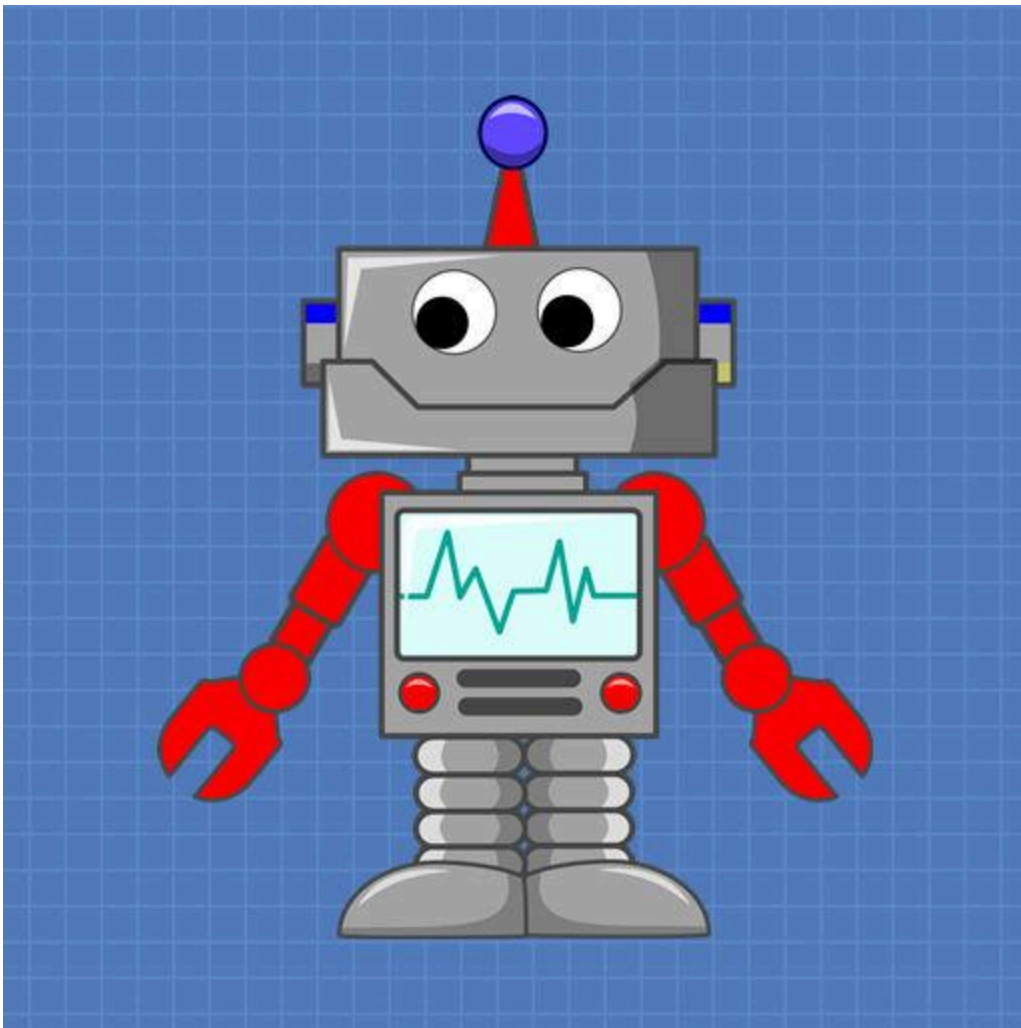


WireClaw - OpenClaw for ESP32



DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

What if you could control an ESP32 microcontroller simply by chatting with it? No Arduino IDE, no uploading sketches, no memorizing pin numbers – just describe what you want and it happens. That is exactly what WireClaw makes possible.



Introduction

[WireClaw](#) is a self-contained artificial intelligence agent that runs directly on an ESP32 microcontroller. You communicate with it through a Telegram chat, and it responds by controlling GPIO pins, reading sensors, creating persistent automation rules, and even connecting to external microcontrollers – all without writing a single line of firmware code after the initial setup.

If you are familiar with OpenClaw – the exciting AI agent framework that lets AI interact with your file system, email, and desktop applications – think of WireClaw as bringing that same concept down to the hardware level. Where OpenClaw gives AI hands to work with your computer, WireClaw gives AI hands to work with the physical world.

<https://dronebotworkshop.com>



In this article, we are going to explore WireClaw from top to bottom. We will look at how it works under the hood, set it up on an ESP32-C6 using an OpenRouter API key and a Telegram bot, and then put it through its paces – controlling LEDs, reading sensors, creating timed rules, and even bridging across to a second microcontroller. Let's get started.

WireClaw

WireClaw is firmware for the ESP32 family of microcontrollers that embeds a complete AI agent directly on the chip. You do not need a server, a Raspberry Pi, or a PC acting as middleware – the AI loop, the rules engine, the device registry, and the web interface all live inside the ESP32 itself.

Supported Hardware

WireClaw runs on three ESP32 variants:

- **ESP32-C6** (recommended – used throughout this article)
- **ESP32-S3**
- **ESP32-C3**

All three require at least 4 MB of flash. The ESP32-C6 DevKitC board used in this article is an excellent choice because it includes an onboard RGB LED and an internal temperature sensor, giving us useful built-in devices to experiment with right away.

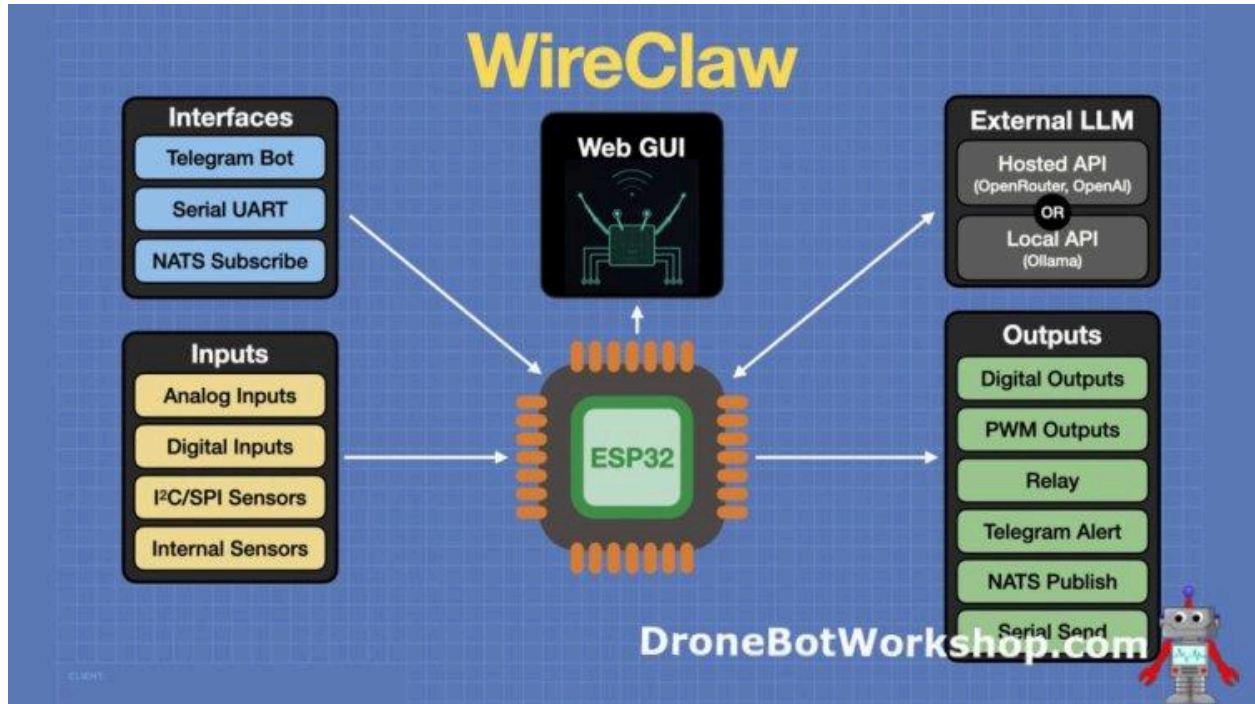
How WireClaw Communicates

WireClaw supports three communication interfaces, and you can use any or all of them simultaneously:

- **Telegram Bot** – the primary interface for most users. Send a chat message from anywhere in the world; the ESP32 polls Telegram for incoming messages and routes them through the AI loop.
- **Serial / UART** – useful for local development, automation scripts, or connecting a host computer directly.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

- **NATS** – a high-performance open-source messaging protocol that enables multiple WireClaw devices to talk to each other and to desktop AI agents such as OpenClaw.



The Dual-Loop Architecture

The most important concept in WireClaw is its dual-loop architecture. Inside the ESP32's main loop, two completely separate processing paths run side by side:

The Rule Loop

The Rule Loop runs on every single iteration of the main loop, continuously, with no network calls and no latency. It reads all registered sensor values, evaluates every active rule, and fires the appropriate actions when conditions are met. Because it never waits for a network response, it reacts in real time.

Rules are edge-triggered: a rule fires once when its condition first becomes true, then automatically resets when the condition clears. This prevents repeated triggering on sustained conditions. All rules are persisted to flash memory in JSON format, so they survive a power cycle or reboot without any intervention.

The Rule Loop supports seven condition types and six action types, giving you a flexible, powerful automation engine that operates completely independently of any AI or network connection once rules have been defined.

The AI Loop

The AI Loop is only activated when a message arrives via Telegram, serial, or NATS. When triggered, it sends the incoming message to a large language model (either cloud-hosted or local), receives a structured response, and executes the requested actions – which may include creating new rules, registering new devices, reading sensors, or configuring the system.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

WireClaw uses an HTTP mode for local LLMs (rather than HTTPS), which saves approximately 40% of RAM utilisation – a significant benefit on a constrained microcontroller. For cloud APIs such as OpenRouter, HTTPS is used automatically.

The key insight is this: **the AI creates the rules, but the rules run without the AI.** Once you have set up your automation through conversation, the ESP32 operates autonomously. The chip does not need to reach out to an LLM every time a sensor crosses a threshold – it already knows what to do.

The Rules Engine

The Rules Engine is the heart of WireClaw's autonomous operation. You define rules through natural language conversation; WireClaw translates your intent into structured rule objects and stores them persistently.

Each rule consists of a sensor to monitor, a condition to evaluate, and one or more actions to take. The engine recognizes seven condition types (including threshold comparisons and time-based triggers) and six action types:

- **GPIO digital output** – set a pin high or low
- **RGB LED control** – set color by name or RGB values
- **Telegram message** – send a notification to your phone
- **NATS publish** – broadcasts a value across the device mesh
- **Serial send** – transmit data out the UART
- **Rule management** – enable, disable, or delete other rules

WireClaw also supports message interpolation in rule actions, allowing dynamic values (such as the actual sensor reading) to be automatically inserted into notification messages.

<https://dronebotworkshop.com>

The Device Registry

Before WireClaw can read a sensor or control an actuator, you register it as a device – again, simply by describing it in conversation. The Device Registry supports seven device types, including digital inputs and outputs, analog inputs, the onboard RGB LED, serial bridges to external microcontrollers, and NATS-connected remote sensors.

The registry also includes virtual clock sensors for hours and minutes (synchronized via NTP), enabling time-based automation without any external real-time clock hardware.

AI Memory

WireClaw includes a persistent AI memory system that stores contextual information about your preferences, device names, and past interactions in the ESP32's flash memory. This memory persists through reboots and is available to the AI loop on every subsequent conversation. You can store preferences (such as your favorite LED color), frequently used values, or any other context that makes your interactions more natural and personalized.

Cloud and Local LLMs

WireClaw works with any OpenAI-compatible API. For cloud use, OpenRouter is an excellent choice: it provides a unified API covering a wide range of commercial models, offers a free tier with sufficient credits to get started, and requires no credit card to sign up. For those who prefer to keep everything local, any Ollama or llama.cpp server on the same network will work, with the added benefits of zero ongoing cost and improved privacy.

The Web Interface

WireClaw includes a full browser-based web interface accessible at the device's local IP address or hostname. From this interface, you can review and edit the system configuration, view and modify the AI memory, inspect all registered devices and their current values, review active rules, and check system status – all without connecting a USB cable or opening the Arduino IDE.

WireClaw Setup

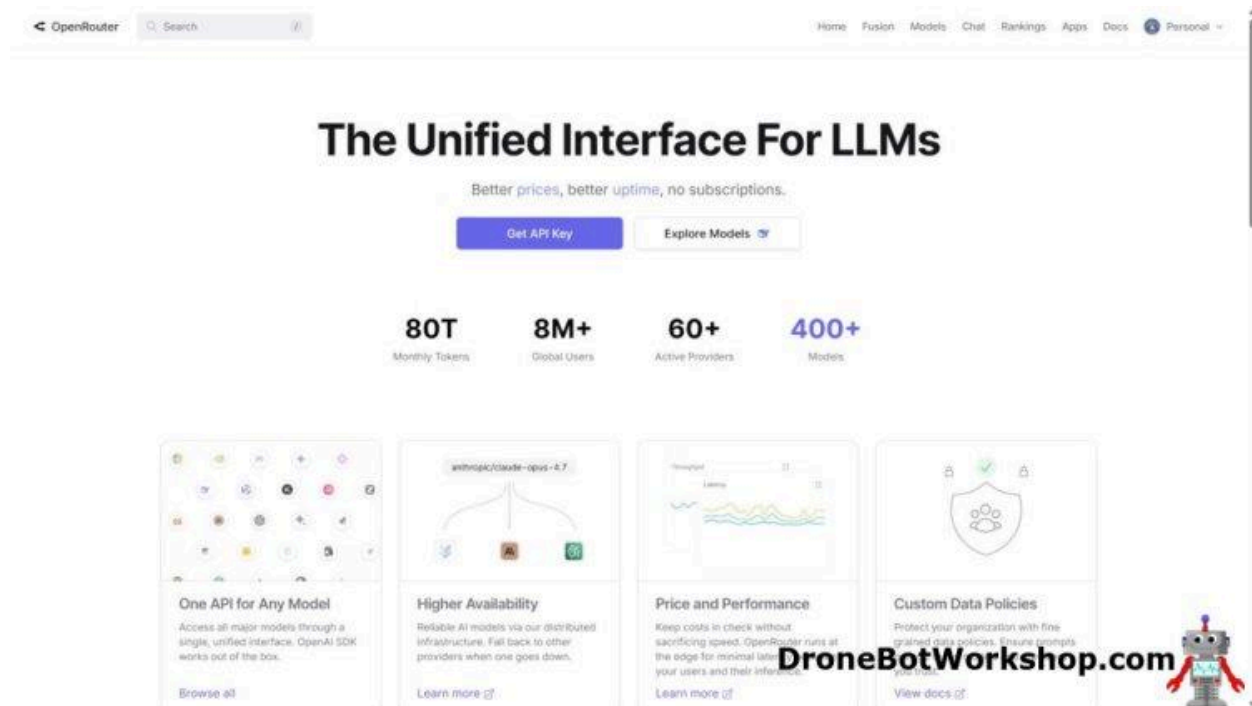
Getting WireClaw running involves three preparation steps followed by flashing and configuring the firmware. Here is everything you need to do before your first chat.

Step 1: Get an OpenRouter API Key

OpenRouter provides a unified API that gives you access to a wide range of large language models through a single endpoint. WireClaw uses it as its default cloud LLM provider.

1. Go to openrouter.ai and create a free account. You can sign in with a GitHub account; no credit card is required to get started.
2. Once logged in, navigate to the API Keys page and click the button to create a new key.
3. Give your key a descriptive name (for example, wireclaw) and confirm.
4. Copy the key immediately and save it somewhere secure – you will not be able to see it again after leaving the page. You will paste this into the WireClaw configuration shortly.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



Note: OpenRouter provides free credits on signup, which is more than sufficient for the experiments in this article. The default model for WireClaw is Google Gemini 2.5 Flash, which is both capable and economical.

Step 2: Set Up a Telegram Bot

WireClaw communicates with you via a Telegram bot that you own and control. You will need three pieces of information from Telegram: a bot token, and your personal chat ID.

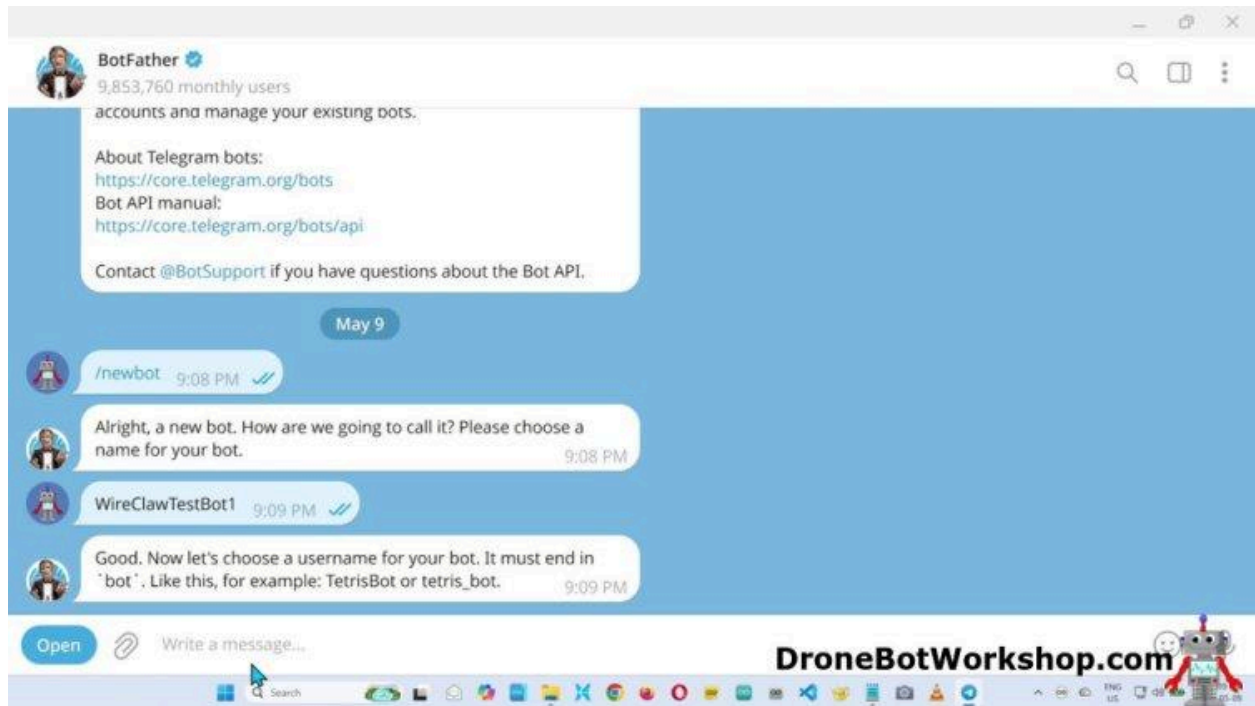
Create a Bot with BotFather

1. Open Telegram and search for **@BotFather** – look for the official account with a blue verified checkmark.
2. Start a conversation and send the command `/newbot`.
3. When prompted, enter a display name for your bot (for example, WireClaw TestBot).

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

4. Enter a username for the bot. Telegram requires usernames to end in bot (for example, mywireclaw_bot).
5. BotFather will reply with a congratulations message containing your bot's **API token**. Copy this token and save it alongside your OpenRouter key.

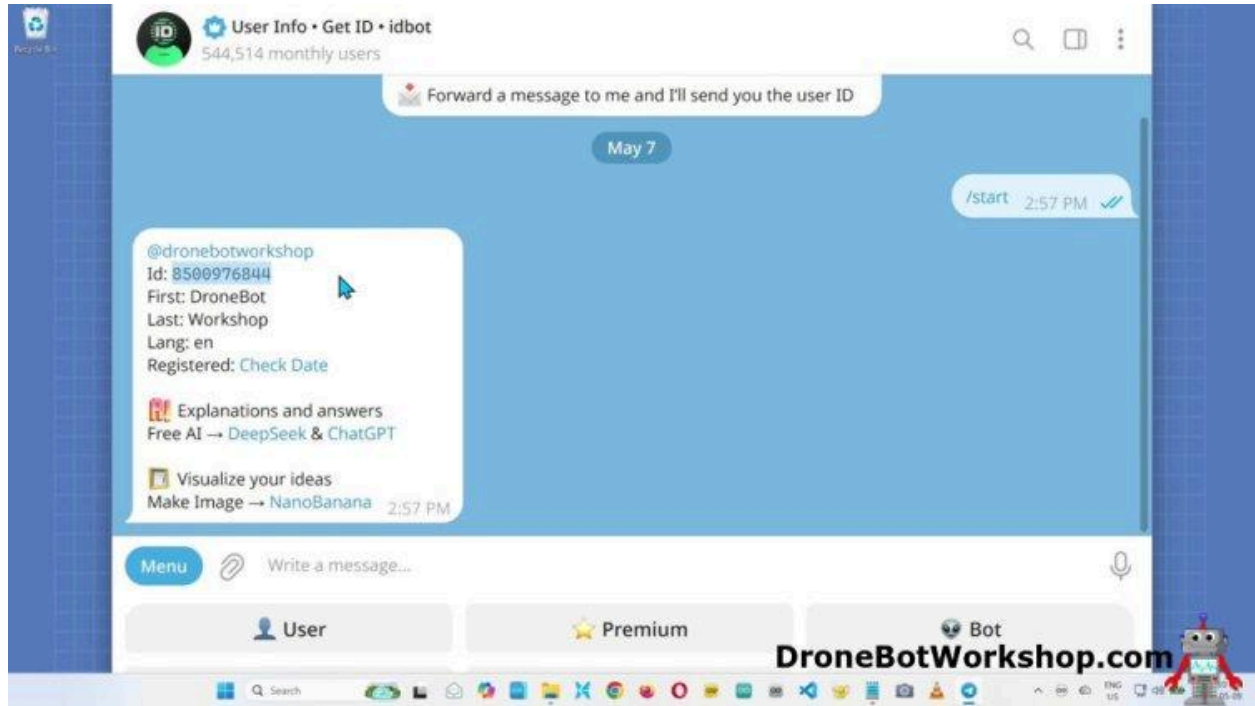


Get Your Telegram Chat ID

1. In Telegram, search for **@userinfobot** (or **@IDBot**) and start a conversation.
2. Send any message (or just /start) and the bot will reply with your numeric Telegram user ID. Save this number – you will need it during WireClaw configuration.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



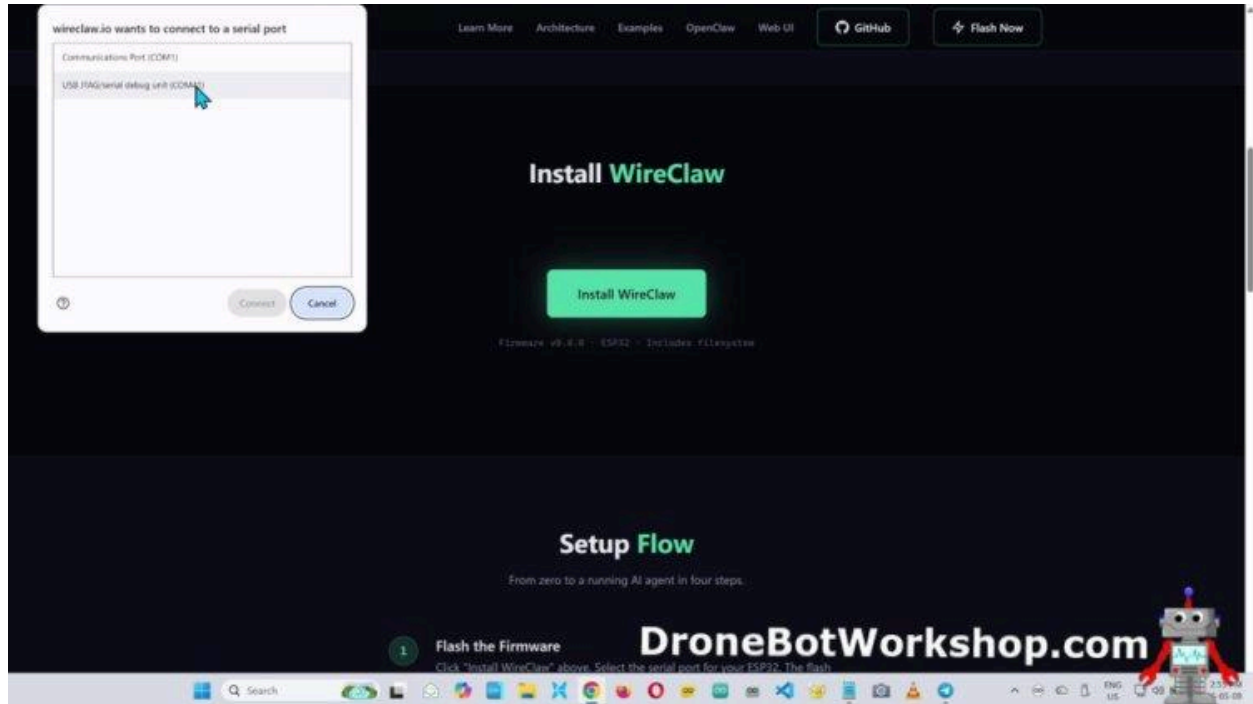
Step 3: Flash WireClaw

WireClaw is flashed using a convenient browser-based utility – no Arduino IDE or command-line tools required.

1. Connect your ESP32 board to your computer via USB.
2. Open a Chromium-based browser (Chrome or Edge) and navigate to the WireClaw Flash Utility page.
3. Click **Install WireClaw**. When prompted to select a port, choose the one labelled JTAG.
4. Tick the option to erase the flash before installing (recommended for a clean start), then click **Install**.
5. The flashing process completes in under a minute. Once done, the ESP32 reboots automatically.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



Step 4: Configure WireClaw

After flashing, the ESP32 reboots into access point mode and broadcasts a Wi-Fi network. Connect your computer to that network, then navigate to 192.168.4.1 to reach the WireClaw setup page.

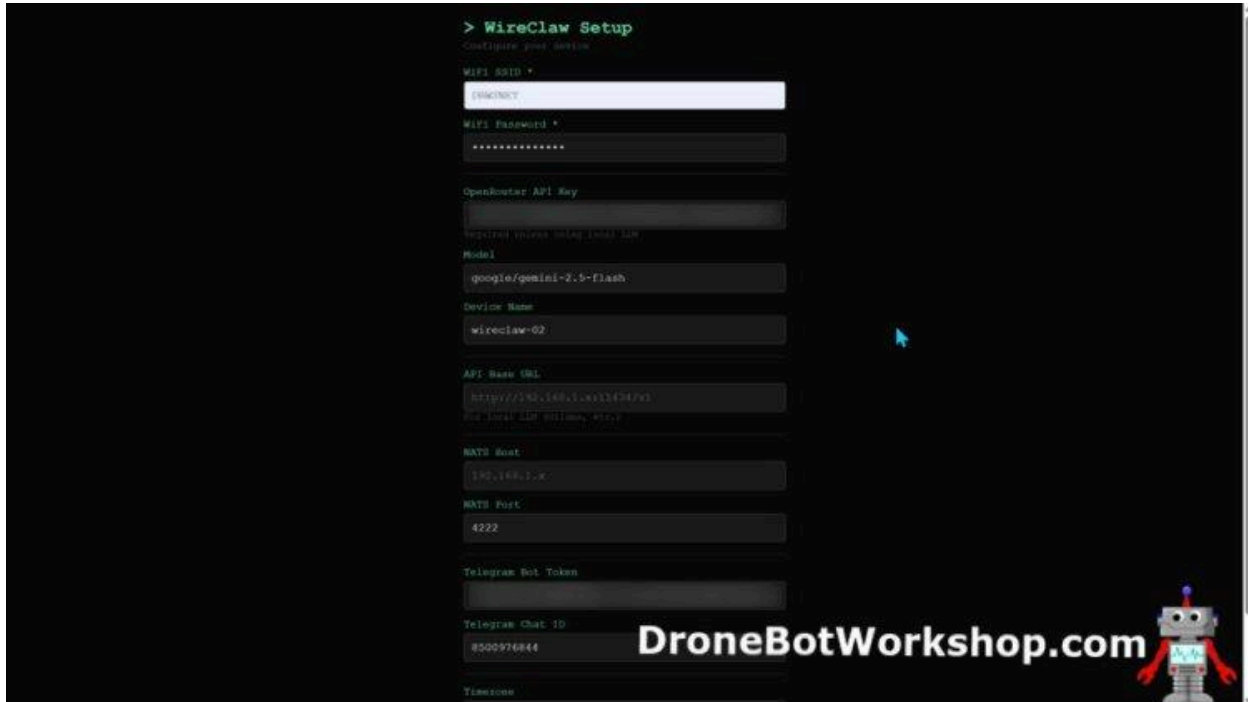
Fill in the following fields:

- **Wi-Fi SSID** – the name of your local Wi-Fi network
- **Wi-Fi Password** – your Wi-Fi password
- **OpenRouter API Key** – the key you saved in Step 1
- **Model** – leave this as the default (Google Gemini 2.5 Flash)
- **Device Name** – a friendly name for this ESP32 (for example, WireClaw-01). You will use this to reach the web interface later.
- **Telegram Bot Token** – the token from BotFather
- **Telegram Chat ID** – your personal chat ID from @userinfobot

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Leave the NATS fields blank for now – we will not be using that feature in this article.
Once all fields are filled in, click **Save & Reboot**.

A screenshot of the WireClaw Setup configuration screen. The screen has a black background with white text and input fields. At the top, it says "> WireClaw Setup" and "Configure your device". Below this are several fields: "WiFi SSID" with the value "DRONEBOT", "WiFi Password" with a masked value "*****", "Openrouter API Key" (empty), "Model" with the value "google/gemini-2.5-flash", "Device Name" with the value "wireclaw-02", "API Base URL" with the value "https://242.168.1.x:3344/v1", "NATS Host" with the value "192.168.1.x", "NATS Port" with the value "4222", "Telegram Bot Token" (empty), "Telegram Chat ID" with the value "8500976844", and "Timespan" (empty). A watermark "DroneBotWorkshop.com" and a small robot icon are visible in the bottom right corner.

> WireClaw Setup
Configure your device

WiFi SSID *

WiFi Password *

Openrouter API Key

Model

Device Name

API Base URL

NATS Host

NATS Port

Telegram Bot Token

Telegram Chat ID

Timespan

DroneBotWorkshop.com

WireClaw will save the configuration to flash and reboot the ESP32, then connect to your Wi-Fi network as a normal client. Within a few seconds, your Telegram bot will send you a message confirming that WireClaw is online, along with its IP address and local hostname (for example, wireclaw-01.local). Navigate to that address in your browser to see the WireClaw web interface.

Testing WireClaw

With WireClaw installed and connected to your network, it is time for your first conversation. For these initial tests we are using the ESP32-C6-DevKitC board with nothing extra connected – the onboard RGB LED and the internal chip temperature sensor are all we need to demonstrate the core features.

<https://dronebotworkshop.com>

Exploring the Web Interface

Before we start chatting, take a moment to explore the WireClaw web interface.

Navigate to your device's address in a browser. You will find several pages:

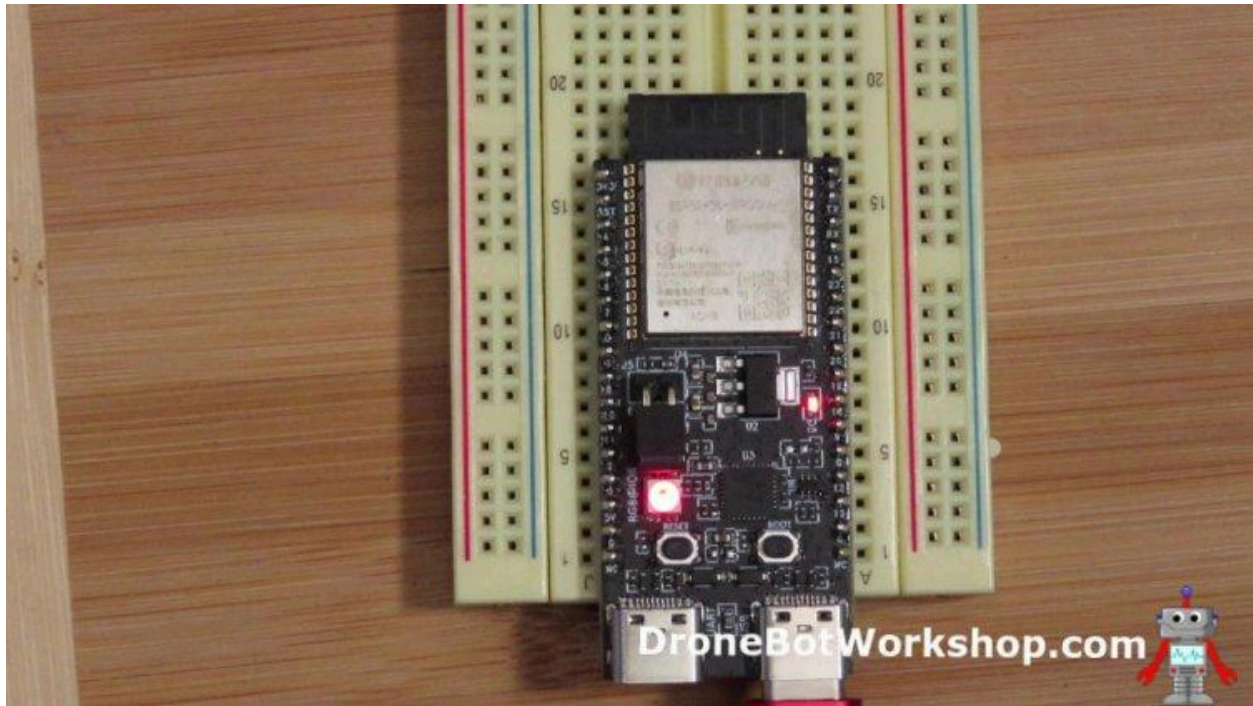
- **Configuration** – the same setup screen from earlier. You can update any setting here and save without reflashing.
- **Prompt** – the system prompt that is sent to the LLM with every conversation. It begins with “you are WireClaw, a helpful AI assistant running on an ESP32...” You can experiment with modifying this to alter the AI's behavior.
- **Memory** – the persistent AI memory store. It starts empty but fills up as you have conversations.
- **Devices** – a list of all registered devices. At this point, you should see the built-in virtual devices: chip temperature, hour, minute, and RGB LED.
- **Rules** – all active automation rules. Currently empty.
- **Status** – firmware version, IP address, free heap memory, and the currently configured LLM.

Changing the LED color

Open Telegram and find the bot you created. Send it a message:

```
Set the LED to red
```

WireClaw will respond confirming the action, and the onboard RGB LED will change to red. Try a few other colors to get a feel for how natural the interaction is. You can use color names, hex values, or even descriptive terms – the LLM will interpret your intent and translate it into the appropriate RGB values.



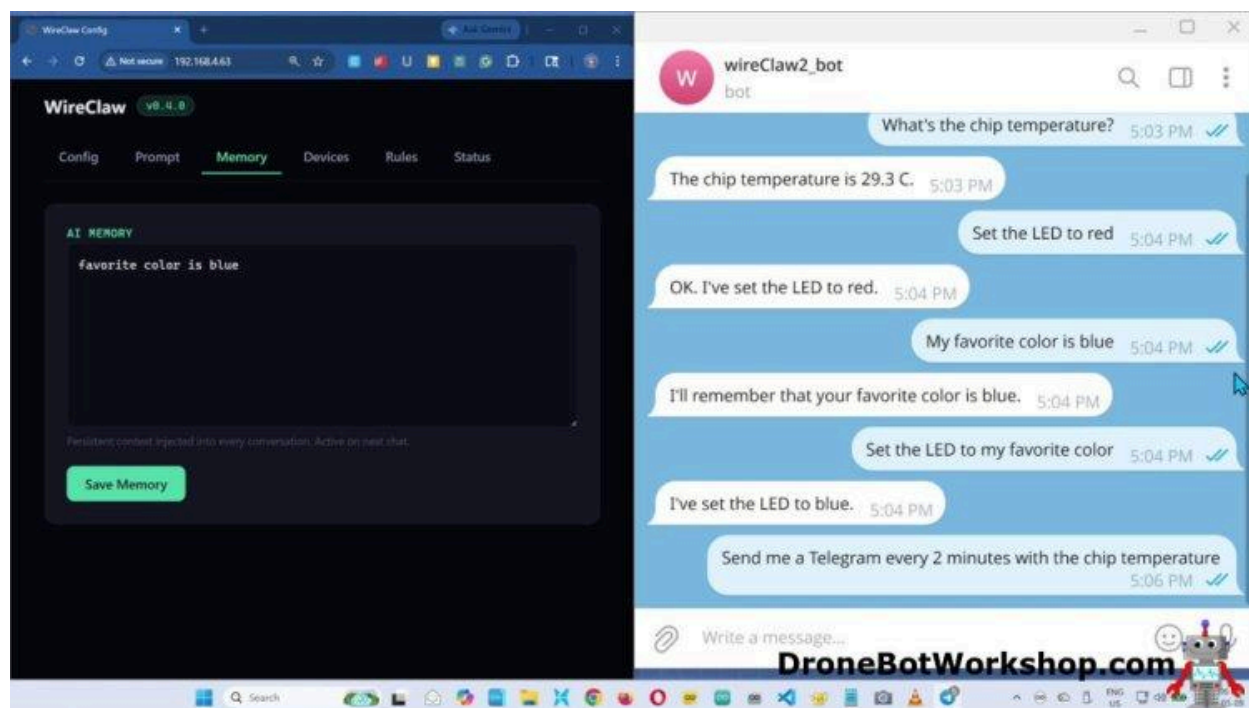
AI Memory – Remembering Your Favorite Color

WireClaw's AI memory lets it retain information about you across conversations and reboots. Let's see it in action:

1. Tell WireClaw: *My favorite color is blue*
2. It will confirm that it has stored this preference.
3. Now navigate to the Memory page in the web interface – you should see the stored fact appear there.
4. Finally, send: *Set the LED to my favorite color*
5. WireClaw will recall that your favorite color is blue and set the LED accordingly, even after a reboot.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

This persistent memory is one of WireClaw's most useful features. You can store device preferences, threshold values, room names, and other contextual information to make your automations feel more natural and personalized.



Reading the Chip Temperature

Send WireClaw a message asking for the chip temperature:

```
What is the chip temperature?
```

WireClaw will read the internal temperature sensor and reply with the current value in degrees Celsius. This is the temperature of the ESP32 silicon itself, which tends to run a few degrees above ambient when the Wi-Fi radio is active.

<https://dronebotworkshop.com>

Creating a Timed Rule

Now let's create our first automation rule. We will ask WireClaw to send us the chip temperature via Telegram every two minutes:

```
Send me the chip temperature via Telegram every 2 minutes
```

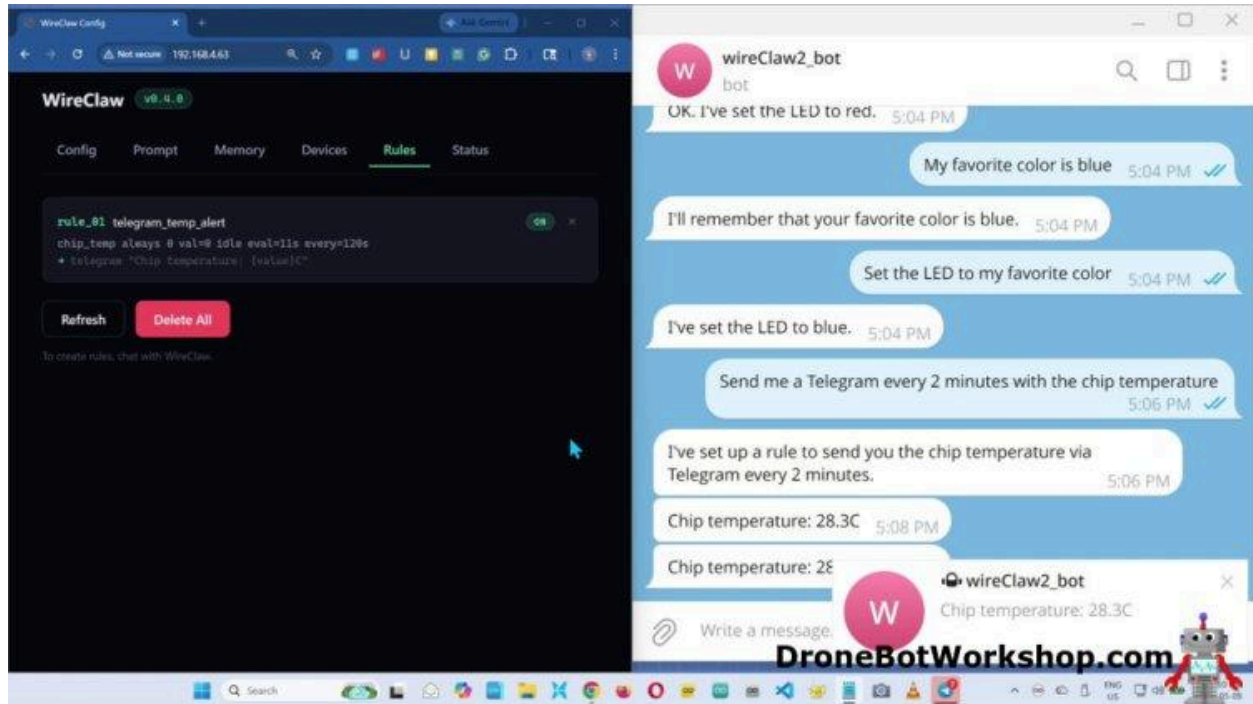
WireClaw will create a time-based rule and confirm it. Navigate to the Rules page in the web interface, and you will see the new rule listed there. After two minutes, you should receive the first automated temperature message in Telegram.

When you have seen the rule working, you can delete it either through the web interface (using the Delete All Rules button) or by asking WireClaw to remove it:

Delete the rule that sends the chip temperature every 2 minutes

WireClaw will confirm the deletion, and the Rules page will be empty again. This illustrates an important point: once rules are running on the ESP32, they operate entirely without the LLM. The AI was only needed to create and delete the rule – the execution itself happens locally on the chip at full speed.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



Using External Devices

So far, we have been working with WireClaw's built-in virtual devices. The platform's real power becomes apparent when you start connecting external hardware. In this section, we will add an LED and a potentiometer to our ESP32 and control them through Telegram.

Components Required

- **ESP32-C6-DevKitC** (the same board from the previous section)
- **Standard LED** (any color)
- **330 Ω resistor** (dropping resistor for the LED)
- **10 k Ω linear potentiometer**
- **Solderless breadboard and jumper wires**

<https://dronebotworkshop.com>

Wiring

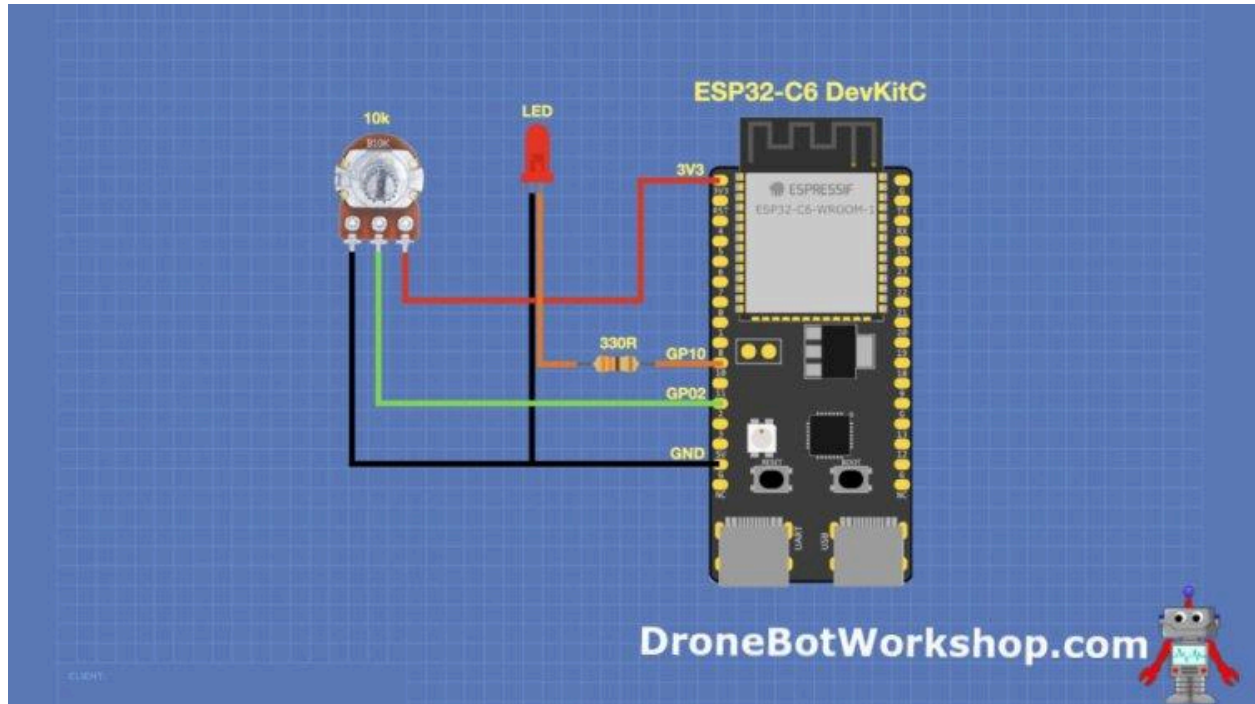
Make the following connections:

LED (Digital Output on GPIO 10)

- LED cathode (short leg) → ESP32 GND
- LED anode (long leg) → one end of the 330Ω resistor
- Other end of the 330Ω resistor → ESP32 GPIO10

Potentiometer (Voltage Divider / Analog Input on GPIO 2)

- One outer pin of potentiometer → ESP32 GND
- Wiper (center pin) of potentiometer → ESP32 GPIO2
- Other outer pin of potentiometer → ESP32 3.3V



Defining Devices in WireClaw

WireClaw needs to know about our new devices before it can work with them. We register them by simply describing them in Telegram:

Registering the LED

Send the following message to your WireClaw bot:

```
Register a digital output actuator called external_led on GPIO pin 10
```

WireClaw will confirm the registration. Refresh the Devices page in the web interface, and you will see **external_led** appear in the list as a digital output.

Registering the Potentiometer

Send:

```
Register an analog input sensor called knob on GPIO pin 2
```

Again, WireClaw confirms, and the device appears in the web interface. You will notice that the Devices page already shows the current potentiometer reading, which updates automatically.

Controlling the LED

Now that the LED is registered, you can control it directly:

```
Turn on external_led
```

The LED on your breadboard will illuminate, and WireClaw will confirm the action. You can also control GPIO pins directly by number if you find that more convenient:

```
Set GPIO pin 10 low
```

The LED will turn off.

Reading the Potentiometer

To read the current potentiometer value:

```
Read knob
```

WireClaw will reply with the current analog reading – a value between 0 and 4095 representing the full range of the 12-bit ADC.

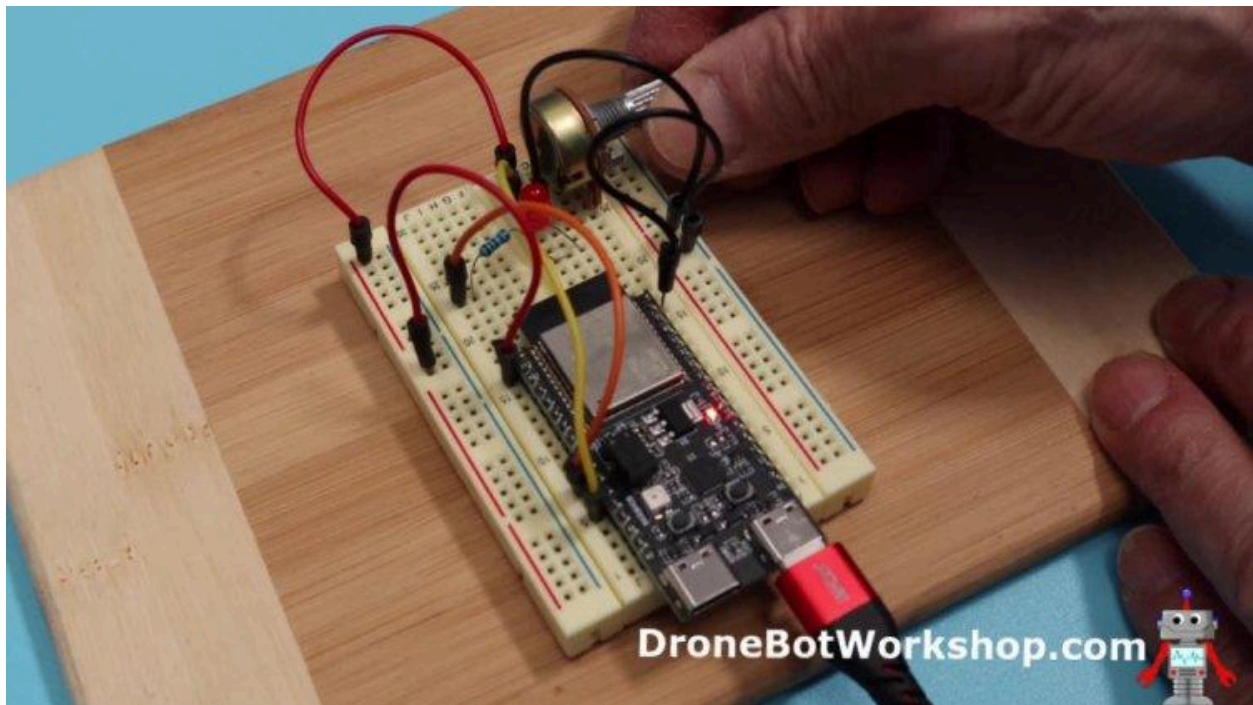
Creating a Rule for the Potentiometer

Let's put the rules engine to work. We will create a rule that sends a Telegram message when the potentiometer is turned up past a certain point:

```
Send me a Telegram message saying "Knob is high" with the current value when knob goes above 2000
```

WireClaw will create the rule. Navigate to the Rules page to see it listed. Now turn the potentiometer up past the midpoint – you should receive a Telegram notification almost immediately, because the Rule Loop is checking the sensor on every iteration of the main loop with no network latency.

Delete the rule when you are done with it, either through the web interface or by asking WireClaw.



Connect an External Microcontroller

WireClaw has a built-in capability it calls Arduino Mode: the ability to receive data from a second microcontroller over UART and treat that data as a registered WireClaw sensor or actuator. The name is generic – you can use any microcontroller that has a UART, not just an Arduino. In this example we will use a Raspberry Pi Pico.

By bridging a second board into WireClaw this way, you effectively extend WireClaw's reach to any peripheral that the second board can read – opening up a vast range of sensors, displays, and actuators that you can then chat with through Telegram.

Components Required

- **Raspberry Pi Pico** (any variant: original, Pico W, Pico 2, etc.)
- **10 k Ω linear potentiometer**
- **Jumper wires (3)**
- The same ESP32-C6-DevKitC from the previous section (existing wiring can remain in place)

Wiring

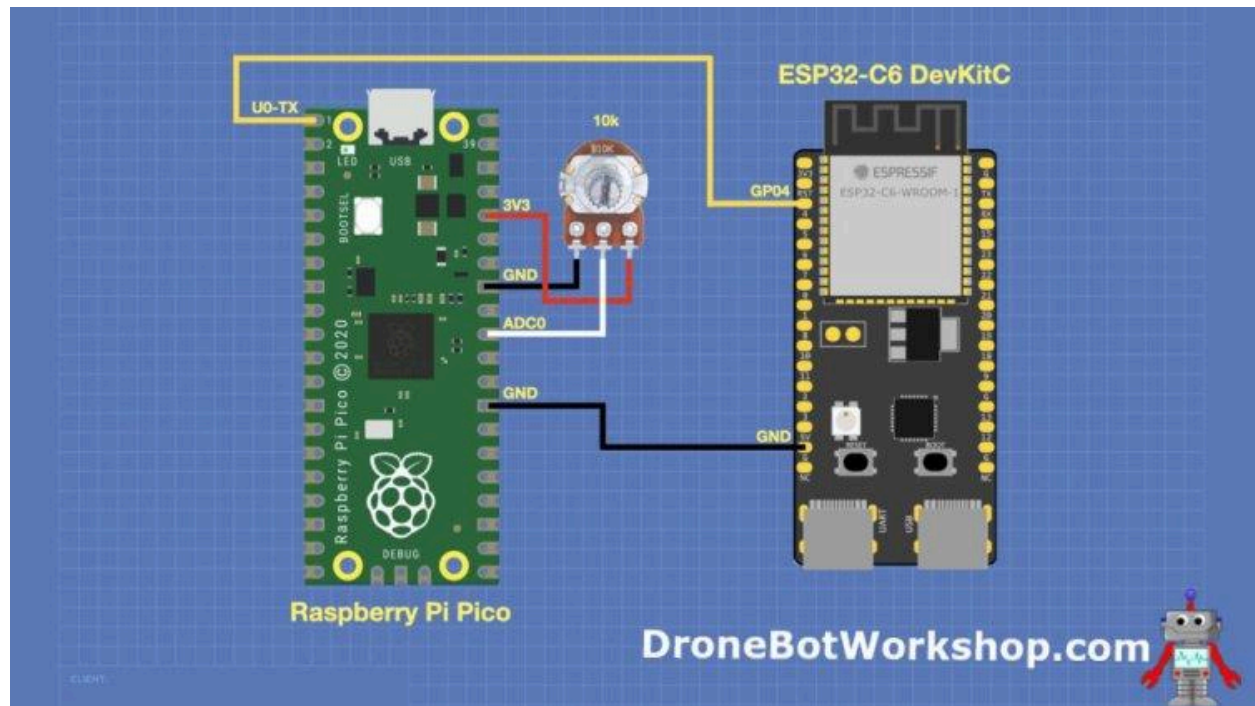
We need three connections between the Raspberry Pi Pico and the ESP32:

Potentiometer (on the Pico)

- **One outer pin of potentiometer** → Pico GND
- **Wiper (center pin) of potentiometer** → Pico ADC0 (GP26)
- **Other outer pin of potentiometer** → Pico 3.3V

UART Connection (Pico to ESP32)

- Pico GP0 (UART0 TX) → ESP32 GPIO4 (UART RX)
- Pico GND → ESP32 GND – this common ground connection is essential



The Pico Sketch

The Raspberry Pi Pico runs a simple sketch that reads the potentiometer and sends the value over UART0 once per second. The code is available on the DroneBot Workshop website; below is a summary of what it does:

- Defines A0 (GP26) as the potentiometer pin and sets the ADC resolution to 12 bits (0 – 4095).
- Initializes USB serial at 115200 baud for debugging in the Arduino Serial Monitor.
- Initializes Serial1 (UART0, TX on GP0) at 9600 baud – this is the link to WireClaw.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

- In the main loop: reads the potentiometer, prints the value to both serial interfaces, and waits one second before repeating.

```
/*  
  Pico Sender for WireClaw  
  pico-wireclaw-sender.ino  
  Sends potentiometer position using UART0  
  Uses Raspberry Pi Pico (out GP0)  
  Used with ESP32 running WireClaw (in GPIO4)  
  
  DroneBot Workshop 2026  
  https://dronebotworkshop.com  
*/  
  
const int POT_PIN = A0; // Pico A0 is GP26  
  
void setup() {  
  analogReadResolution(12); // 0 to 4095  
  
  Serial.begin(115200); // USB serial monitor  
  Serial1.begin(9600); // UART0: TX on GP0, RX on GP1  
  
  delay(1000);  
  Serial.println("WireClaw Serial Bridge Sender Started");  
}  
  
void loop() {  
  int value = analogRead(POT_PIN);
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
Serial.println(value);    // Debug over USB

Serial1.println(value);   // Sent to WireClaw

delay(1000);

}
```

Flash this sketch to the Pico using the Arduino IDE with the Raspberry Pi Pico board support package installed. Once uploaded, the Pico will start sending potentiometer readings to WireClaw at 9600 baud, once per second.

Registering the Connection in WireClaw

With both boards powered and the wiring in place, open Telegram and tell WireClaw about the new connection:

```
Connect an Arduino on serial at 9600 baud and register it as a sensor called Arduino_L
```

WireClaw will confirm that the serial bridge has been registered. Navigate to the Devices page, and you will see **Arduino_L** listed, along with the current value received from the Pico. Turn the potentiometer on the Pico – the value in the device list will change.

You can now read the sensor value directly through Telegram:

```
Read Arduino_L
```

WireClaw will reply with the current potentiometer reading from the Pico, just as it would for any other registered sensor.

You can also create rules based on this value – for example, triggering a Telegram alert when the Pico's potentiometer is turned up high – exactly as you would for any other WireClaw sensor.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

WireClaw also supports a more sophisticated serial protocol: if your external microcontroller sends JSON-formatted name/value pairs (for example, {"temperature": 23.5, "humidity": 61}), WireClaw can parse and register each field as a separate sensor. This makes it straightforward to bridge a complex sensor node into WireClaw with a single UART connection.

Conclusion

WireClaw is a unique product that changes how we interact with microcontroller hardware. By embedding an AI agent directly on the ESP32, it eliminates the traditional edit-compile-flash-debug cycle for a wide class of hardware tasks – you simply describe what you want, and it happens.

The possible applications for this technology are broad. Home automation and environmental monitoring are obvious starting points – imagine describing your whole-house temperature and lighting logic in plain English and having it just work, with no YAML files or automation scripting. WireClaw is equally suited to industrial monitoring applications, where non-technical operators need to configure alert thresholds without developer support. The serial bridge feature opens the door to building multi-board sensor networks, where a low-power slave board handles physical sensing while the ESP32 running WireClaw provides connectivity and intelligence.

Worth keeping an eye on is ESPClaw, a similar project from Espressif themselves. It currently supports fewer ESP32 models than WireClaw, but the fact that the chip manufacturer is exploring this space suggests that AI-on-chip is a direction the industry is taking seriously.