Internet Of Things

ESP32 WebSocket – Complete Manual

A practical guide for sending and receiving real-time data between ESP32 and a Web Browser

---

1. Introduction

What is a WebSocket?

A **WebSocket** is a communication protocol that creates a **persistent, bidirectional** connection between a client (browser, PC, mobile) and a server (ESP32).

Unlike HTTP, which works as:

- Browser sends request
- ESP32 sends response
- Connection closes

WebSockets keep the connection **open at all times**, allowing:

- ESP32 → Browser (push messages in real time)
- Browser → ESP32 (send commands instantly)

This allows continuous, low-latency communication.

Why WebSockets?

Traditional HTTP is slow and inefficient for real-time applications because:

- You must repeatedly refresh or poll for updates.
- There is increased latency and network traffic.

WebSockets solve this:

- No repeated HTTP requests
- Real-time updates in milliseconds
- ESP32 can push data to the browser automatically

Is WebSocket an IoT protocol?

WebSocket is **not an IoT protocol by design**.
It is a **web communication protocol**.

However, it is widely used in IoT because:

- It works directly with **web browsers without any extra software**
- Supports **bi-directional, real-time** communication
- Very easy to use for dashboards, controls, and live monitoring

In IoT, WebSocket complements protocols like:

- MQTT (for cloud messaging)
- HTTP (for configuration)
- CoAP (for constrained devices)

For **ESP32 + Browser**, WebSocket is the *ideal choice*.

Where is WebSocket used?

- Real-time sensor dashboards
- Remote control systems
- IoT monitoring
- Robotics control and telemetry
- Chat applications
- Streaming live data like GPS, temperature, motor RPM, ADC values
- Smart home dashboards

---

2. Required Libraries

For ESP32 WebSockets, you need:

Core Libraries

(a) WiFi.h

Used for:

- Connecting ESP32 to WiFi
- Getting IP address
- Managing network state

It comes pre-installed with the ESP32 board package.

(b) WebServer.h

Used for:

- Hosting a webpage
- Serving HTML directly from ESP32 memory

This is optional but required if you want to view a browser UI.

Comes built-in with ESP32 package.

(c) WebSocketsServer.h

This is the main library, used for:

- Creating WebSocket server on ESP32
- Handling clients
- Sending/receiving data
- Managing WebSocket events

---

3. Installation steps (Arduino IDE)

1. Open Arduino IDE → **Tools → Board → Boards Manager** → search "esp32" → install **esp32 by Espressif Systems**.
2. **Sketch → Include Library → Manage Libraries**:
    - Search and install **WebSockets** (Markus Sattler).

This installs:

- WiFi library
- WebServer library
- All necessary dependencies
- ESP32 board definitions

---

4. Important WebSocket Functions (Most Used)

These are the only functions you need for 90% of projects.

4.1 Start WebSocket Server

```
webSocket.begin();
```

4.2 Attach Event Handler

```
webSocket.onEvent(webSocketEvent);
```

4.3 Inside loop, process WebSocket events

```
webSocket.loop();
```

4.4 Send message to all connected clients

```
webSocket.broadcastTXT("message");
```

4.5 Send message to a specific client

```
webSocket.sendTXT(clientID, "message");
```

4.6 Receive messages in the callback

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length)
```

---

5. Understanding the WebSocket Workflow

Step 1: ESP32 connects to WiFi

This makes ESP32 accessible through a network IP.

Step 2: ESP32 starts a WebServer

Serves the HTML user interface.

Step 3: ESP32 starts a WebSocket server

Listens for real-time communication.

Step 4: Browser loads the webpage

The HTML page contains JavaScript that opens a WebSocket connection.

Step 5: Connection established

Browser and ESP32 start exchanging messages instantly.

---

6. Full Step-by-Step Implementation

This example creates:

- A WebSocket server
- A webpage with ON/OFF buttons
- Real-time communication
- Sending data every 2 seconds
- No SPIFFS, no external storage

Complete ESP32 Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <WebSocketsServer.h>

const int ledPin = 2;

const char* ssid = "vivo 1610";
const char* password = "123456789";

WebServer server(80);
WebSocketsServer webSocket(81);

void webSocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t
length) {
  switch (type) {
    case WStype_CONNECTED:
      Serial.printf("\nClient %u connected", num);
      break;

    case WStype_DISCONNECTED:
      Serial.printf("\nClient %u disconnected", num);
      break;

    case WStype_TEXT: {
      String cmd = (const char*)payload;
```

```
      if (cmd == "ON")  digitalWrite(ledPin, HIGH);
      if (cmd == "OFF") digitalWrite(ledPin, LOW);
      break;
    }
  }
}

const char index_html[] PROGMEM = R"HTML(
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ESP32 WebSocket</title>
  </head>
  <body>
    <h2>LED Control</h2>
    <button onclick="send('ON')">ON</button>
    <button onclick="send('OFF')">OFF</button>

    <h3>Counter: <span id="counter">--</span></h3>

    <script>
      var ws = new WebSocket("ws://" + location.hostname + ":81/");
      function send(v){
        ws.send(v);
        }
      ws.onmessage = function(e){ document.getElementById("counter").innerText
= e.data; }
    </script>
  </body>
</html>
)HTML";

void html_page(){
  server.send(200, "text/html", index_html);
}

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

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }
  Serial.println("\nConnected: " + WiFi.localIP().toString());

  server.on("/", html_page);
  server.begin();

  webSocket.begin();
  webSocket.onEvent(webSocketEvent);
}

unsigned long lastTime = 0;
int count = 0;

void loop() {
  server.handleClient();
  webSocket.loop();

  if (millis() - lastTime > 2000) {
    lastTime = millis();
    count++;
    String msg = String(count);
    webSocket.broadcastTXT(msg);
  }
}
```

---

7. Testing the System

1. Upload code
2. Open Serial Monitor
3. Note the ESP32 IP address
4. On your PC/mobile browser, enter:

http://<ESP32-IP>/

5. The webpage loads
6. WebSocket automatically connects
7. Press ON/OFF buttons
8. Watch ESP32 LED respond instantly
9. Every 2 seconds ESP32 sends a message back

---

8. Key Points to Remember

- WebSocket connection remains open
- You do not refresh the browser
- ESP32 can push data anytime
- Browser JavaScript receives messages instantly
- Very fast, ideal for sensor dashboards
