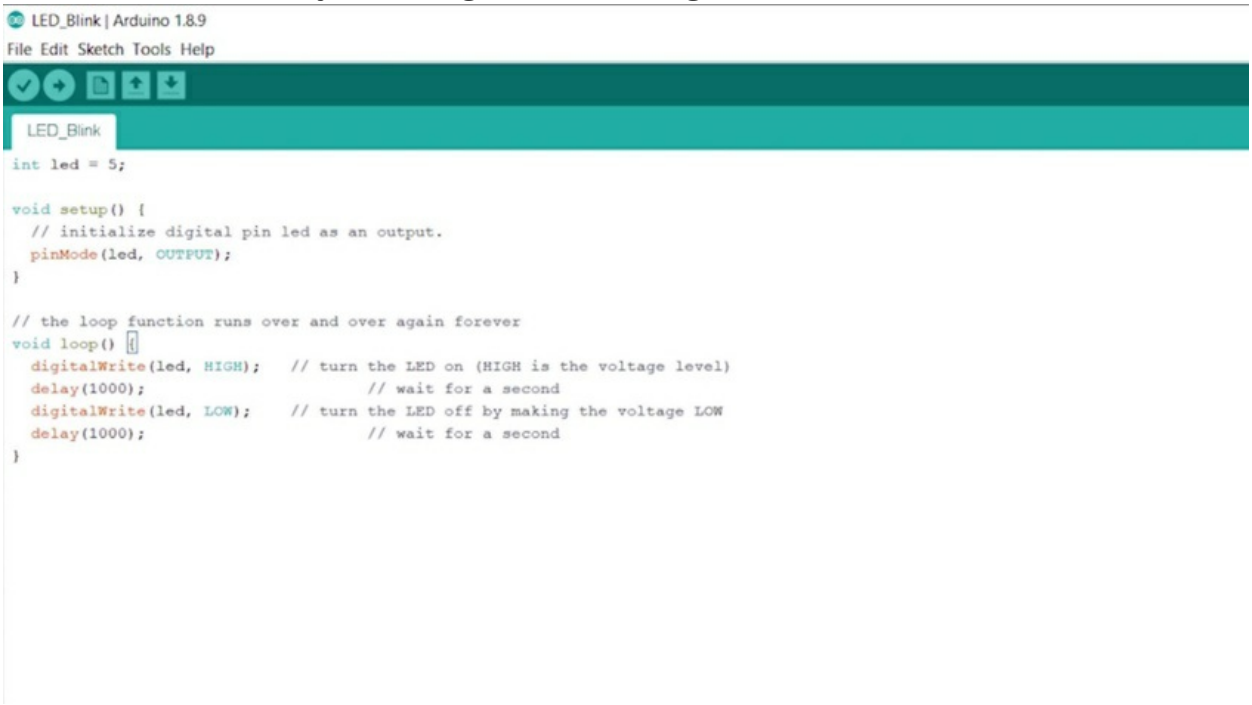


Now if you go to tools and check scroll d to check for the boards, you will find the ESP32 dev module. Now you are all set to program the Thing. This is the Arduino development environment. Arduino makes it easy to write code It does most of the work for you



The Arduino language is nothing but a set of C/C++ functions which gets called every time you run the program In Arduino, the program you run is called a Sketch This is what gets uploaded to the microcontroller board. First things first, let's look at these two functions over here Why are they here in the first place These are functions which get called whenever you compile and run an Arduino Sketch The first one is the Setup function The Void here is used to specify that the function itself does not return any value The setup function is used to initialize variables, libraries, declare pin modes, etc. This runs only once in the program. Now coming to the Loop function here this is where you write your actual code This will run multiple times. Let's look at an example to blink the onboard LED on the thing to get a better idea. First, connect your Thing via the USB to Micro USB cable to any of your computer's USB ports to start programming. You need to select the correct serial communication port also referred to as the COM port in Arduino. You can find this under tool and port. For Mac users, the file which maps to this port is the USB serial port. Since the Spark fun, ESP32 thing has an FTDI USB to TTL Serial converter built-in, you don’t need to use an external one to program it. This makes it super convenient to program it Once you have selected your serial port, you can upload a blank sketch to test if your code is compiling and uploading correctly. To do this just select upload. Here you

will find your debug messages after uploading your code. You will also get information on the amount of memory your whole code and variables are consuming. Hence optimizing your code is essential when your microcontroller has limited memory. If you have successfully uploaded your code to the ESP32, you will get this message



Now let’s go back to the code to blink the on-board LED on the Thing. The LED on the Thing can be accessed as PIN5 in Arduino Here we are first assigning a variable LED as integer number 5. Now this variable LED can be accessed as Pin 5 on the Arduino Here we are setting the particular PIN5 as the output. We need to do this only once, hence we include this in the setup function Here the digital Write sets the particular pin, in our case Pin no 5 as High or Low The delay of 1000 is in milliseconds which means toggle ling the LED on and off every 1 second. Now upload the sketch You will be asked to save the file first. Once uploaded, you will see the onboard LED on the thing blinking. The onboard LED on the Thing can be used as a test LED to debug certain errors in the code. The onboard button can be accessed as PIN 0 similarly. It is also a nice feature if you don’t want to use an external button for testing purposes.

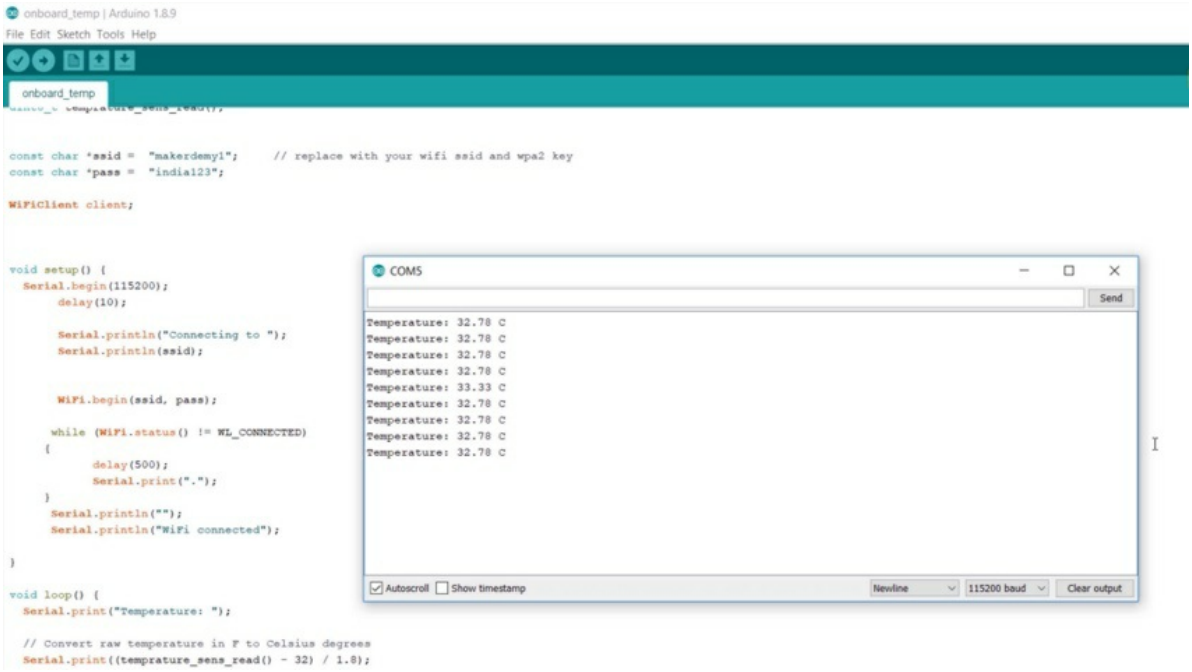
WORKING WITH THE ONBOARD SENSORS ON THE THING

We will learn the following How the temperature sensor on the Thing works and reading its Values How the Hall Effect sensor on the Thing works and reading its Values. How the Touch-sensitive GPIOs on the Thing works and reading its Values. Let's now look at the onboard sensors on the Spark fun ESP32 Thing. As you have seen in the previous projects, the ESP32 microcontroller chip on the Thing has a temperature sensor, a Hall Effect sensor, and Touch-sensitive GPIOs built right into it. The onboard temperature sensor on the Thing is different from your usual temperature sensors. Instead of measuring the ambient temperature, the sensor measures the internal temperature of the ESP32 microcontroller chip itself. The temperature sensor has a range of -40 degrees Celsius to 125 degrees Celsius. However, the absolute temperature measurements are not accurate This is because an offset temperature gets introduced with a different value for each ESP32 chip manufactured. So you may ask, then why even have a temperature sensor inside a chip that is not accurate and does not measure ambient temperature? Well, the temperature sensor on the Thing is not meant for that in the first place. The sensor is useful if you want to measure the

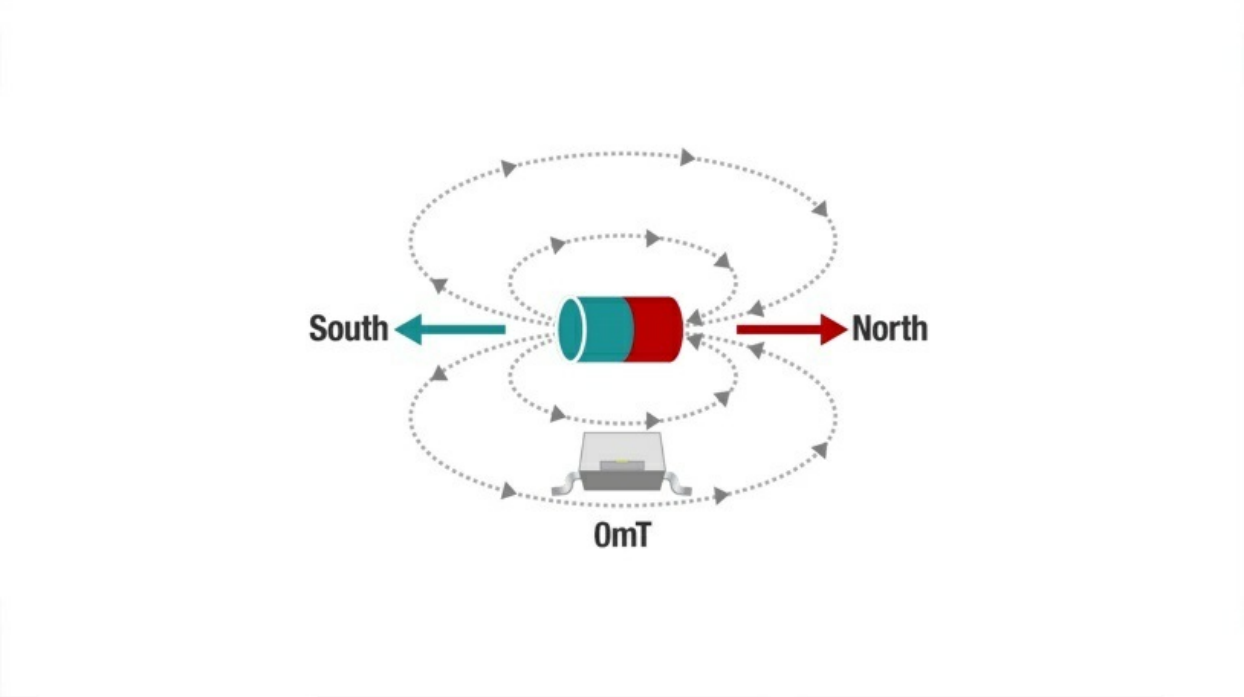
temperature difference in your chip and not the absolute value.
To illustrate this let's look at an Arduino code to measure the onboard temperature sensors value when it is connected to the Wi-Fi. We will learn about connecting to the Wi-Fi in detail in the next section. You can find the code in the Resources Section. This part of the code is used to inform the C++ compiler in Arduino to compile the C functions/libraries along with the

C++ libraries as one module Next we have the Serial. begin function, which is used to set the Baud rate for serial transmission between your computer and the Thing. In Arduino, the baud rate is the same as the bit rate Which means 1 baud = 1 bit Next, we have the Serial. print function, which prints the data to the serial port in ASCII format.

This function is used to get the raw temperature values in Fahrenheit. This is converted to Celsius by using the following formula. The delay here is introduced to run the code continuously at intervals of 1 second. Now you can compile and upload the code. Once uploaded you can go to tools and select Serial Monitor.



A Serial Monitor is a tool that allows you to send commands from your computer to the Thing. It also helps in reading the incoming Serial data from the Thing and to display it. Now select the correct baud rate which we have set in the Serial. begin function. As you can see, you are getting the temperature readings in Celsius So now how do you make sense out of it? The temperature sensor readings can be helpful to determine any abnormal changes in the internal temperature of the chip. This can happen in case there is an inherent malfunction in the chip itself which can cause the chip to become hot. In which case measuring the temperature difference would be useful Now let’s look at how to use the inbuilt Hall Effect sensor on the ESP32.



The Hall Effect sensor is used to determine the direction and strength of a magnetic field It can be used for proximity sensing applications.

hall_effect | Arduino 1.8.9

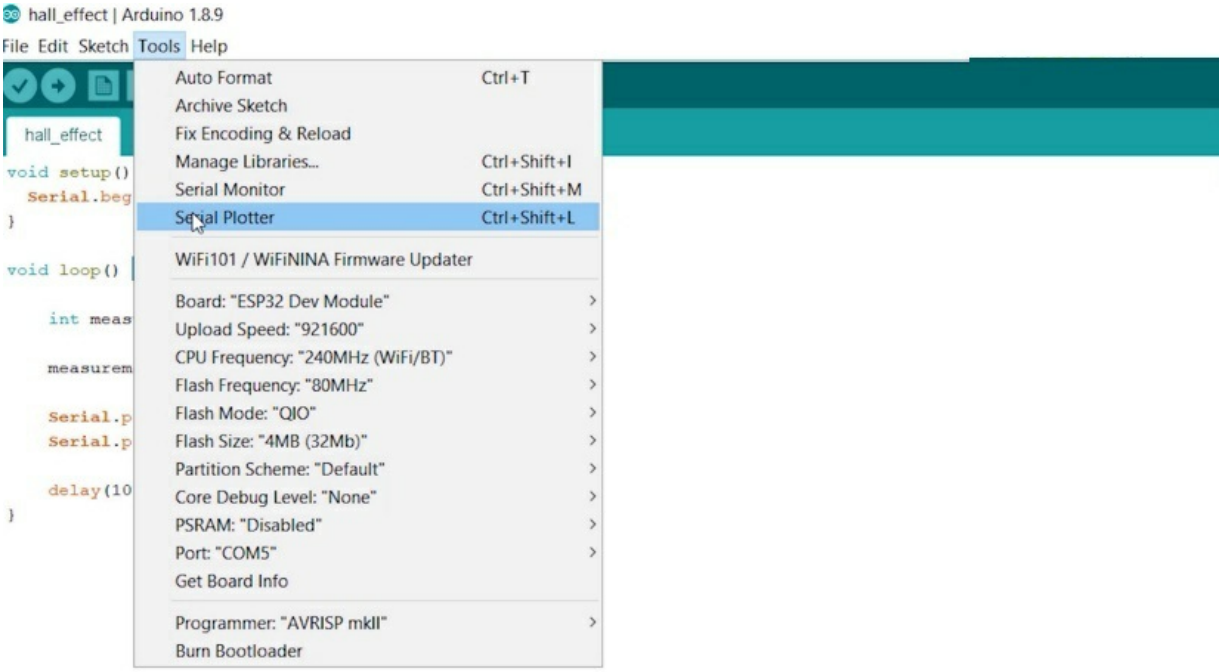
File Edit Sketch Tools Help

hall_effect

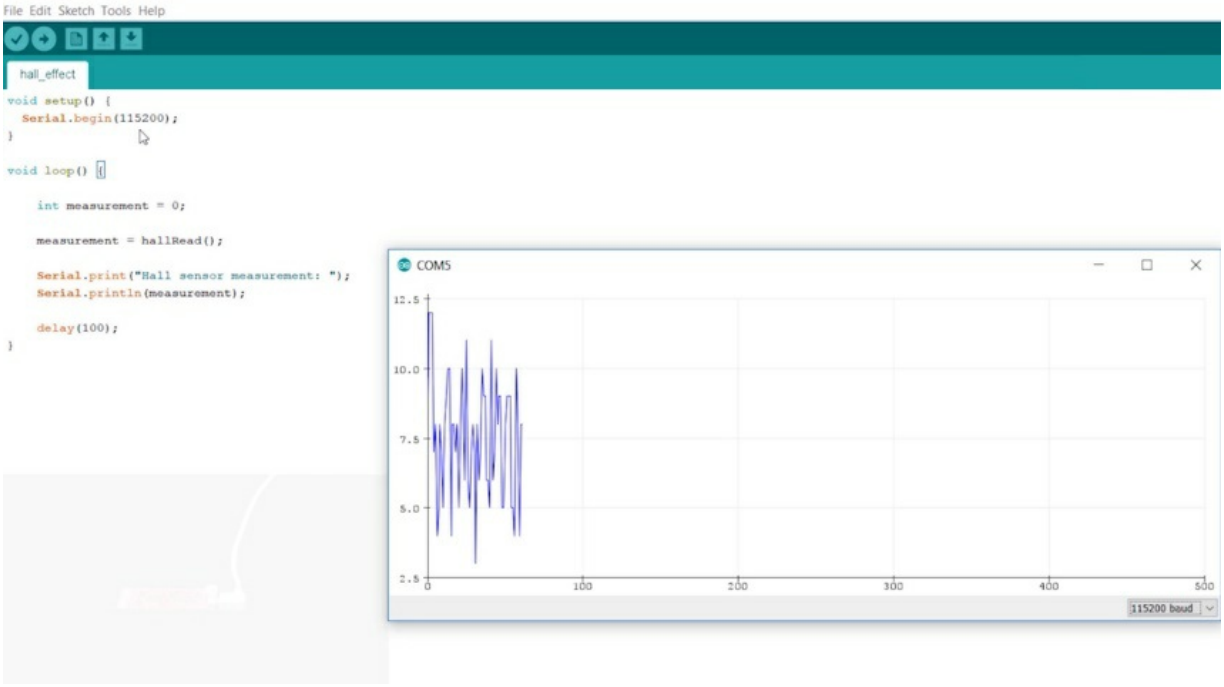
```
void setup() {
  Serial.begin(115200);
}

void loop() {

    int measurement = 0;

    measurement = hallRead();

    Serial.print("Hall sensor measurement: ");
    Serial.println(measurement);

    delay(100);
}
```

This is the code to measure the Hall Effect sensor readings Here we are setting the baud rate as 115200 The higher the baud rate, the faster is the transmission. The Hall Effect sensor produces a changing voltage with varying magnetic fields This voltage is measured by the internal Analog to

Digital converter. Hall read is an inbuilt function to read the Analog to Digital converter values This is stored in a local variable called measurement. This is then printed out to the Serial port. Now we will use another tool, not the serial monitor this time. This tool is the Serial Plotter.



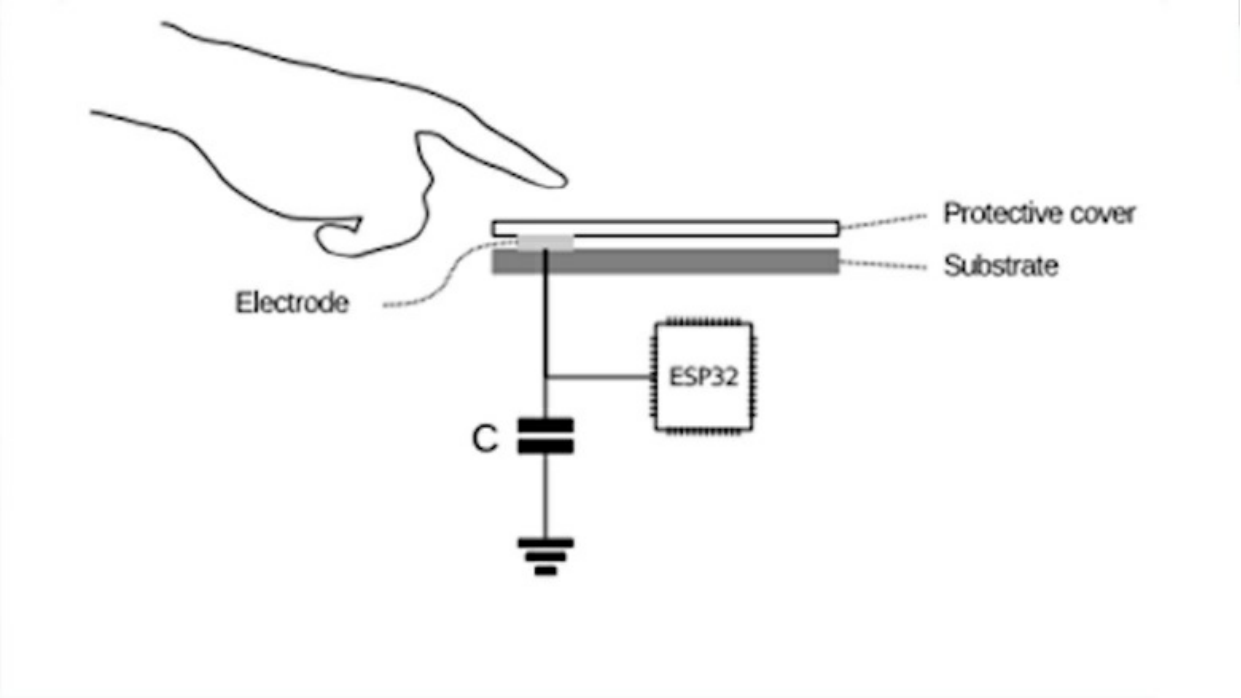
The Serial plotter is a great way to visualize your incoming serial data You can go to tools and then select the Serial plotter You can also open the Serial Monitor to view the readings in the text.



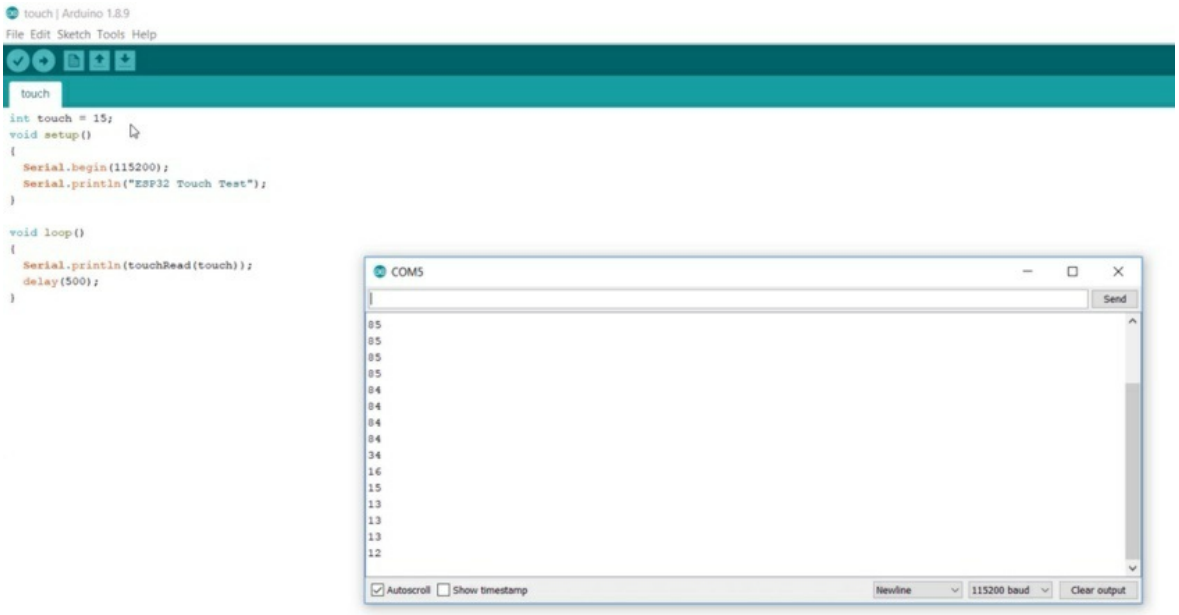
In the Serial Plotter, you can see that the readings are noisy. Let’s see what happens when we bring a magnet close to the ESP32 chip on the Thing.

The value suddenly shoots up. Now if we turn over the magnet, you will see that values drop d in the negative direction Hence the Hall Effect sensor can measure the direction of the Magnetic field. If you vary the magnets. distance from the chip, you will see that the readings will vary according to the

distance Hence the Hall Effect sensor in the Thing can also measure the strength of the magnetic field. Now let’s look at the touch-sensitive GPIOs on the Thing. We can choose from any of the ten touch-sensitive GPIO Pins. Here we are choosing GPIO PIN15. The touch Read function is used to read the analog to digital converted values and print it to the Serial monitor and Serial plotter.



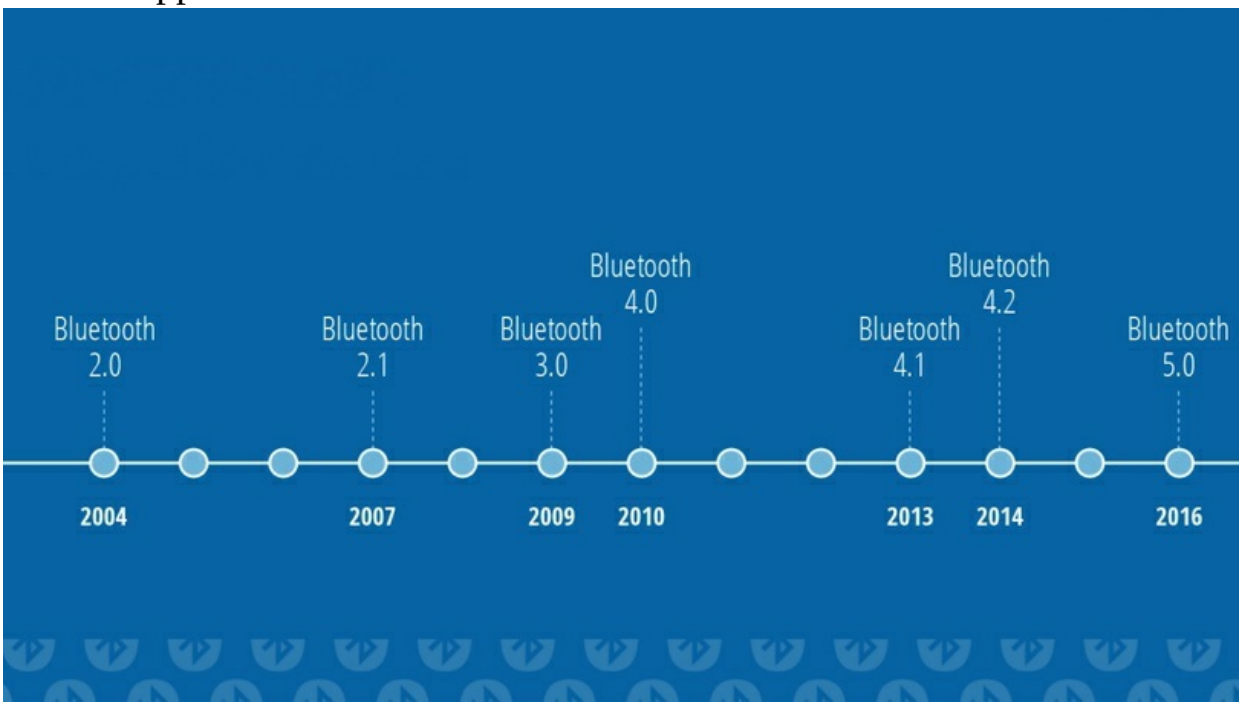
The Touch-sensitive GPIOs on the Thing work on the concept of capacitive sensing. It changes capacitance based on the distance of your finger from the GPIO pin This change in capacitance is converted to voltage change and measured by the ADC on the Thing. Now if we open the Serial monitor, we can see the change in analog values when we touch the pin.



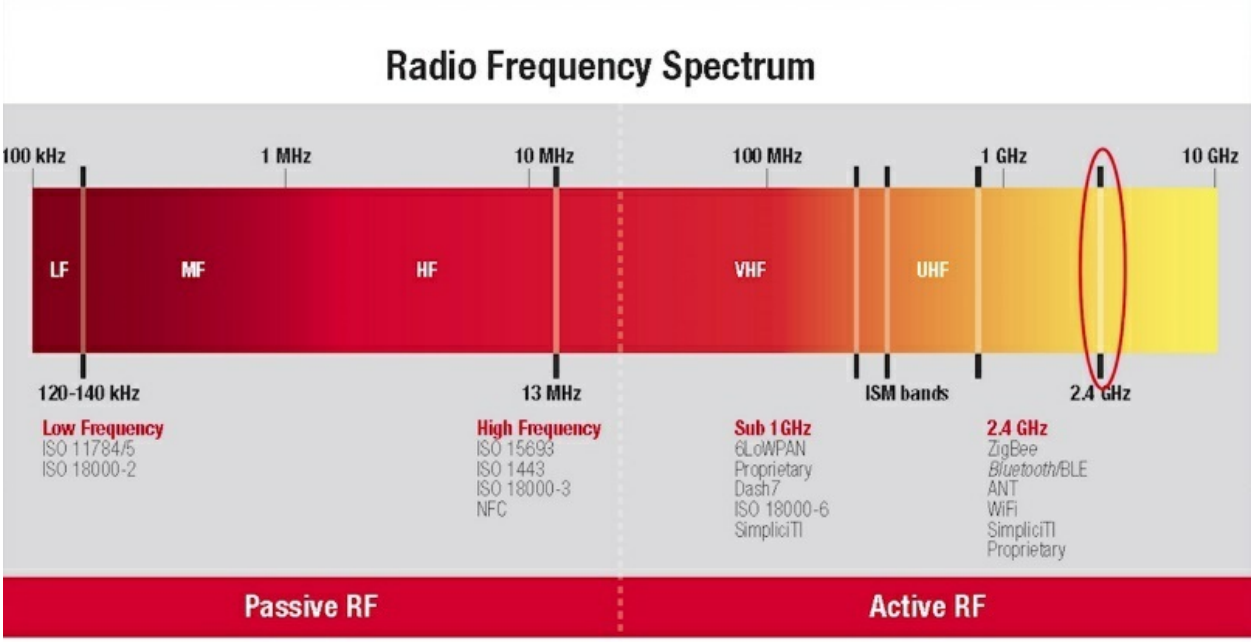
In this project, we learned how the temperature sensor, Hall Effect Sensor, and touch-sensitive GPIOs work and reading Values from the same In this section we cover the following introduction to the project getting to know you. Yes, P3 do exploring this park for an ESB terrible thing looking up this park fan yes Peter relating to the Arduino I. D.

UNDERSTANDING BLUETOOTH LOW ENERGY AND WIFI

What is Bluetooth and Bluetooth low energy Comparison between Bluetooth and Bluetooth Low Energy What is Wi-Fi and how it compares to Bluetooth for IoT Applications



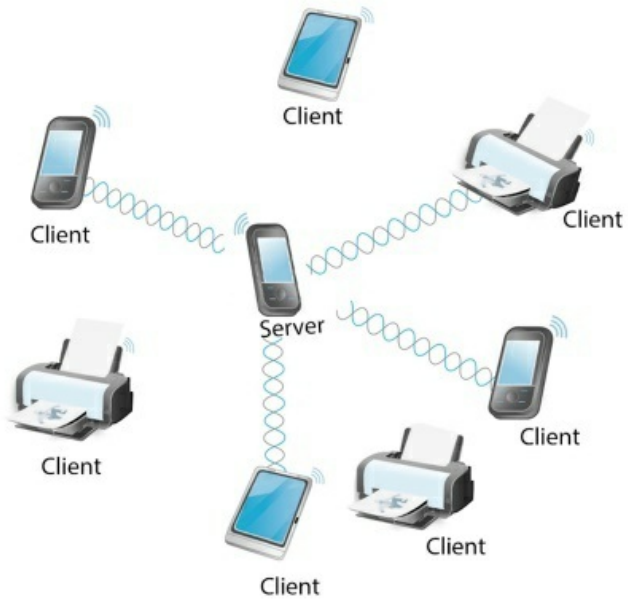
Bluetooth has been around for many years, and it has become prominent even more so now Reason? The world is moving towards wireless technology, and more and more devices are incorporating Bluetooth technology Wireless headphones to data transfer in smartphones Bluetooth has effectively reduced the need for wired communication



Let's look at what is Bluetooth Bluetooth is a communication standard that operates in the 2 4 Gigahertz radio frequency spectrum along with other protocols like ZigBee, Wi-Fi, BLE, etc This means that the data transfer between Bluetooth devices is in the frequency range of 2 40 to 2 48 Gigahertz

Bluetooth uses a master-slave architecture, where the master can

communicate with up to seven slaves in a piconet



You can also call it a server/ client architecture, which is normally the case when a smartphone application is the master device communicating with client devices via Bluetooth connection In the piconet, the slave devices can only communicate with the master device, but cannot communicate among themselves

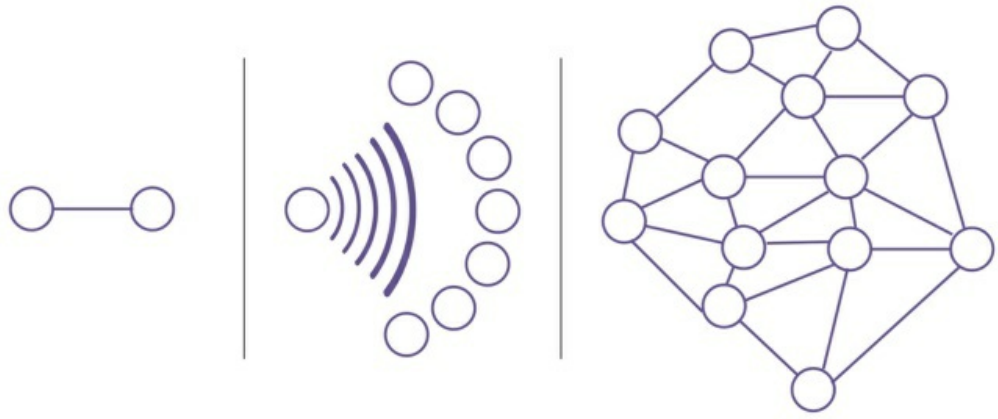
Bluetooth

7 Slaves

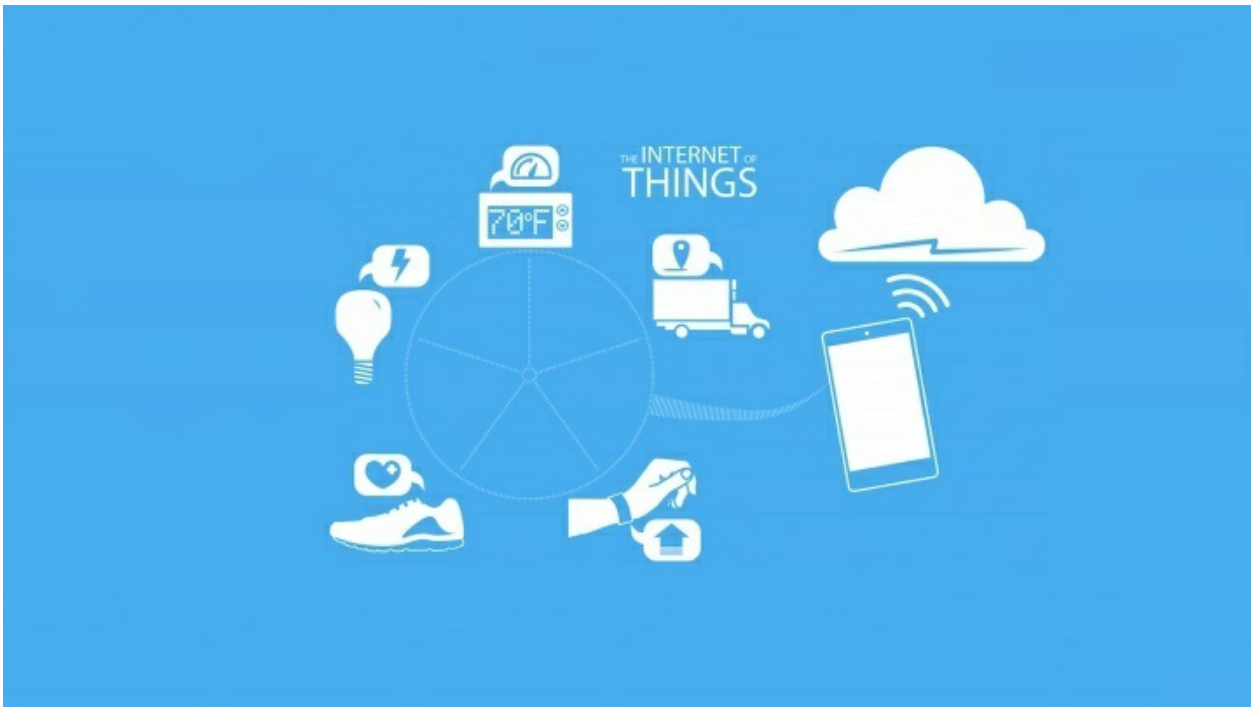
Bluetooth SMART

Unlimited Devices

The ESP32 chip on the Spark fun ESP32 Thing as discussed in the previous section has support for both Bluetooth and Bluetooth low energy So how are these two different and how do they compare? Bluetooth Basic Rate/Enhanced data rate also kn in simpler terms as Bluetooth Classic is meant for continuous data streaming Transferring large data like audio and project is supported by Bluetooth classic, On the other hand, Bluetooth low energy or Bluetooth smart is excellent for short burst data transmissions Meaning devices send small bits of data and hence save up bandwidth Bluetooth classic can connect with up to 7 slave devices whereas there is no such limit on the number of devices which can connect via Bluetooth low energy

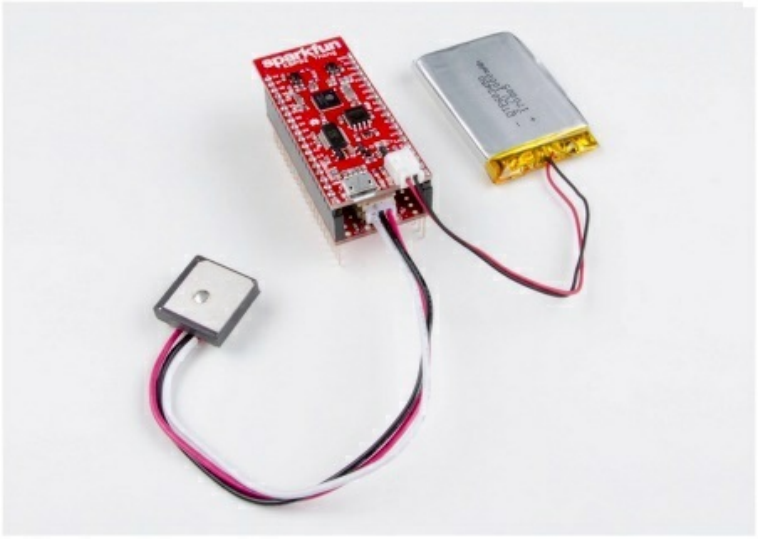


Bluetooth classic supports only point to point communication, On the other hand, Bluetooth low energy supports point to point, broadcast, and mesh topologies Bluetooth classic data rate ranges from 1 megabit per second to 3 megabits per second In comparison, the data rate for Bluetooth Smart is only in the range of 125 kilobits per second to 2 megabits per second Bluetooth low energy normally remains in sleep mode unless and until a connection is initiated

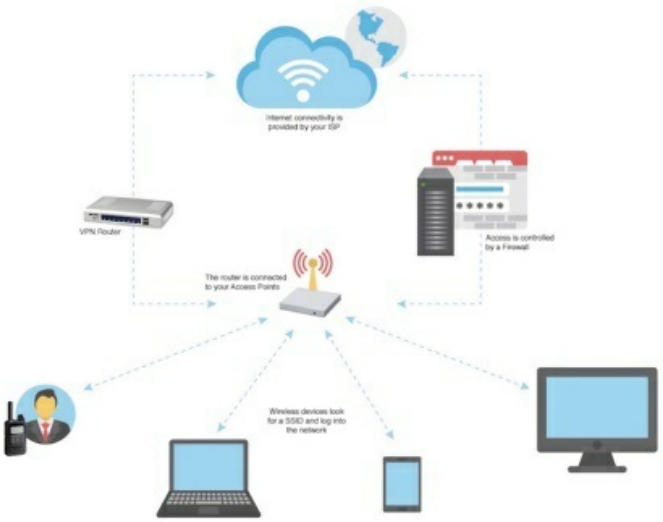


Now, you must have got an idea as to where Bluetooth low energy is used, in IoT applications of project

Bluetooth low energy is used in wearables, health-based sensors, home automation applications, etc All of the above features combined make Bluetooth low energy consume far less power than the Bluetooth classic



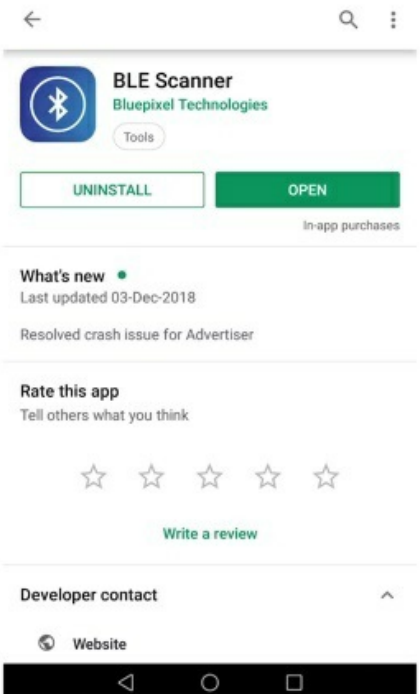
To put it in context, using Bluetooth low energy connection on the Thing can make it last days on battery power To understand Bluetooth even further in-depth, you can check out the Bluetooth Basics guide by Spark fun, the link to which is in the resources section Now let's talk about Wi-Fi Wi-Fi is also a wireless protocol which can operate in both 2 4 Gigahertz and 5 Gigahertz frequency spectrums The ESP32 has support for 2 4 Gigahertz Wi-Fi



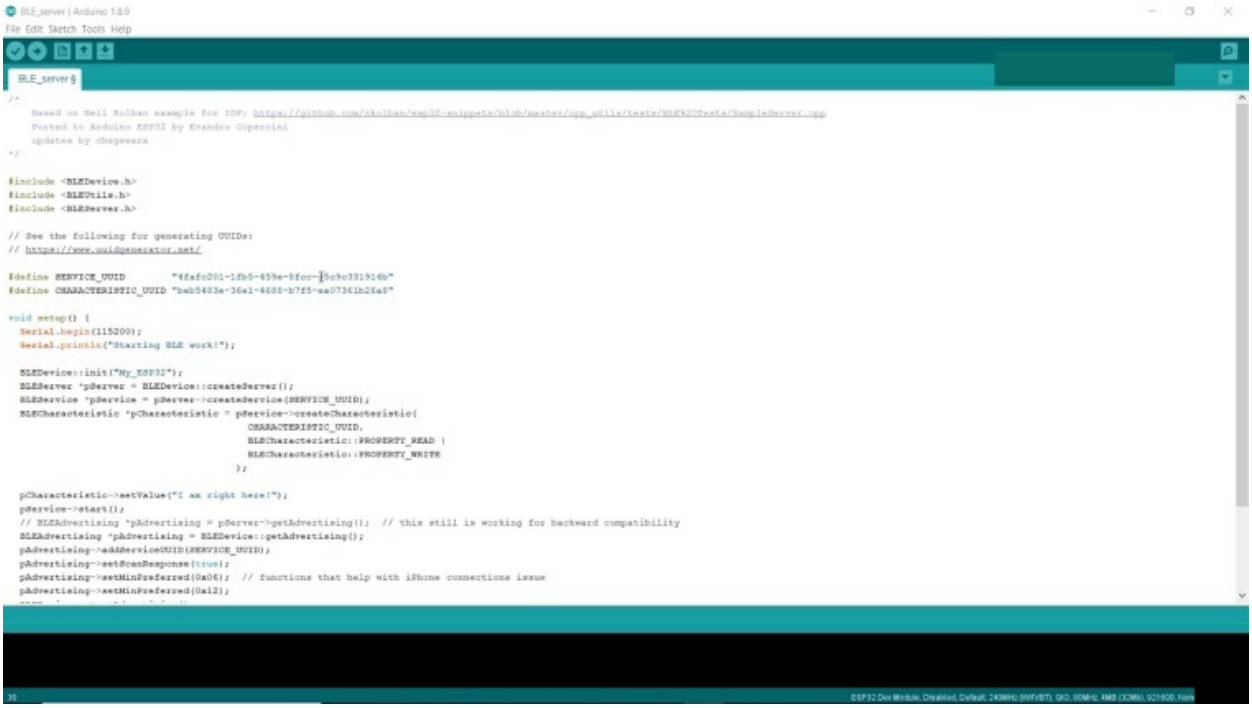
Wi-Fi is a type of WLAN or Wireless Local area network connection This means it is suitable for connections up to a small office or your home Bluetooth, on the other hand, is a type of WPAN or wireless personal area network Hence Wi-fi has a greater area of coverage compared to Bluetooth Devices connected to a Wi-Fi network can exchange data between themselves In Bluetooth however, the slave devices can only communicate with the Master device and not among themselves Bluetooth is capped at a much slower speed when compared to Wi-Fi Bluetooth is great if your projects demand short-range communication only, with limited bandwidth However if you need a larger range of communication along with a larger bandwidth you need to go for Wi-Fi For IoT applications, considering the power consumption and bandwidth requirement, Bluetooth, especially Bluetooth low energy is ideal If you want to send data to the cloud from your device.

ESTABLISHING BLE CONNECTION WITH THE THING

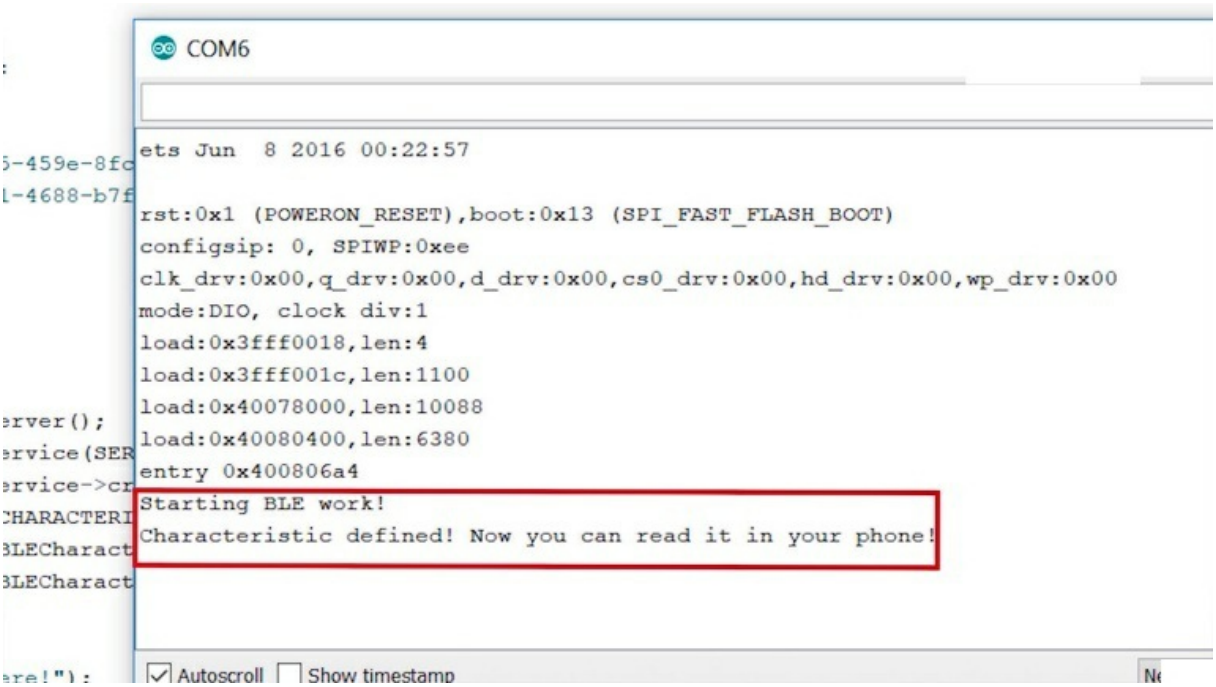
How to make data available on the Thing for the client device to read it How to write data from the client device to the Thing Let’s look at how to establish Bluetooth low energy connection between the Thing and your Smartphone Here your Thing will be the BLE server and the Smartphone will be a BLE client device



First, you need to dload the BLE scanner application which is available on both the Play Store for Android and App Store for IOS Now let’s open up the App This is the BLE scanner app Here you can find the list of all Bluetooth devices nearby

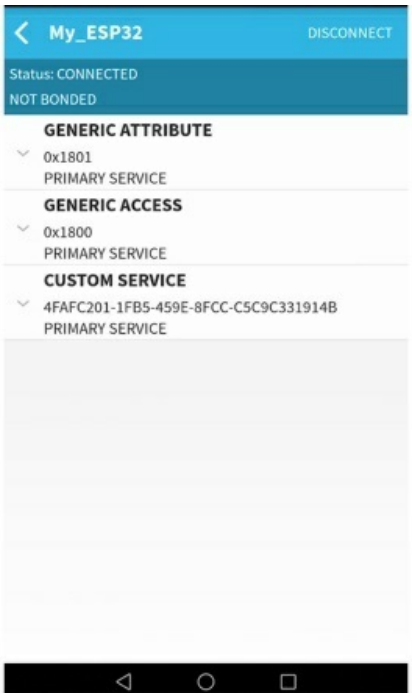


Now let’s look at the code to establish our ESP32 Thing as a Bluetooth server You can find the code in the resources section Before diving into the code Let us first upload it Once uploaded, you need to open the Serial Monitor If you do not see any value, you might need to press the Reset button on the Thing



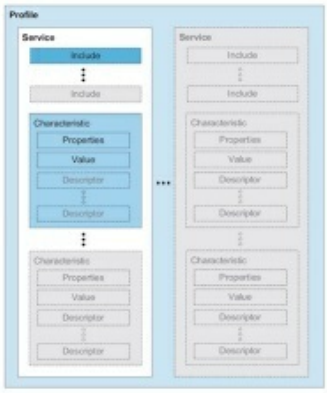
You will see this message pop up Now let’s go to the BLE Scanner App and

see if we can find our Thing Sure enough, you can Let’s connect to the Thing now



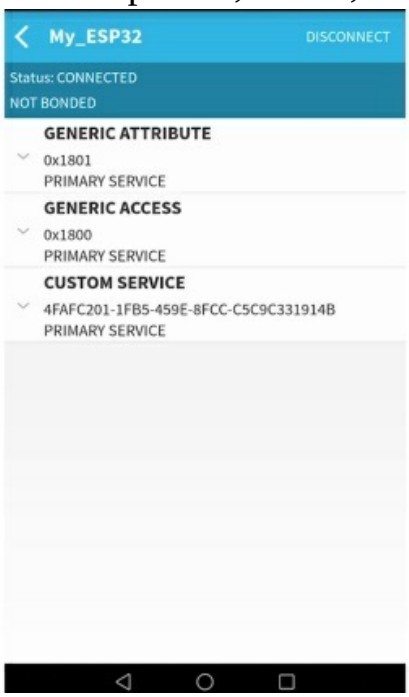
You will see the Status as connected This means you have successfully established a BLE connection to the Thing Now you will find the following the Generic attribute, Generic access, and Custom service

GATT or Generic Attributes profile

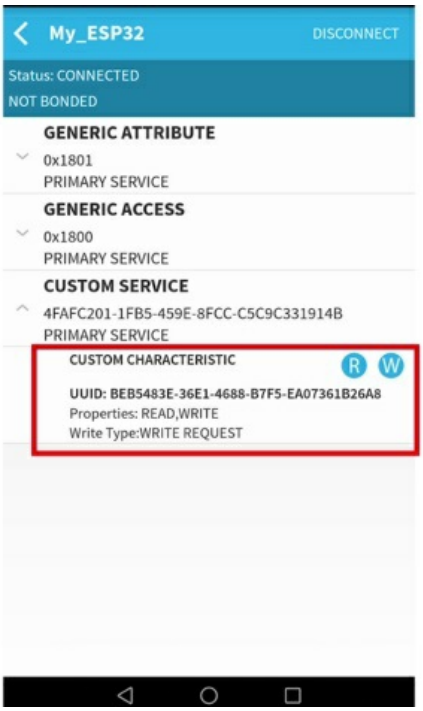


To understand this we first need to understand the GATT or Generic

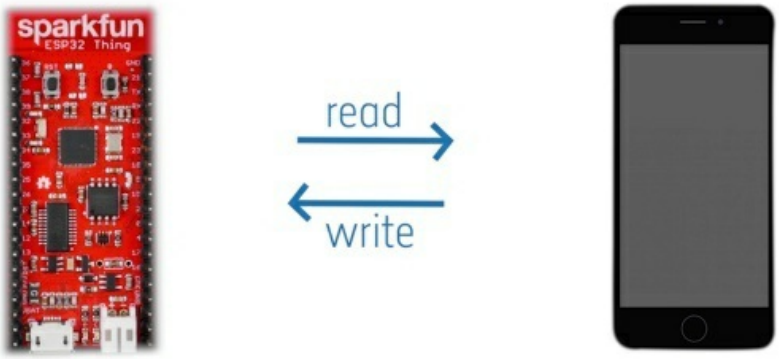
Attributes profile hierarchy You can say this is a format that defines how BLE devices communicate The topmost layer of GATT is called a profile, which is a collection of services Services can be either predefined as given by the Bluetooth Special interest groups or SIG, or you can create a custom service Services further contain Characteristics Each service has its set of characteristics Characteristics are what hold the actual data Characteristics further contain attributes like Properties, Value, descriptors, etc



Now coming back to the App you can see that there are two of the predefined services We also have a custom service Here you can see we have three services Two of them are predefined The first one is the Generic attribute service Inside the custom service,



you can see that we have a custom characteristic The Custom service has an attribute called properties Properties describe how you can interact with the characteristic value Here, we have read and write properties



This enables us to read data from the BLE device, in our case, the Thing, and write data to it Now let’s jump right back to the code to get a better idea of how it works

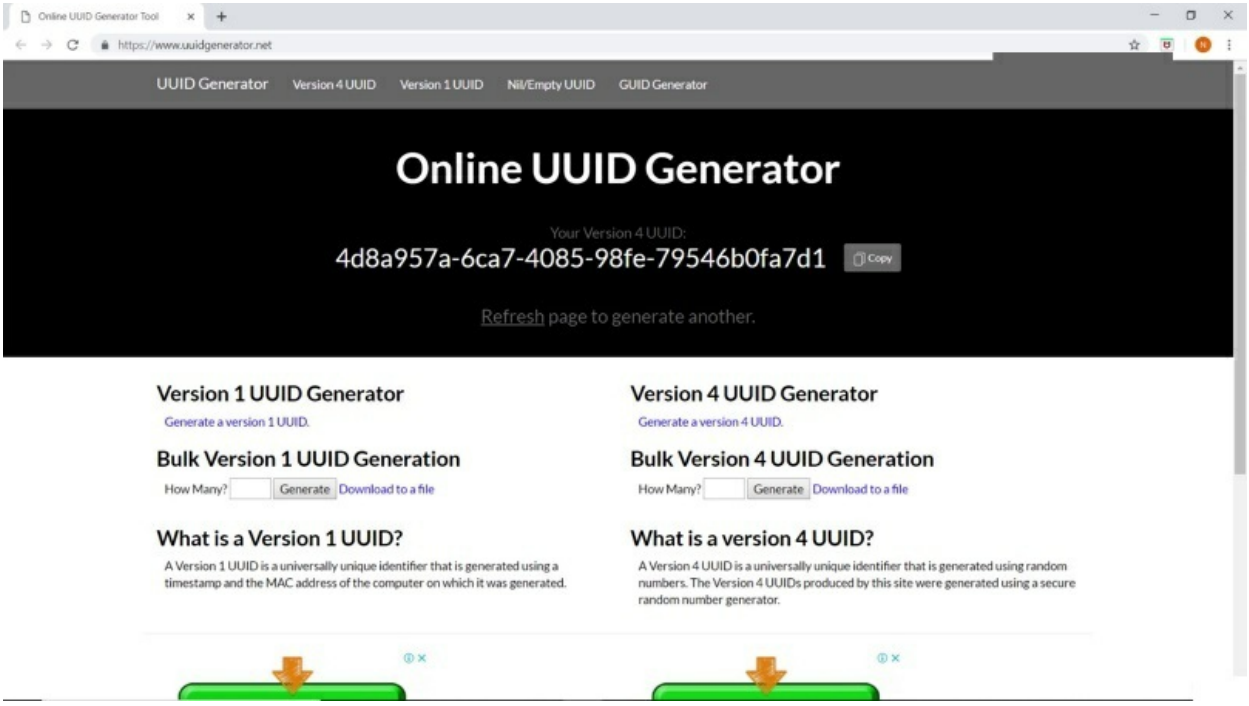
BLE_server | Arduino 1.8.9

File Edit Sketch Tools Help

BLE_server §

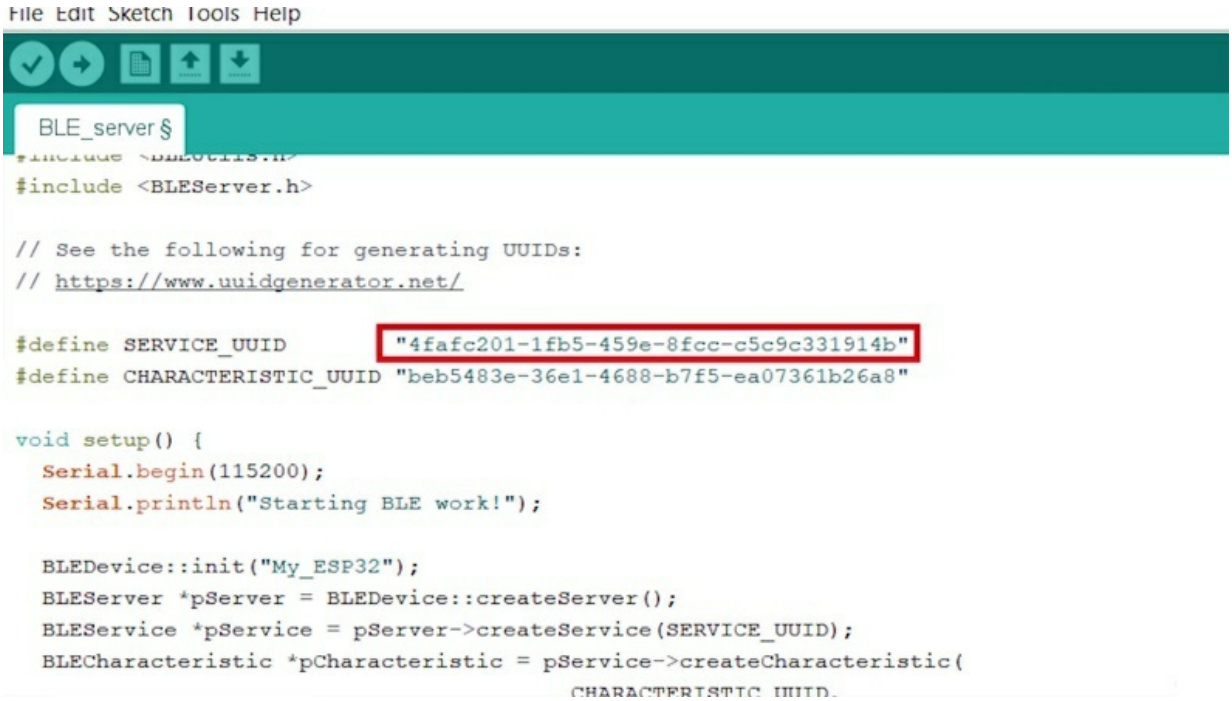
```
/*
    Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/master/cpp_utils
    Ported to Arduino ESP32 by Evandro Copercini
    updates by chegewara
*/

#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");
```

These are the libraries we need to import to use the BLE functions in Arduino IDE to communicate to the app Next, we define two UUIDs or Universally Unique Identifiers



A UUID is a unique 128 bit or 16-byte number which is used to differentiate between services Each service has its UUID Some UUIDs are already allotted to predefined Services, like the Generic attribute and Generic access

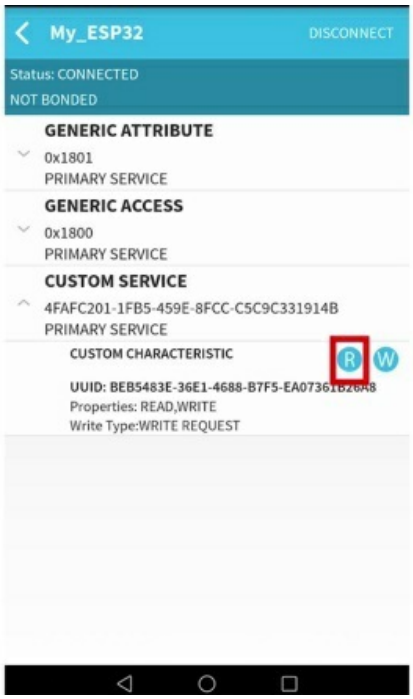
If we want to create our service and characteristic, we would need to have a custom UUID for both of them To generate your UUID you can go to the following website, the link to which is given in the resources section No need to worry about running out of UUIDs, there are two to the power 128 combinations!



Now copy the UUID generated and paste it here to assign it to the name SERVICE UUID You would need another UUID for the Characteristic Just refresh the website and you would get a brand new UUID Copy that and paste it for the Characteristic UUID

```
// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");

  BLEDevice::init("My_ESP32");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristi
                                       CHARACTERISTIC_UUID,
                                       BLECharacteristic::PROPER
```

Now let's look at our setup code here First, we set the serial transmission baud rate to 115200 Next, we print this to the Serial Monitor so that we can know that the BLE setup is about to start Here first initialize the Device name can change it to whatever you want This name will be displayed on the BLE scanner app So choose wisely This is used to create the BLE server This is used to create a new service, with the Service UUID which we had specified before This is used to create a new characteristic in the service which we have created with the characteristic UUID as specified before Next, here we have created two properties, which will enable us to interact with the App The first one is the read property which allows the BLE client,



in this case, being the Smartphone app to read the data available with the BLE server, being the ESP32 Thing The next property is the write property, which allows you to write data from the app to the Thing Next,

```
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID,
                                         BLECharacteristic::PROPERTY_READ |
                                         BLECharacteristic::PROPERTY_WRITE
                                       );

  pCharacteristic->setValue("I am right here!");
  pService->start();
  // BLEAdvertising *pAdvertising = pServer->getAdvertising();  // this still is
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);  // functions that help with iPhone connect
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
  Serial.println("Characteristic defined! Now you can read it in your phone!");
}
```

we have the set value function, which is used to set the data value for the read property This is used to make the data available to the app, which can then read it You can enter whatever data you want here, and it will be displayed as

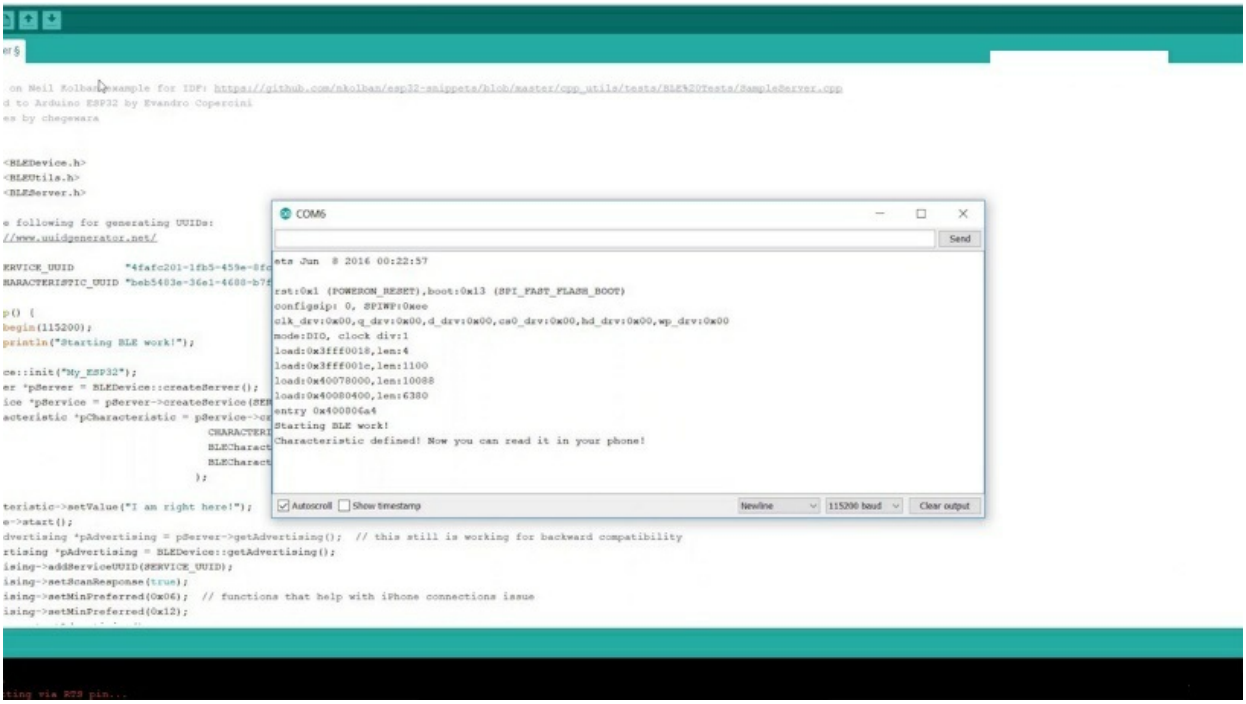
a value on the app When you read it

This is used to start the Customs Service Bluetooth low energy server works on the concept of advertising, which is exactly what it sounds like The BLE device advertises itself, something like saying hey I am here you can connect to me Once the connection is established, it listens for data from the client and the client can read the data available in the server This part of the code is just for that it is used to set the advertising configurations Next, we have the start advertising function which starts the BLE server advertising Once this is done, we can print to the monitor to confirm the completion of the creation of the customer service and characteristic

Now the thing is an advertising and can be discovered by your BLE scanner app You can leave the Loop code blank

Now if you go back to the app, you will see the custom service UUID which we had set in the code You will also see the custom characteristic UUID which we had set in the code In the properties field you will see two properties Read and Write which we had enabled in the code Now if you

select Read right here, you will see that the message we had specified in the code is now visible as Value, in both ASCII for us to read and Hexadecimal formats

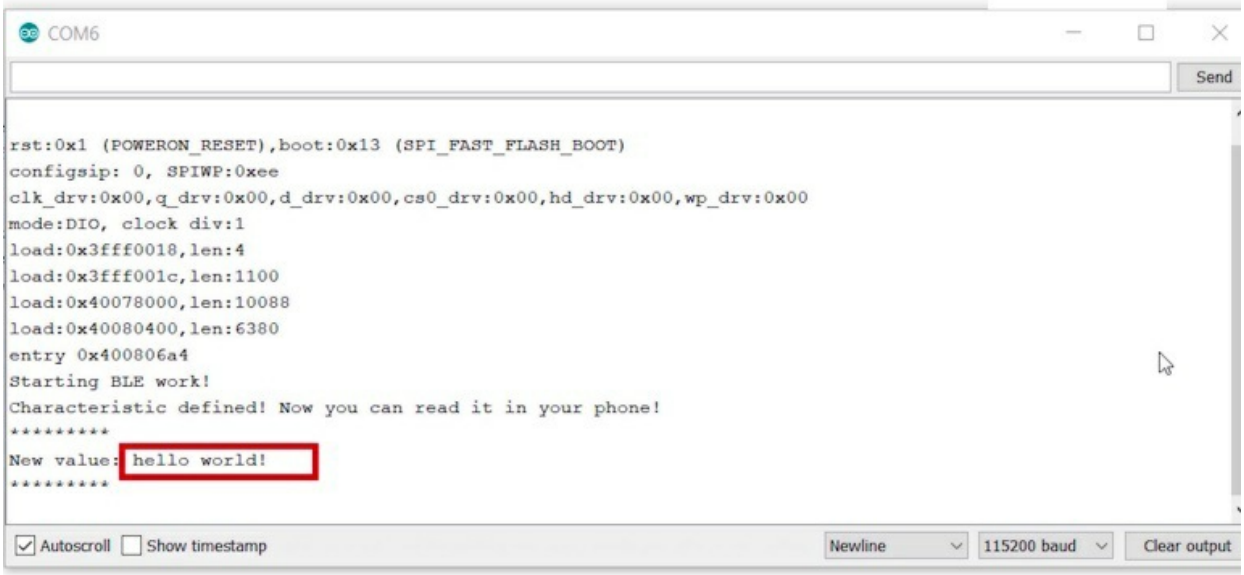


However, if you now select write and try to write data, you will not see any data on the serial monitor This is because we have not included the write callback which gets called when you try to write data from the app Let's now see how to write data to the Serial Monitor via the app You can find the code in the resources section

```
#include <BLEServer.h>

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string value = pCharacteristic->getValue();

      if (value.length() > 0) {
        Serial.println("*********");
        Serial.print("New value: ");
        for (int i = 0; i < value.length(); i++)
          Serial.print(value[i]);

        Serial.println();
        Serial.println("*********");
      }
    }
```

First, we need to include the write callback function which will get executed whenever the Thing receives data from the app Here we have the MyCallback class, which contains the write callback function The write callback function gets the data as a string value If the length of the string value is greater than zero that means we have received some data If the length of the string value is zero, that means we have not received any data and the if condition will not get executed Here we print out the value of each character in the string from zero to the length of the string Now the rest of the code in the setup function is the same as the previous version
for reading data But what has changed is this The setcallback here is used to set the write callback function for the write property So when you click on write,

the onwrite function which we had included earlier in the myCallback gets executed Now lets upload the code and open up the Serial monitor Now go to your app and in the custom service click on the write button Now enter the data you want to write to the Serial monitor Now if you click on OK,



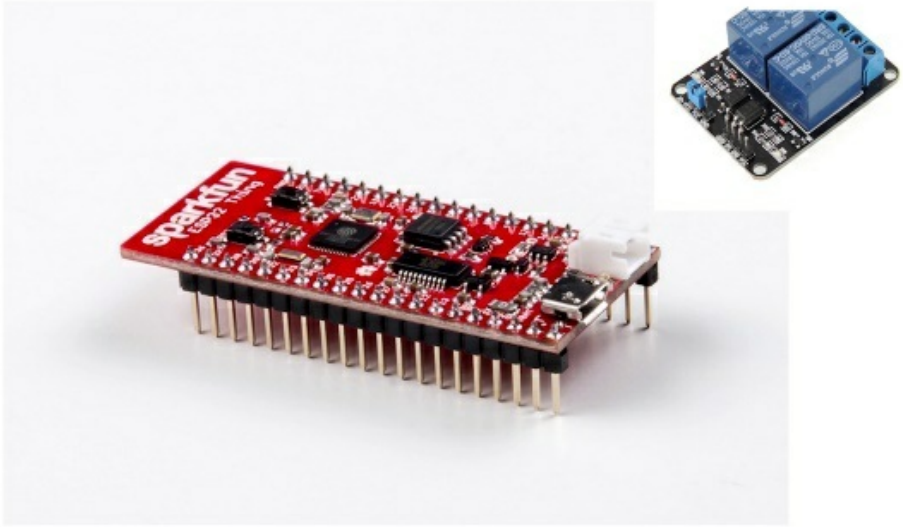
you will see the data printed out on your Serial monitor we learned how to establish the Thing as a BLE server We also learned how to make data available on the Thing for the client device to read it.

INTERFACING A RELAY WITH THE THING

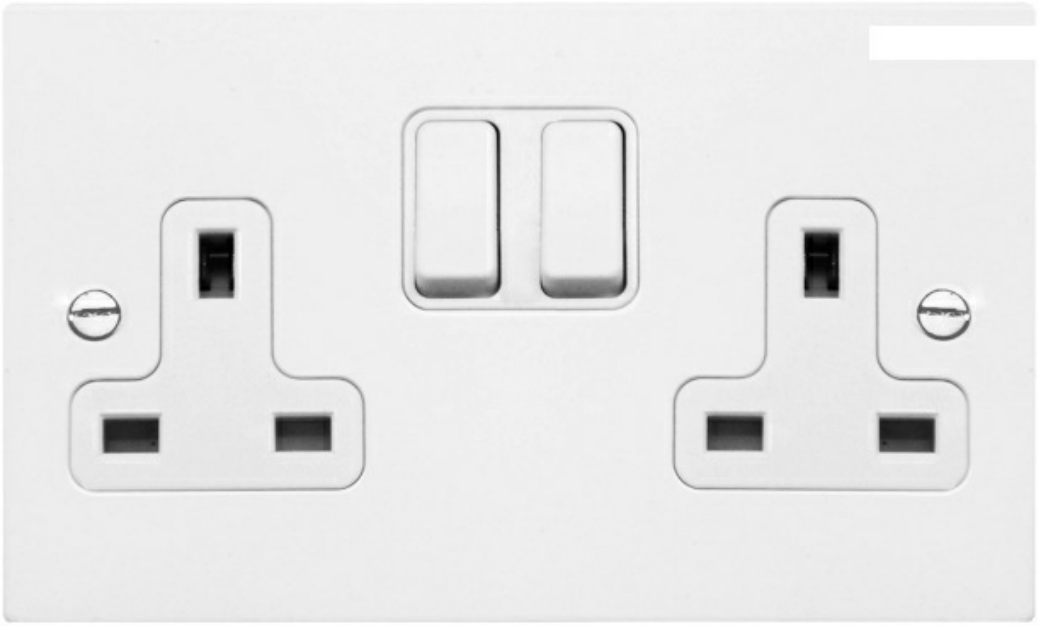
We will learn the following What is a Relay and how it works Interfacing the Relay module with the Thing to control 2 bulbs

In the previous project, we looked at interfacing PIR sensors with the Thing Now the first Thing will be responsible for detecting people in the hallway The second thing will be responsible for switching on the lights in the hallway

Now let's take a look at the second part of the project

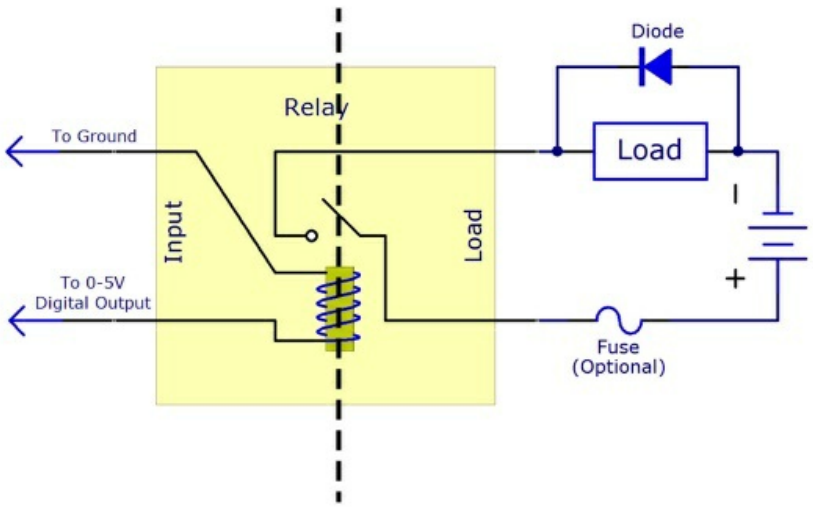


Here the second thing will be using a relay to turn the lights ON and OFF



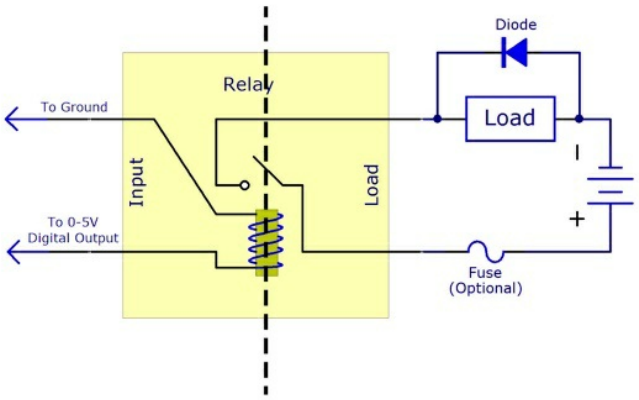
We will be using an AC mains supply to power the bulbs Now, the ESP32 Thing outputs a maximum GPIO current of 12 mA To drive a high current circuit for controlling the bulbs you would need an interface in between

This is where the relay comes in A relay is nothing but an electromechanical switch

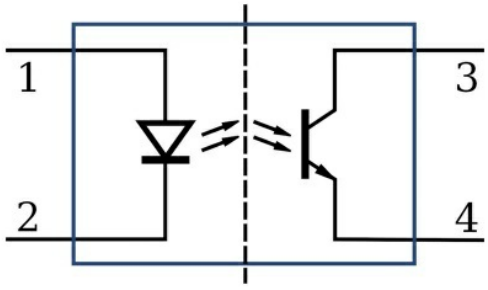


It is used to control a high voltage circuit using a low voltage circuit

The relay itself is rated at 10 Amps at 250V and 125V AC and 10Amps at 30V and 28V DC Now dealing with such high power circuits can be dangerous



We need to isolate the high power circuit from the low power circuit - in our case being the ESP32 Thing

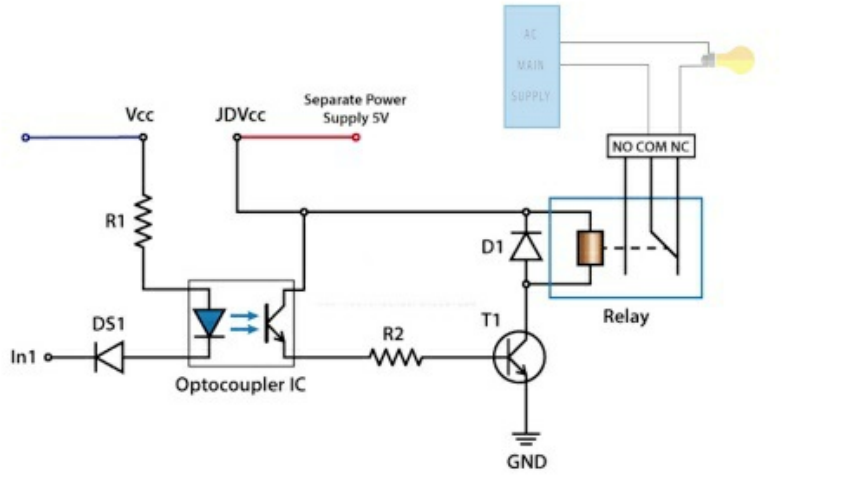


Hence we need to use an opt coupler circuit

Fortunately, there already exists a relay module that consists of both the Opt coupler IC and the Relay Here we will be using a 5V, 10Amp - 2channel relay

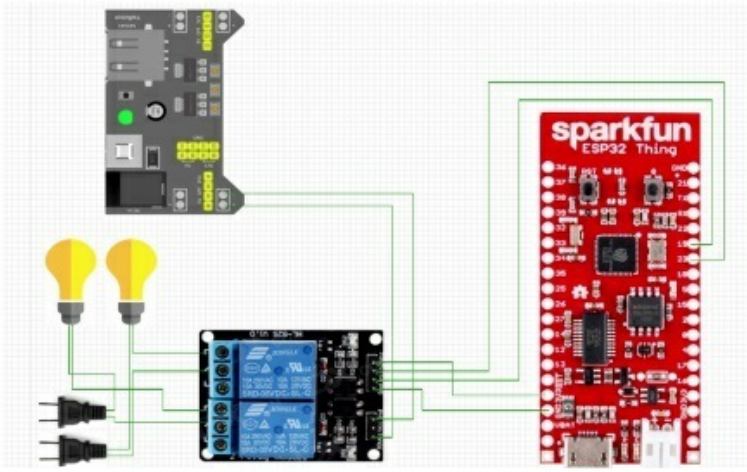
Now let's look at the Relay module itself The Relay module consists of 2 of the Relays which we talked about earlier It also consists of two opt coupler IC's, one for each relay Now on this side, you will see 4 pins VCC, IN2, IN1,

and Ground The VCC pin here is used to power up the opt coupler IC Here it needs to be connected to 3 3V on the Thing IN1 and IN2 are signal pins for relay 1 and 2 respectively So if you need to control Relay1, you need to provide a low signal at IN1 pin Since the Thing operates on a 3 3V level, the HIGH signal will be 3 3V, and the low signal will be 0V The next pin is the ground pin Now the JD VCC pin here is for powering up the relay itself This needs to be connected to 5V Since the ESP32 Thing itself can only supply a maximum of 3 3V you need to connect this pin to an external 5V power supply Now coming to the other side of the module you will find connectors for Normally open, Normally closed and Common



The relay consists of an SPDT or Single pole double throw switch So when you connect the bulb in the Normally closed position, the circuit is already complete and the bulb will be on by default When you apply a low signal to the IN1 pin of the Relay module, the current will flow through the Opt coupler energizing the electromagnet and activating the Relay Now the contact will switch from the normally closed to the normally open position breaking the circuit and hence switching off the bulb But if the bulb is connected in the Normally open position, it will be OFF by default Now when you apply the LOW signal to the IN1 pin, the relay activates and the contact switches to the normally open position, completing the circuit and

hence switching ON the relay



Now that you have seen the working of a Relay, let's look at the connection diagram Check out the Resources section for the same Here the VCC of the relay is connected to the 3 3V pins on the Thing The IN1 pin is connected to PIN19 and the IN2 pin is connected to PIN23 on the Thing The JD-VCC pin is connected to 5V on the breadboard power supply Ensure that all the ground pins, on the Thing the Relay and the power supply are connected The bulbs are connected in the Normally open position

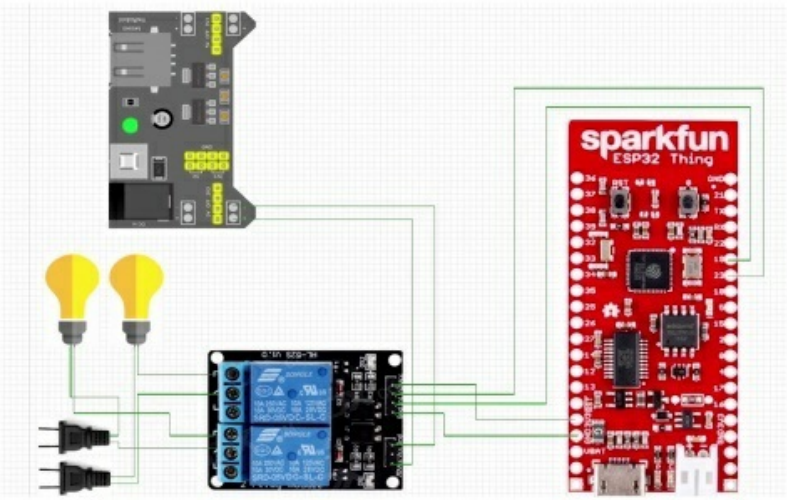
Relay

```
int relayPin1 = 23;
int relayPin2 = 19;

void setup() {
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
}

void loop() {
  digitalWrite(relayPin1, HIGH);
  delay(4000);
  digitalWrite(relayPin2, HIGH);
  delay(4000);
  digitalWrite(relayPin1, LOW);
  delay(4000);
  digitalWrite(relayPin2, LOW);
  delay(4000);
}
```

Now let's look at the code First, since we will be using the GPIO's 19 and 23 on the Thing, hence we need to assign them to the variables relayPin1 and relayPin2 here Next, here we initialize both the pins as output since we will be sending the HIGH and LOW signal from these pins Now in the loop code, we first set PIN 23 as HIGH Thus the first relay will not be activated and the bulb will be switched OFF Next, we set PIN 19 as HIGH after a delay of 4 seconds Thus the second relay will not be activated and the 2nd bulb will also be OFF Next again after a delay of 4 seconds we set PIN 23 as LOW, activating the first relay and switching ON the bulb Similarly, we do this for PIN 19, switching on the Second bulb after a 4-second delay Now if you upload the code and switch on the breadboard power supply as well as the mains power supply, you will see the bulbs lighting up as expected In this project, we learned about what a Relay is and how it works We also learned how to interface the Relay module.

INTERFACING THE RELAY SETUP WITH CAYENNE AND CREATING A PROJECT

We will learn the following How to control the Relay setup from the Cayenne dashboard. Creating a new project on Cayenne



Now let's look at how we can control the relay setup from the Cayenne dashboard. The relay setup connections are the same as seen in the previous Project.

Cayenne_Relays

```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";
int relayPin1 = 23;
int relayPin2 = 19;
String state1;
String state2;

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "6187eaf0-7184-11e9-ace6-4345fcc6b81e";
unsigned long lastMillis = 0;


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
```

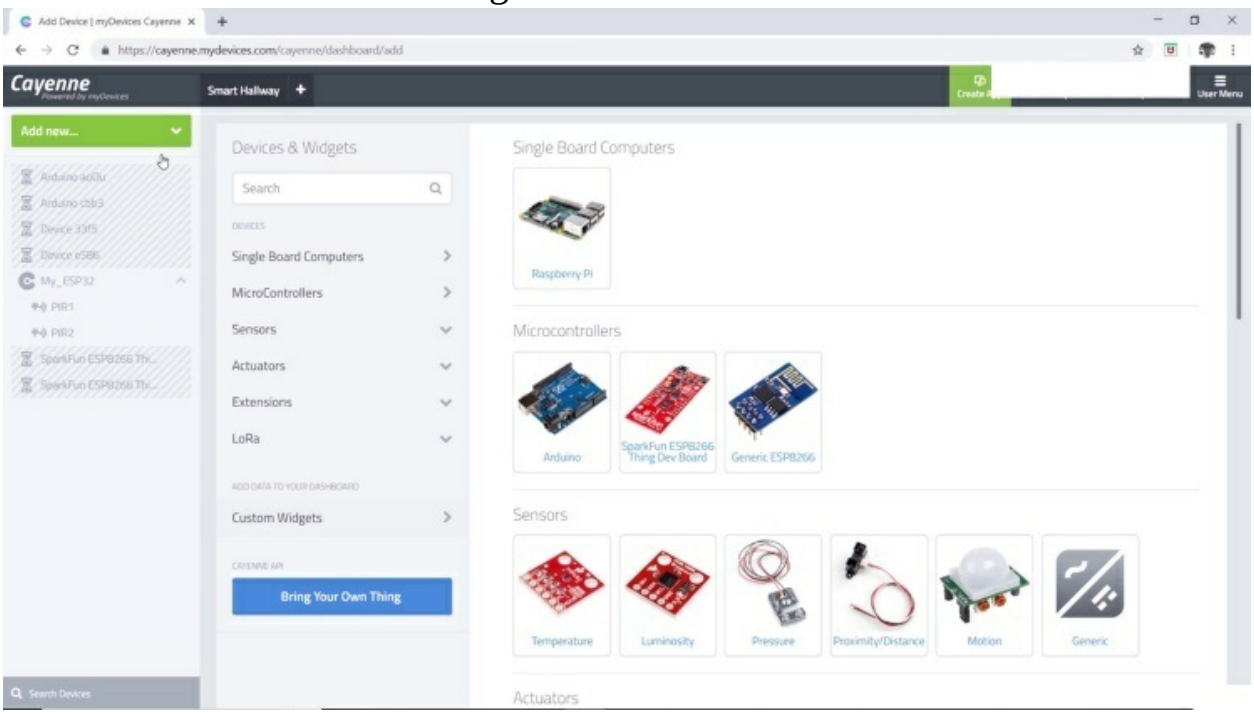
Let's check out the code first. Here we are assigning the signals pins for relays 1 and 2 which are Pin 23 and Pin 19 respectively.

Cayenne_Relays

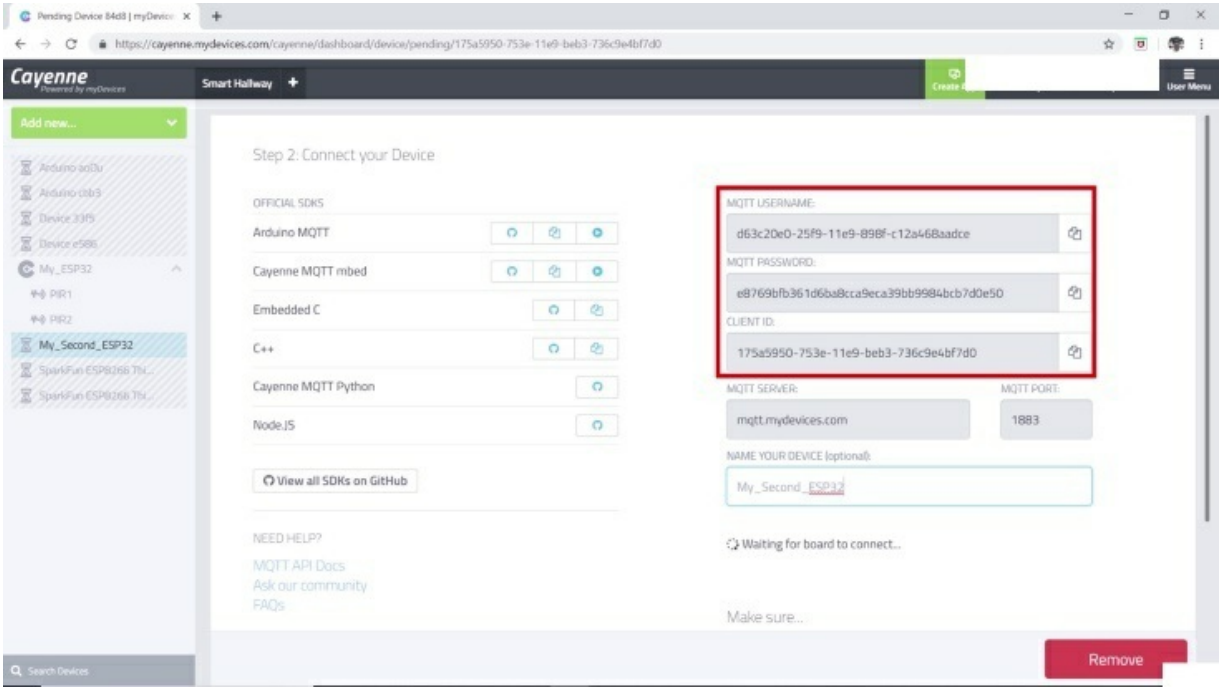
```
#define CAYENNE_DEBUG
#define CAYENNE_PRINT Serial
#include <CayenneMQTTESP32.h>

// WiFi network info.
char ssid[] = "makerdemy1";
char wifiPassword[] = "india123";
int relayPin1 = 23;
int relayPin2 = 19;
String state1;
String state2;

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "d63c20e0-25f9-11e9-898f-c12a468aadce";
char password[] = "e8769bfb361d6ba8cca9eca39bb9984bcb7d0e50";
char clientID[] = "6187eaf0-7184-11e9-ace6-4345fcc6b81e";
unsigned long lastMillis = 0;


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
```

Here we are assigning the Variables state1 and state2 which we will be using later in the code. Now in the Cayenne authentication Information, the username and password will be the same for both the Things which we will be using in the project, however, the Client ID will be unique. Hence to get

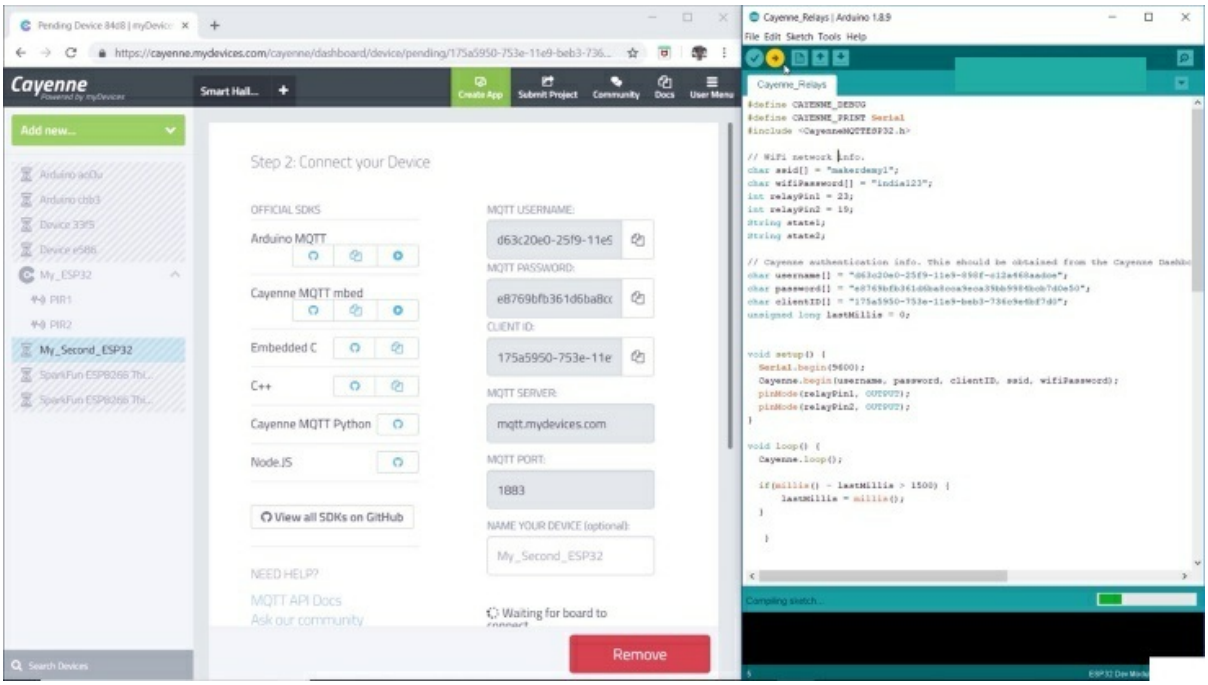
the Client ID for the new Thing



let's go to the Cayenne dashboard first Now let's add our Second Thing to the dashboard. Go to Bring your Thing here.



Now copy the MQTT Authentication information and paste it into the code here.



Name the Second device whatever you want. You need to upload the code first to connect the Thing to the Cayenne dashboard Once you do that, the Cayenne MQTT connection will be established and the new device will be online on your dashboard.

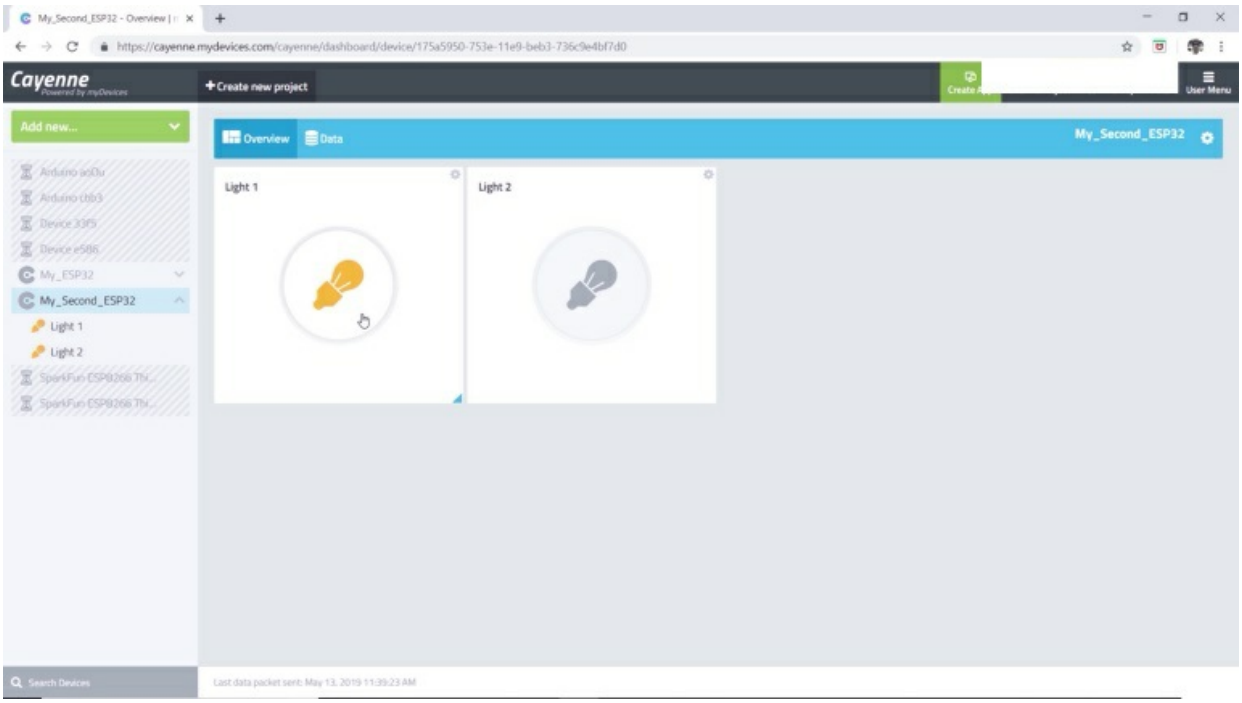
```
char clientID[] = "6187eaf0-7184-11e9-ace6-4345fcc6b81e";
unsigned long lastMillis = 0;


void setup() {
  Serial.begin(9600);
  Cayenne.begin(username, password, clientID, ssid, wifiPassword);
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
}

void loop() {
  Cayenne.loop();

  if(millis() - lastMillis > 1500) {
      lastMillis = millis();
  }

   }
```

Now let's jump back to the code here. we are setting the signal pins which we had assigned earlier, relay pins 1 and 2 as output. Inside loop code. we do not

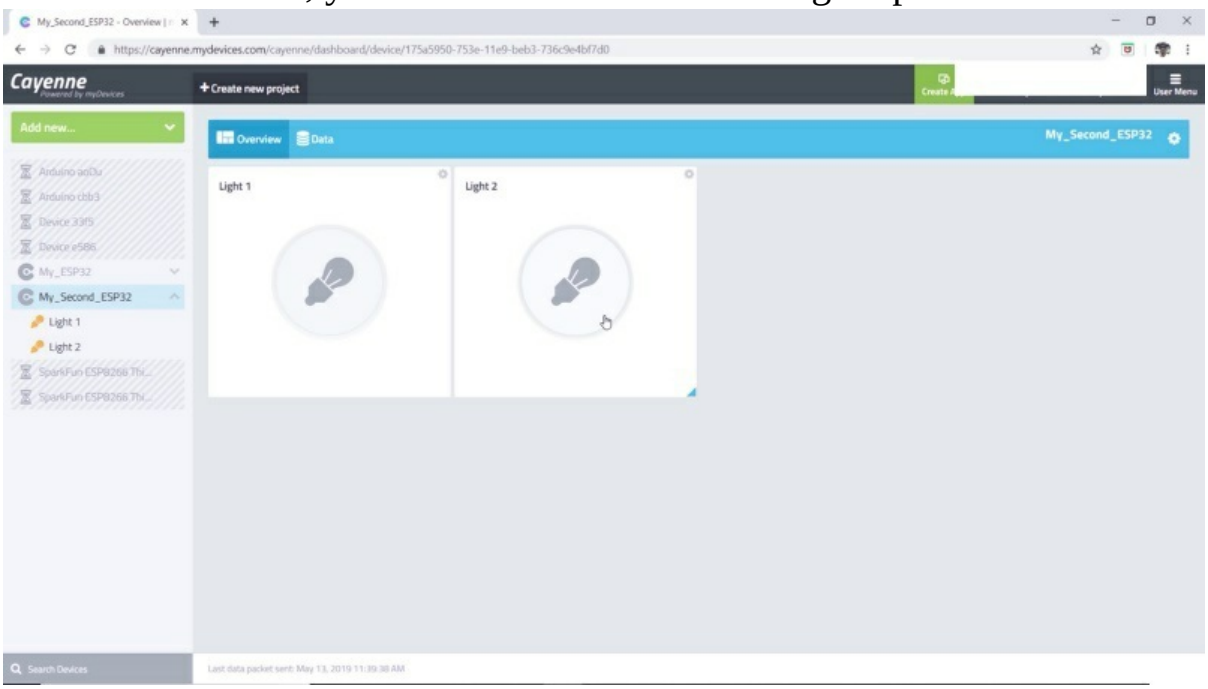
need to include anything because we will be processing actuator commands from the dashboard and not sending and not sending any data to the dashboard.

```
CAYENNE_IN(2)
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
//ss message here. If there is an error set an error message using getValue.setError(), e.g getValue.s                    je");
state1 =  getValue.asString();

if(state1 == "1")
{
  digitalWrite(relayPin1, LOW);
  }
else
{
  digitalWrite(relayPin1, HIGH);
  }
}

CAYENNE_IN(3)
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
//ss message here. If there is an error set an error message using getValue.setError(), e.g getValue.setError("Error message");

state2 =  getValue.asString();

if(state2 == "1")
{
  digitalWrite(relayPin2, LOW);
  }
else
{
  digitalWrite(relayPin2, HIGH);
  }
```

here. we will be using two Cayenne In functions, one to control relay1 and the other to control Relay2 For the first Cayenne In function, we will be using channel number 2. Here we will be using the state1 variable to store the state of the button widget which we used to control relay 1. Now if the state 1 variables value is 1, it means that the button on the dashboard has been pressed. Hence we set the relayPin1 as low, to activate relay 1 and hence switch on bulb 1. If the state of the button is zero the relay Pin 1 will be set as HIGH to deactivate relay 1 and hence switch OFF bulb 1. For the next Cayenne In function, we will be using channel number 3 We will be using the state2 variable to store the state of the 2nd button. Now if the state2 is 1, it means the 2nd button on the dashboard has been pressed. Hence we set the relayPin2 as LOW here to switch ON bulb 2. If the state2 is zero, we switch OFF bulb 2 by setting relayPin2 as HIGH here. Now before uploading the code, we need to add two button widgets to the dashboard. You need to set the channel number for button 1 as two and the channel number for button 2 as three as we had set the same for the Cayenne In functions in the code.

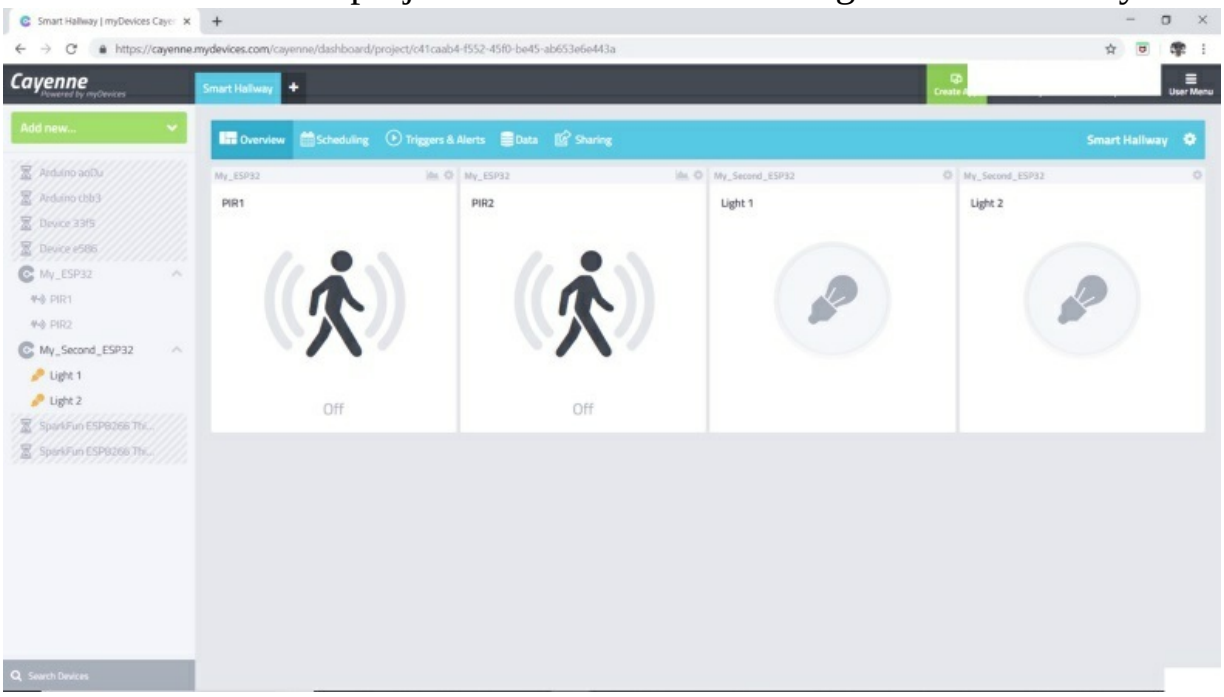


Now that you have added the widgets, let's see the setup in action. When you press button 1 you will see the first bulb light up. When you again press button 1, you will see the second bulb switch OFF When you press the second button here, you will see the second bulb light up.



When you again press the second button you will see the second bulb switch OFF Now we have successfully interfaced our relay setup with Cayenne,

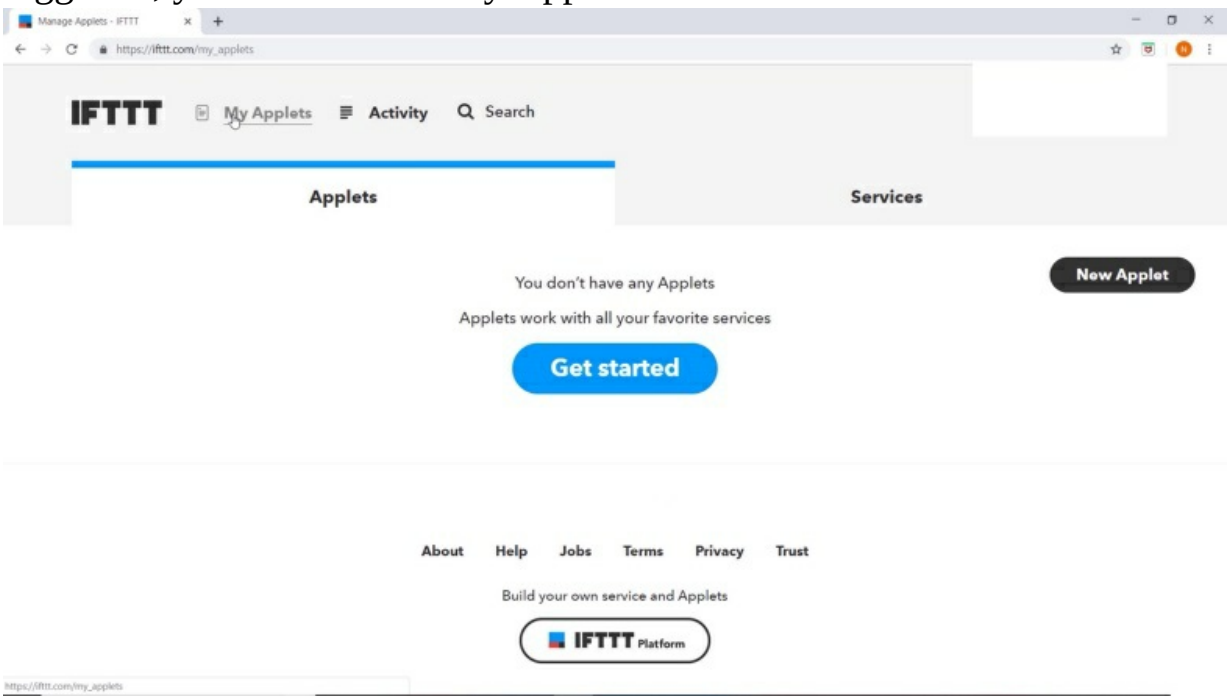
which completes the second part of the Project. Now switching between devices on the dashboard to view the widgets can be troublesome. Wouldn’t it be great if you could view all your Widgets in the same place? Here the Project Feature in Cayenne comes in handy. Let's now look at how we can create a new project on Cayenne. The Project feature in Cayenne allows you to combine widgets from multiple devices on a single custom dashboard. Let's look at how to create a project. Go to create a new Project here You can enter the name of the project Here we will be naming it Smart Hallway.



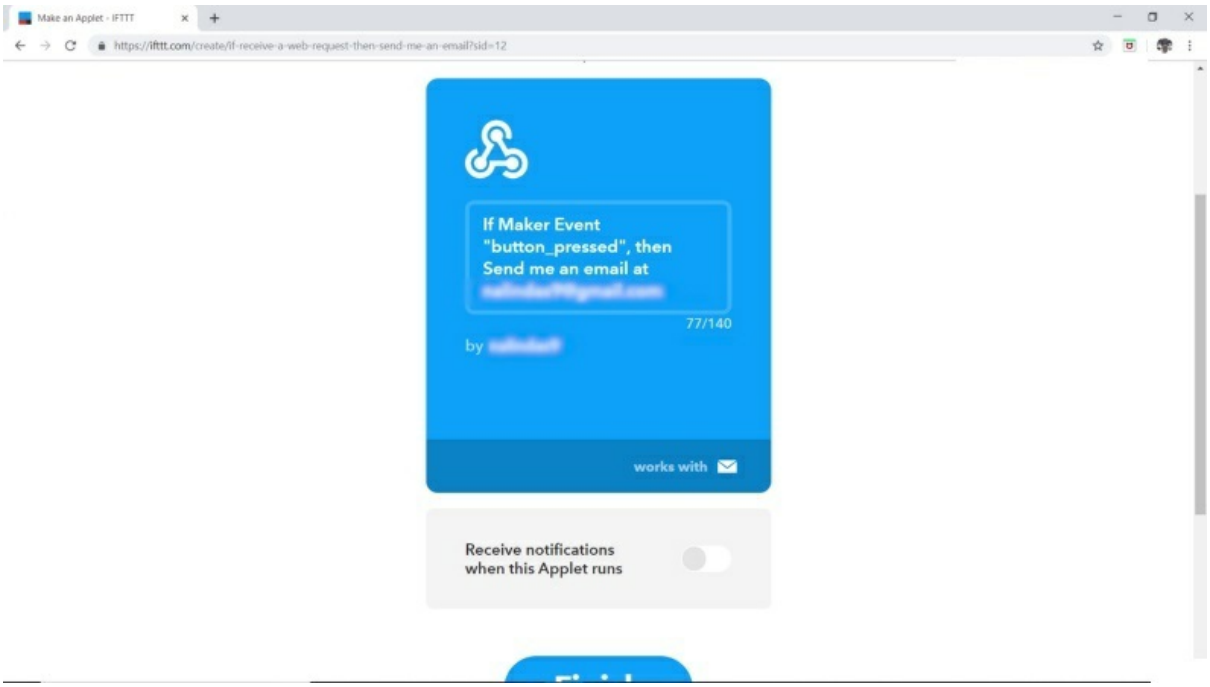
Now in the devices list, you can add the widgets PIR1and PIR2 which we had created for the first Thing to the project dashboard Let's also add the other widgets Light1 and Light2 which we had created for the Second thing. Now we have all the widgets we need for our project on a single dashboard. You can see the particular device name for which the widget is associated here. You can create your set of Triggers, Alerts, and Events for the Project you wish to build. You can create new projects and have a separate dashboard for each one of them, we learned about how to control the Relay setup from the Cayenne dashboard.

SETTING UP NOTIFICATION SMS USING IFTTT

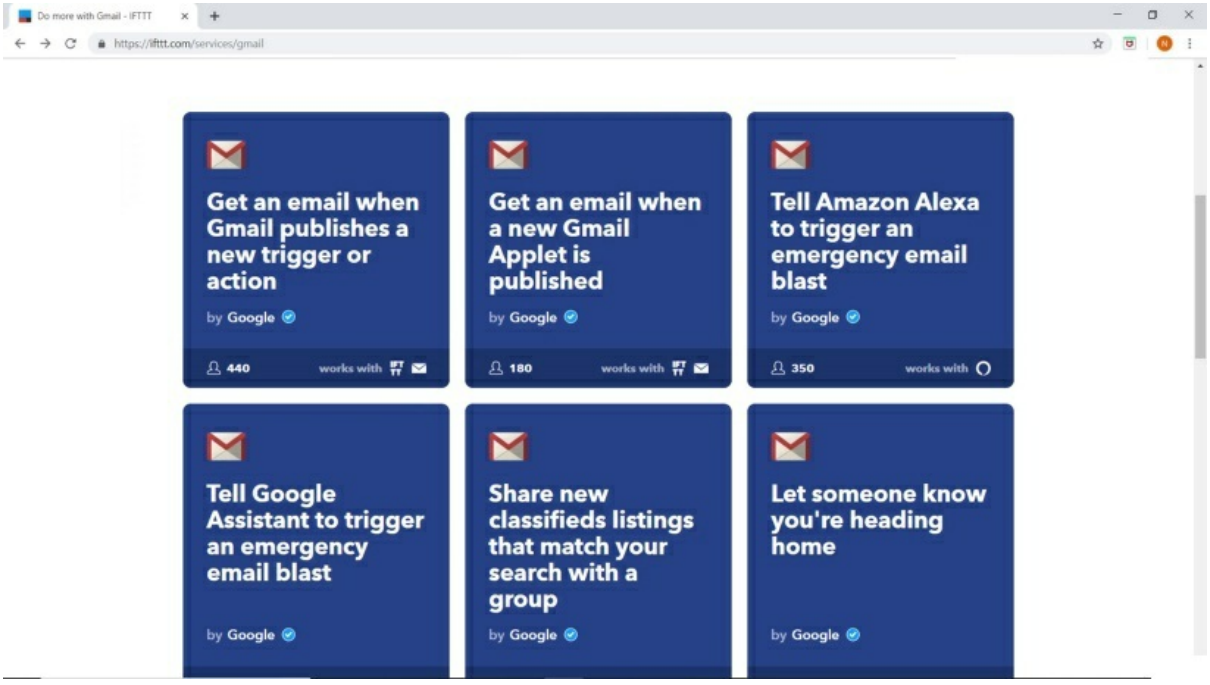
We will learn the following What is IFFFT and how it works What are Webhooks and how can it be used in Cayenne Now wouldn’t it be great if you could expose Cayenne to the web, interacting with various web services like Android SMS, Twitter, Facebook, etc. ? For example, if someone just entered the Hallway, you could get an SMS on your mobile or post a tweet? All of this is possible using IFFFT or If this then that It helps various web services and devices to get talking with each other Lets now see how we can use IFFFT to send a custom notification SMS to our mobile phone when a person enters the hallway Lets go to the IFFFT website now You can either sign up or continue with your Google or Facebook account Now once you are logged in, you will find the My Applets section



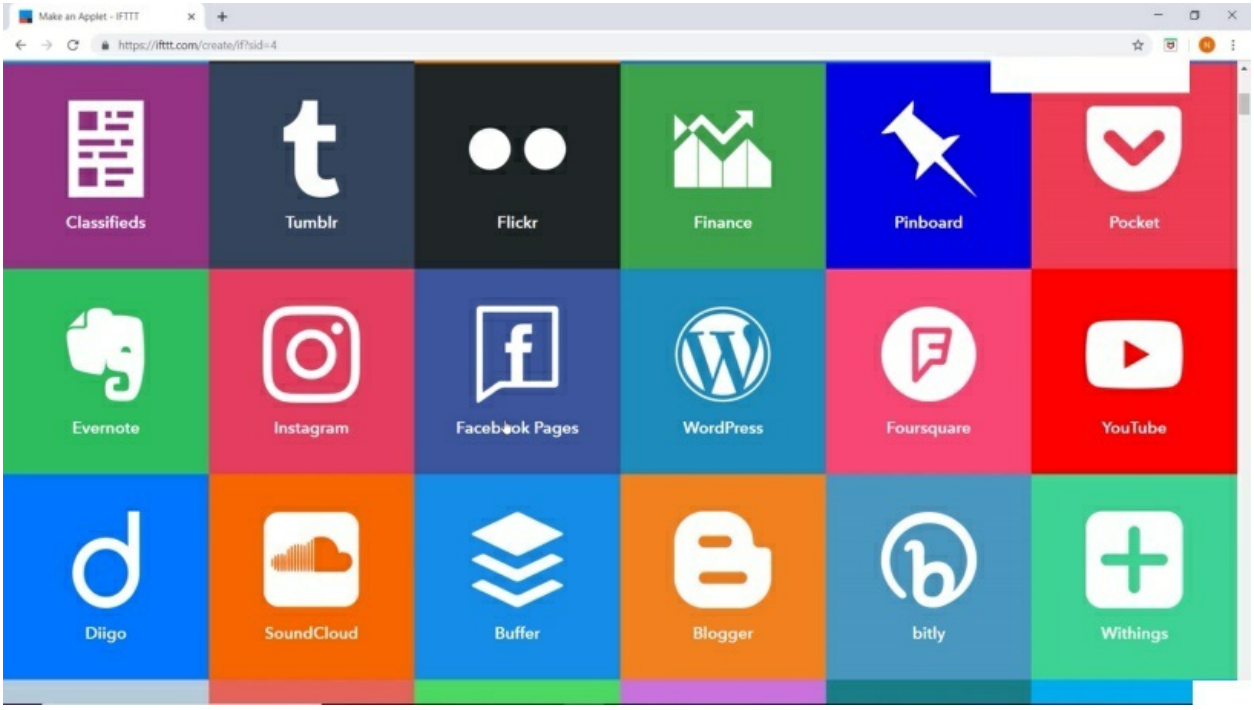
Here you’ll find two tabs Applets and services Here you will find all the applets which you have created



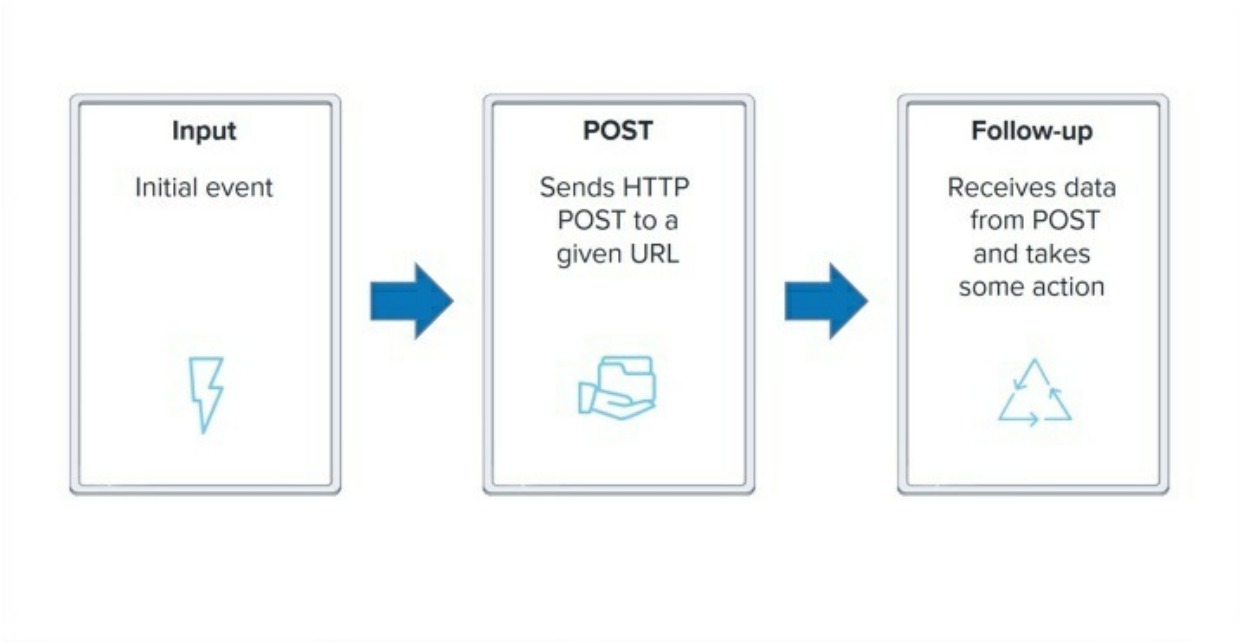
An Applet is nothing but the conditional statement itself if this then that If an event occurs, then an action is performed It is similar to the trigger feature in-game but it offers a lot more school It is similar to the trigger feature in Cayenne, but it offers a lot more scope Now the next tab is the services tab Here you will find all the services you will be using Services are what make up the Applet Each applet will have a This service and a That service connected to it



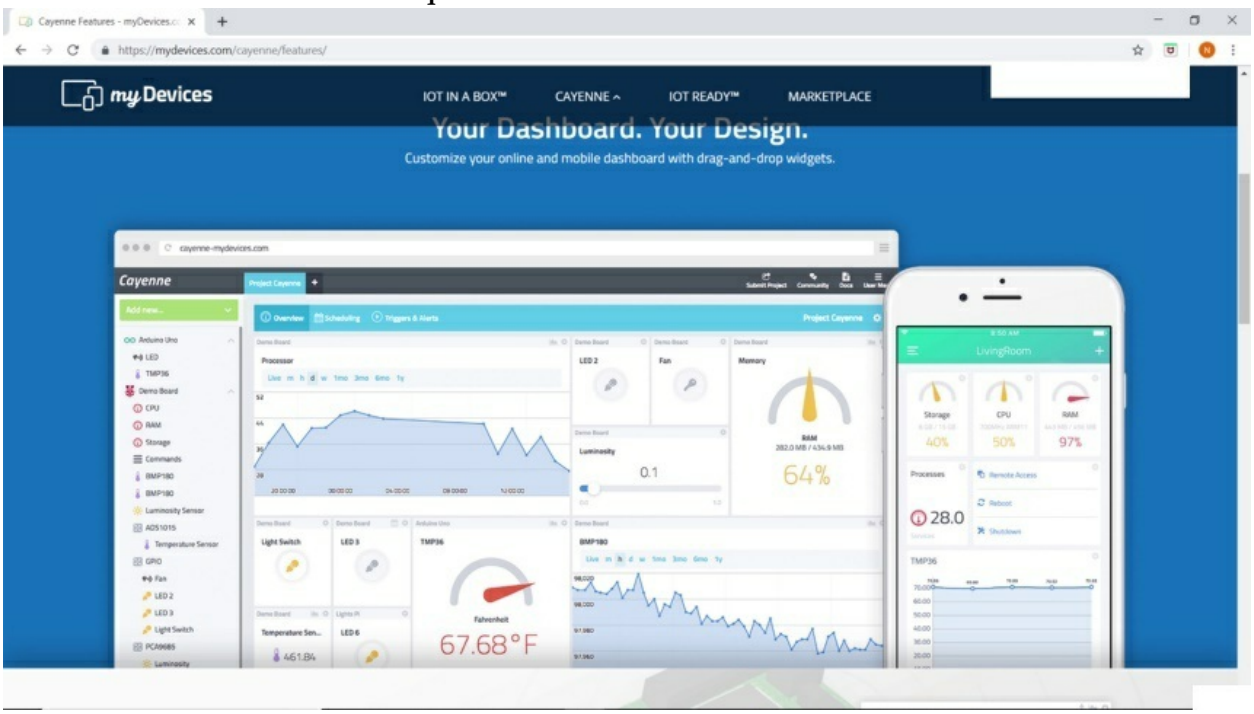
Each service has its set of triggers and actions If you want to explore IFFFT further in-depth about creating custom services you can refer to the IFFFT Platform documentation, the link to which is in the resources section Lets now look at how we can create our applet You can go to New Applet here



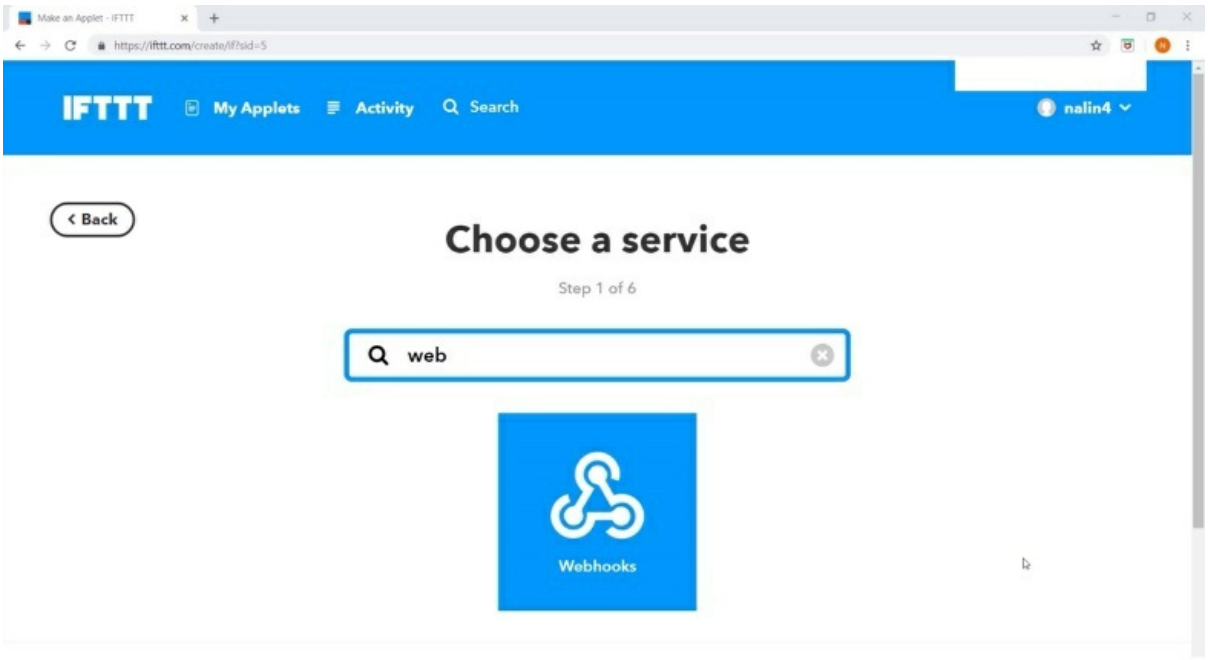
Now you need to select this service Click here for that Now you can choose any service you want as a trigger here we will be using webhooks



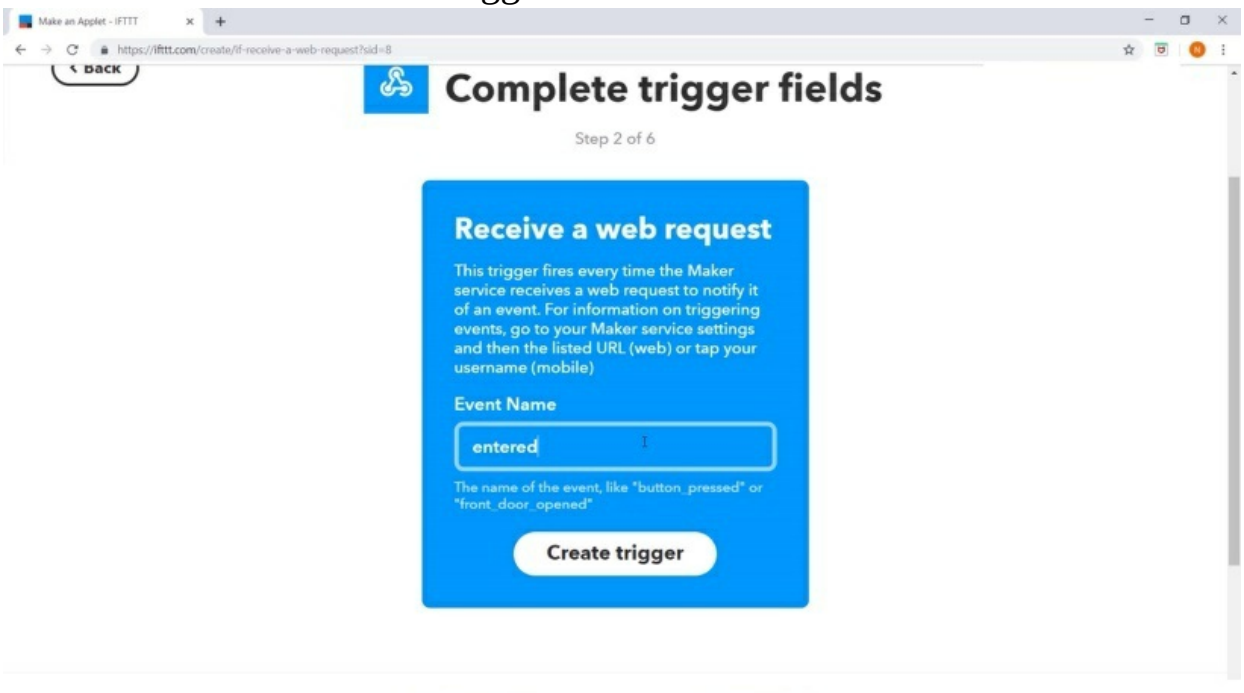
Now let's discuss what webhooks are Webhooks are used to notify and send a message whenever an event has occurred over the Internet You hook it to an event and use the web to send a notification that it happened Now as you know, to transfer any kind of data over the web, HTTP is used So once an event has occurred to which a webhook is hooked to, it will send an HTTP Post request to a given URL HTTP POST, unlike HTTP GET, is used to send data to the server Now IFFFT has support for webhooks, which means it can receive HTTP POST requests whenever an event occurs okay where should it receive this HTTP Post request from?



Cayenne of project This is where the event occurs a person entering the Hallway Cayenne has support for webhooks, from which you can send an HTTP GET to POST, PUT or DELETE request to any URL you want Now to trigger an event in IFFFT from Cayenne, you would need to make a POST request to a specific URL

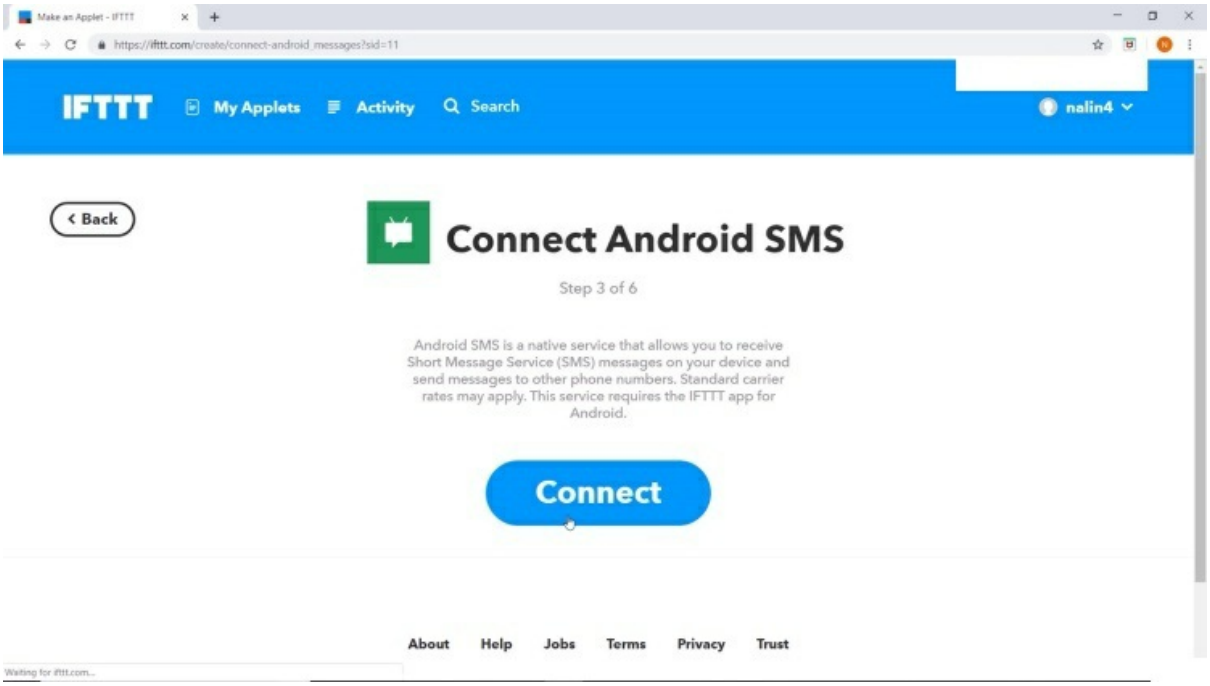


We will learn where we can get that URL in some time but first, let's go back and resume creating our applet Once you click here and click on connect, you will be asked to choose the trigger

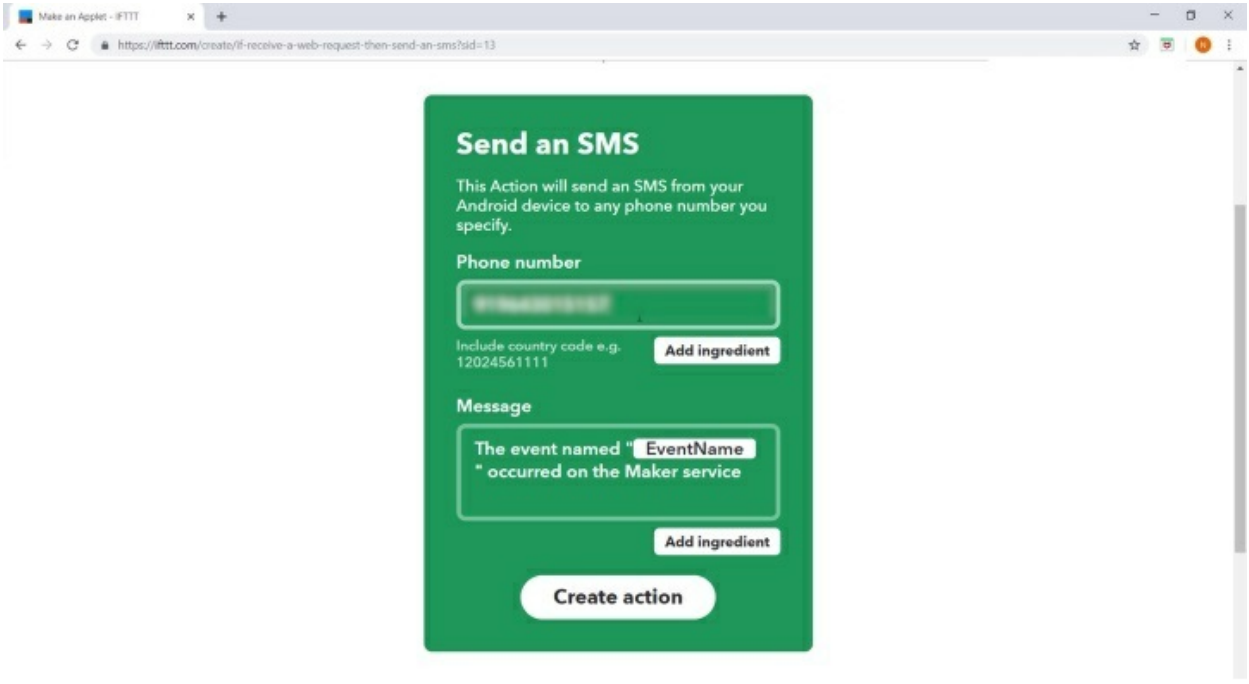


Now in the Event Name field, you can name your event anything you want here We will be naming it this Now you can create the trigger Here you can see that in the If condition, your webhook has to be connected Now let's

check out that services Here you can choose any of the various action services, which will execute if this service is triggered Now since we want to send a custom SMS, lets search for that service

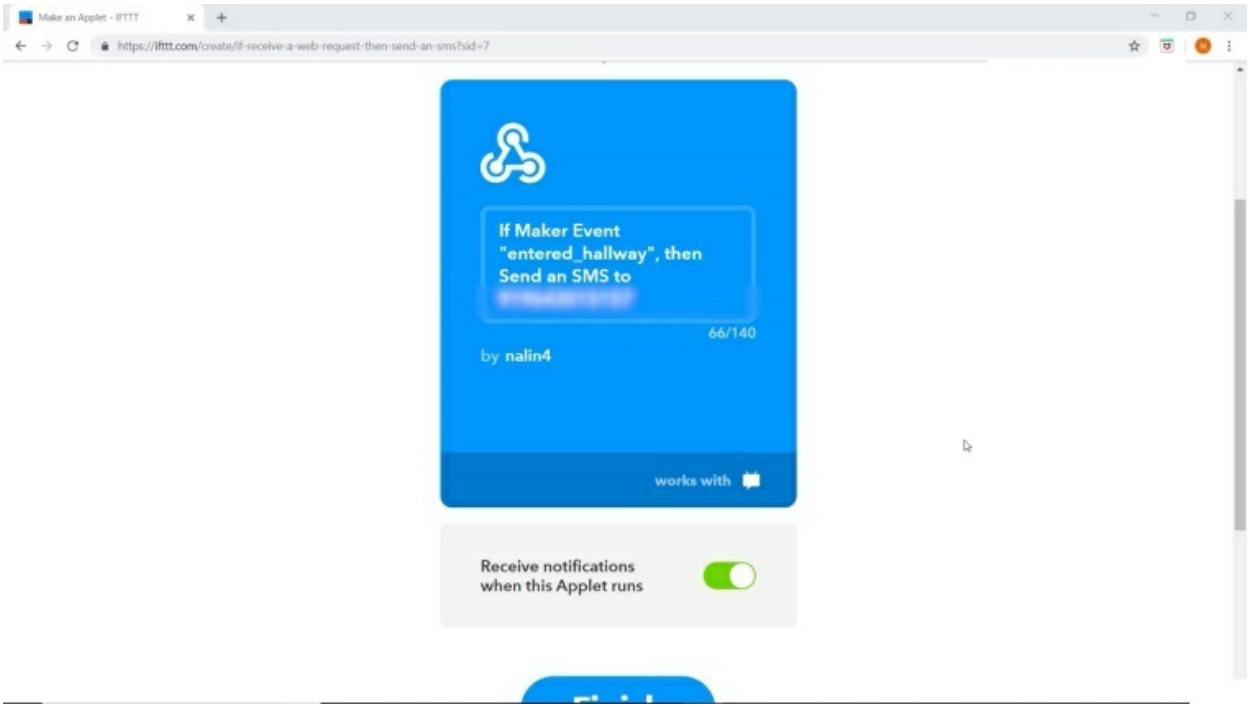


Here you can find the Android SMS service Click on Connect

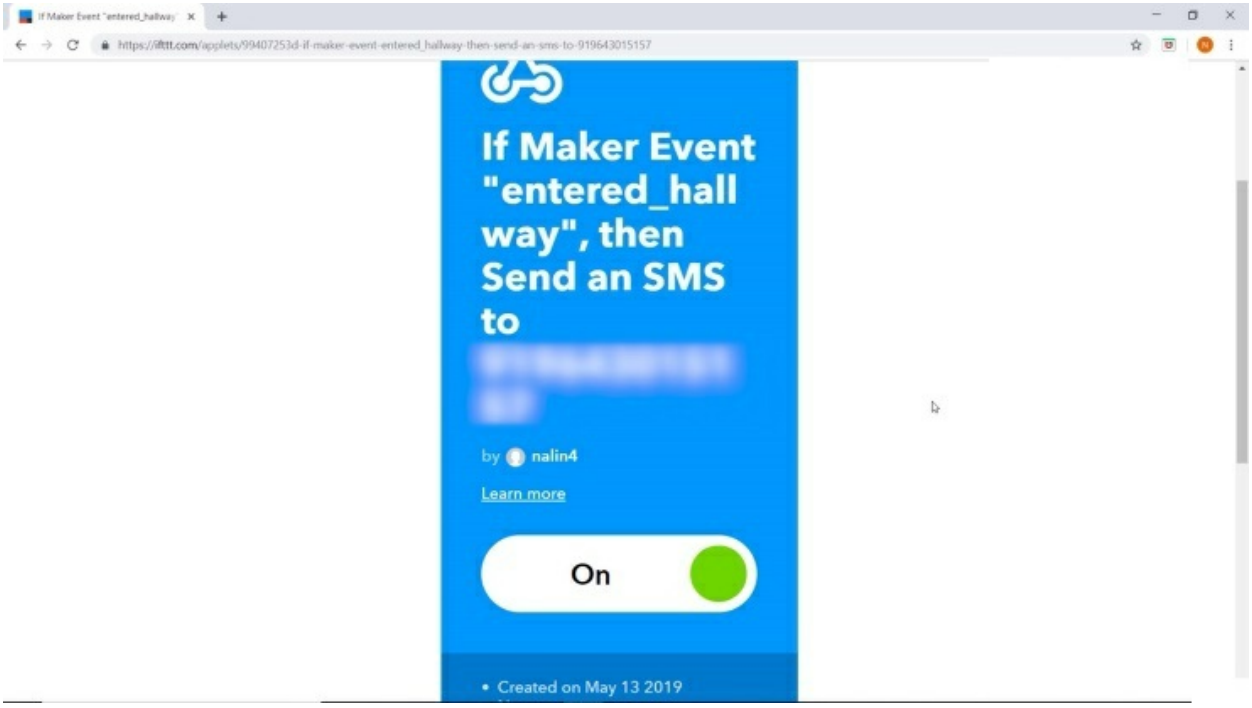


Now you can click here to choose the action, which is to send an SMS Here you can enter the Phone number along with your country code to which you

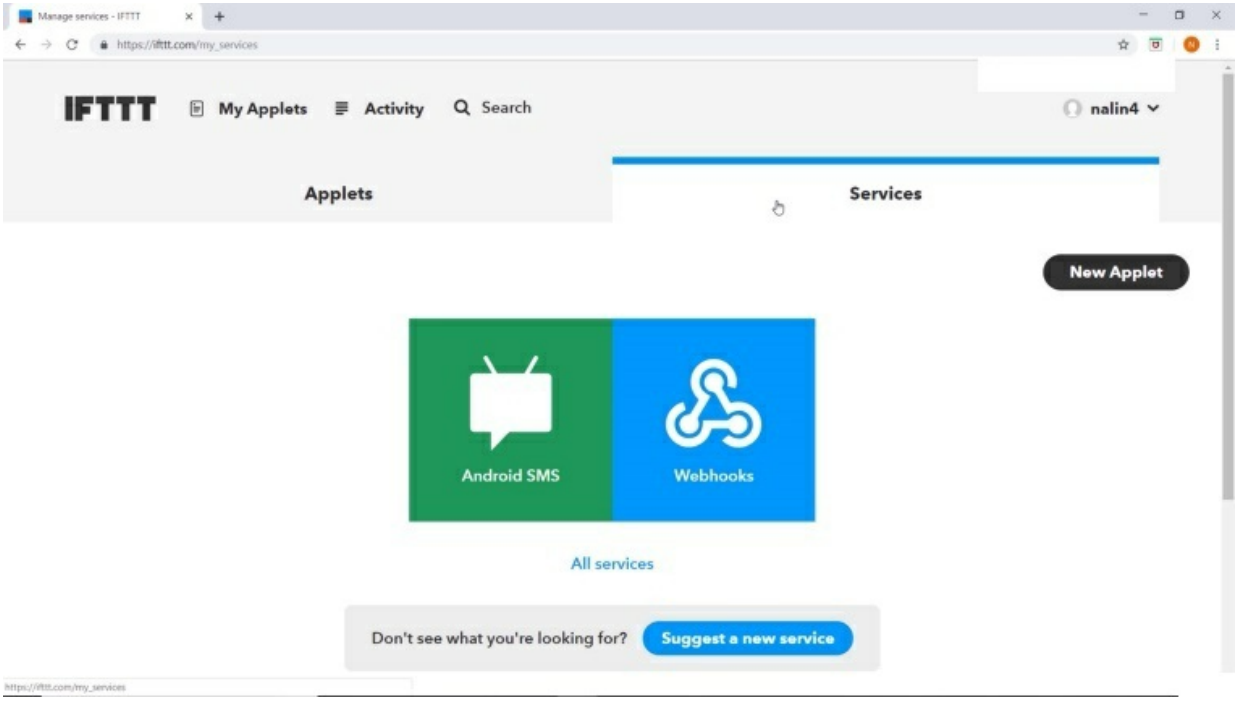
want the notification SMS to be delivered In the Message can type anything you want, and this will be sent as an SMS Here we will type this message Now click on create action here



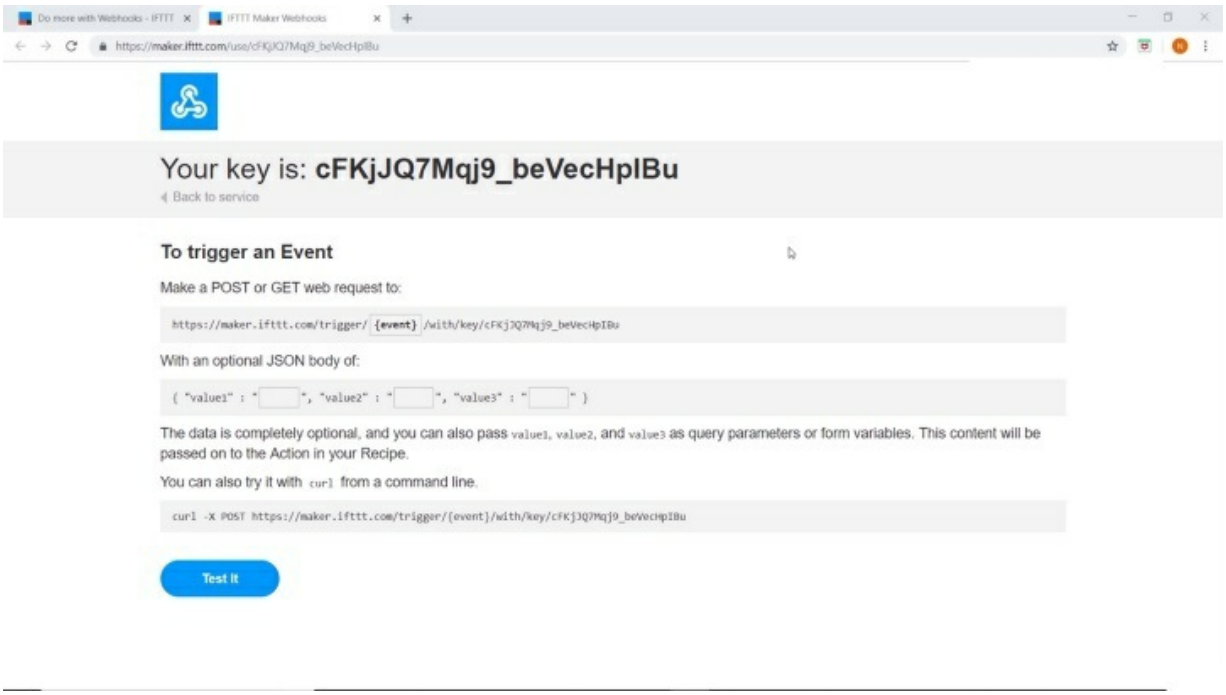
Here you can review the contents of your SMS The SMS character limit is set to a maximum of 140 characters You can also choose to receive a notification every time this object runs



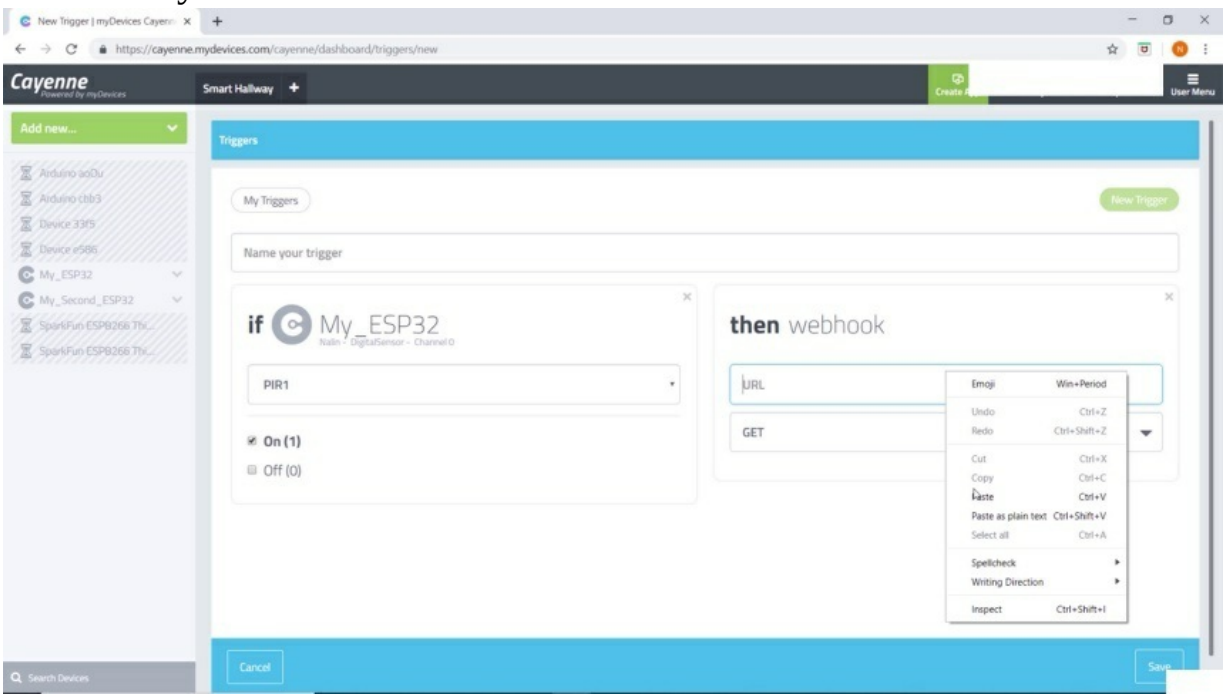
Now once you click on the finish here, you can see that you have successfully created your first Applet You can choose to switch the Applet ON or OFF Here you can see the number of times this Applet has run Now in the activity tab where you'll get information on the date and time you connected a service, your service edit history, your Applet creation history, etc Okay Now that you have created your applet you need to get the U R L to which you need to make the s UDP now that you have created your Applet, you need to get the URL to which you need to make the HTTP Post request so that the event is triggered and the Applet runs To do this you can go to My Applets, and then Services



Here you will find the Webhook service which you used Now in the Webhook documentation here, you will find the URL to which you need to make the POST request

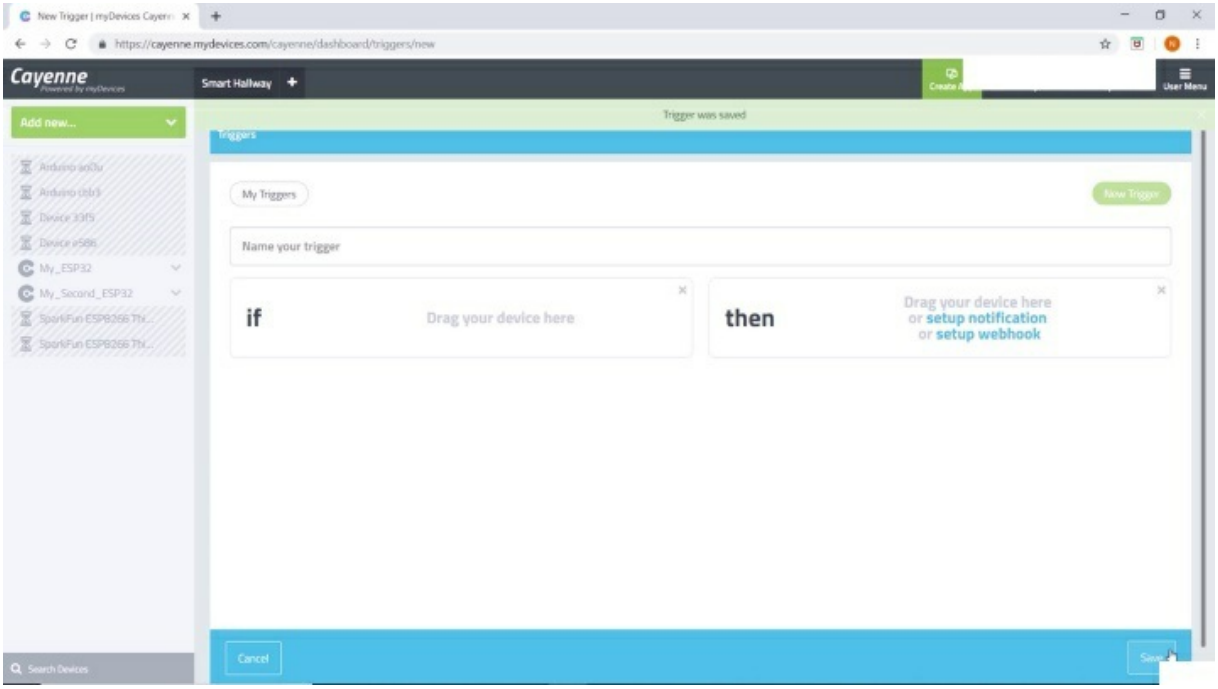


Here you need to enter the Event name you want to be triggered in our case, it was entered the hallway which we had specified while setting the event name Now you can copy this URL In cayenne, let's create a new trigger now When the first PIR sensor detects motion, it means that someone just entered the Hallway

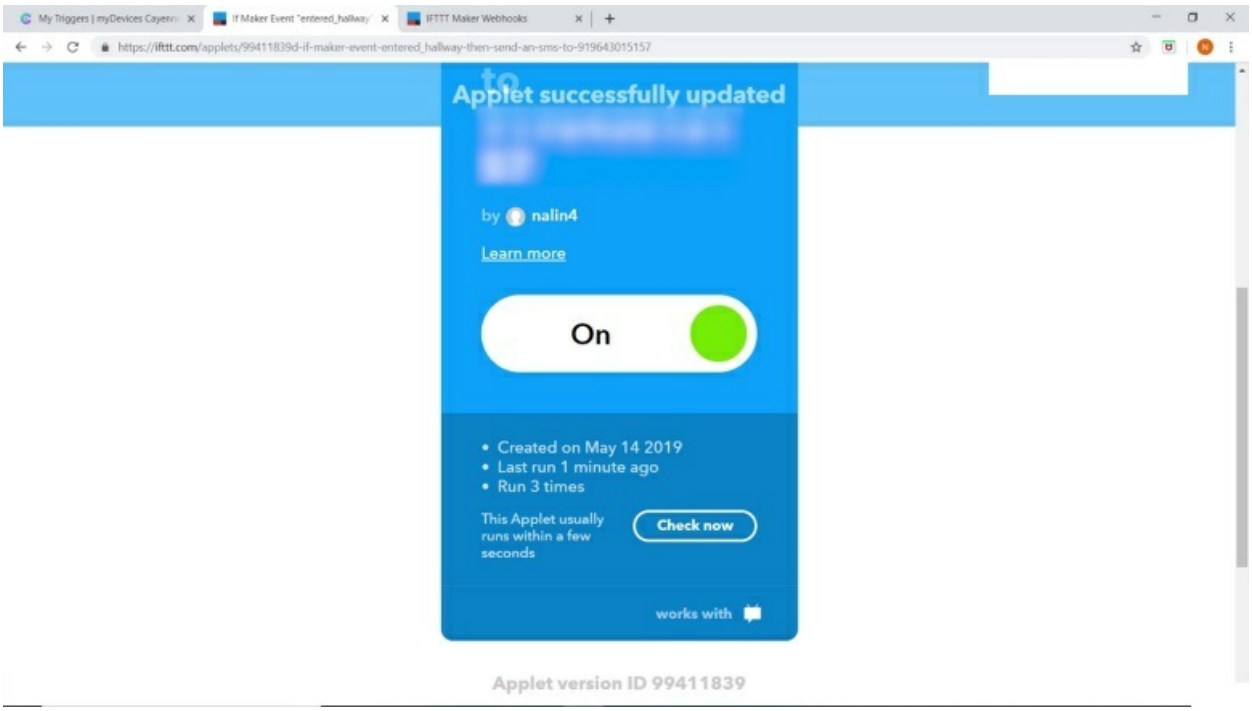


Now the trigger here should be 1 and here you can click on setup Webhook

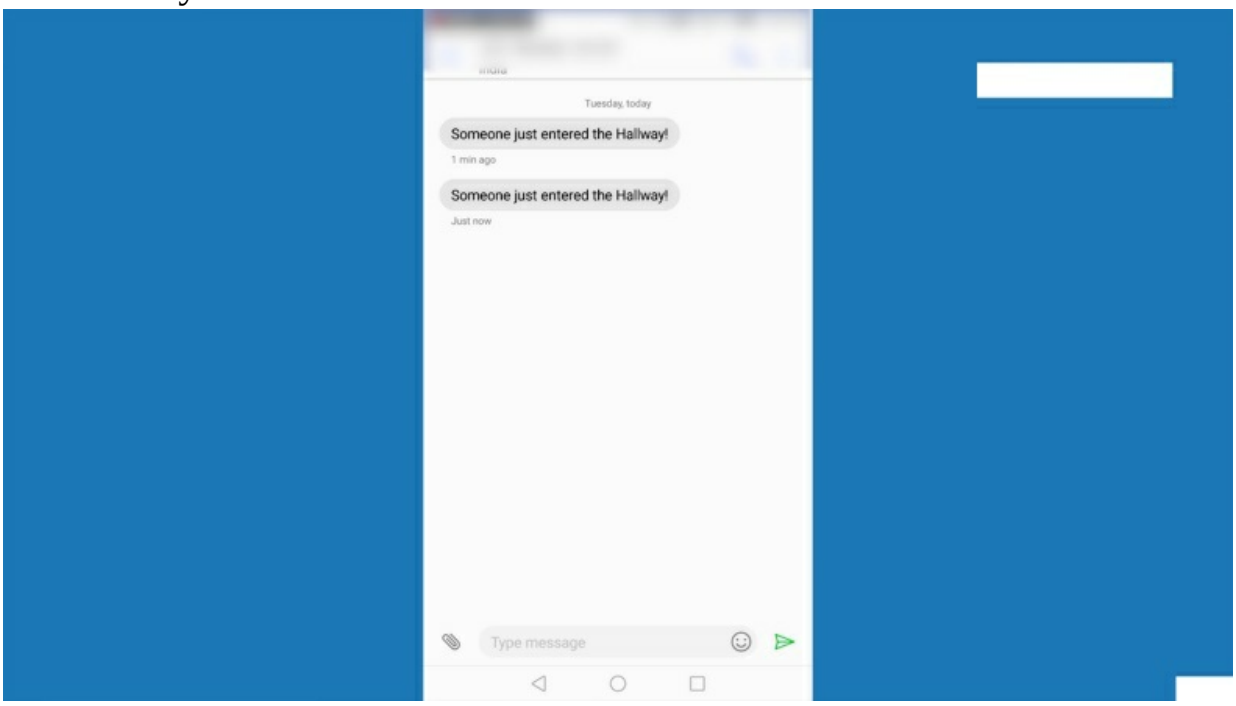
In the URL field, you need to paste the URL you had copied from the Webhook documentation Now since we want to make a POST request to this URL, lets select POST here You don't need to change the type here, as it is by default webhook Now let's name the Trigger Once you save the trigger



let's check whether our trigger ran When you enter the first PIR sensors range, you will see that the Trigger here has run It means that an HTTP Post request has been made to the specified URL



Now let's check-in IFFFT if our Applet has run or not Now when you go to My Applets and select the Applet you created, you will see that it has successfully Run It means that the Action service must have been executed



Now let's check if that is the case Yes, it is We have received the SMS with our custom message notifying us successfully In this project, we learned about what IFFFT is and how it works We also learned about what webhooks

are and how can it be used in Cayenne In this section, we covered the following Interfacing PIR sensors with the Thing Interfacing a Relay with the Thing Interfacing the Relay setup with Cayenne and Creating a Project Integrating the PIR set up and Relay setup using Triggers Setting up notification SMS using IFFFT

THE END