

How to use ROS on Arduino Nano

Objective [↗](#)

Adapt the `Arduino Nano` (publisher/subscriber) to receive messages using `ROS`.

What to do [↗](#)

- Connect the `Arduino Nano` as a `ROS` publisher/subscriber using `roserial`.
- Receive messages from `ROS` (for example: {time_ON, delay} or just a simple value).

Requirements [↗](#)

In Arduino IDE:

- Board: `Arduino AVR Boards (Arduino Nano)`
- Libraries:
 - `ros.h` (comes with `roserial_arduino`)

Configuration [↗](#)

- **Board:** `Arduino Nano`
- **Processor:** `ATmega328P`
- **Port:** The `COM` port of the `Arduino Nano`

⚠ The old `Bootloader` is required if the `Arduino Nano` is not original (clone).

In ROS (Ubuntu 24.04.6 LTS executed from Windows WSL) [↗](#)

⚠ The setup was tested using Ubuntu 24.10 LTS and it did not work.

First, `Ubuntu` is required because of the libraries dependencies. For this, there are several options: using a virtual machine, installing the OS on the PC, or using `WSL`.

`WSL` was the chosen option. To install it, follow these steps:

1. Open `PowerShell` as Administrator and run:

```
1 wsl --install
```

2. If `Ubuntu` doesn't install automatically, install it manually from the `Microsoft Store`:

2.1 Open the `Microsoft Store`.

2.2 Search for " `Ubuntu` " (`Ubuntu 20.04 LTS` is the recommended version, see warning above).

2.3 Click on "Install".

2.4 Once the installation is complete, open `Ubuntu` from the `Windows Start menu` and follow the setup process (create a user and password).

Install and set up ROS Noetic on Windows 10/11 using WSL (Windows Subsystem for Linux) and prepare the environment for development with Arduino. [🔗](#)

Step-by-step: install ROS Noetic on Ubuntu WSL [🔗](#)

All these commands are run inside the `Ubuntu` terminal in `WSL`.

1. Update system packages: [🔗](#)

```
1 sudo apt update
2 sudo apt upgrade -y
```

2. Install basic dependencies: [🔗](#)

```
1 sudo apt install curl gnupg2 lsb-release -y
```

3. Add the ROS repository key: [🔗](#)

```
1 curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

4. Add the ROS Noetic repository: [🔗](#)

```
1 sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-l
```

5. Update and install ROS Noetic full desktop: [🔗](#)

```
1 sudo apt update
2 sudo apt install ros-noetic-desktop-full -y
```

6. Configure the ROS environment (so it loads in each terminal): [🔗](#)

```
1 echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

7. Install additional ROS development tools: [🔗](#)

```
1 sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential -y
```

8. Initialize rosdep: [🔗](#)

```
1 sudo rosdep init
2 rosdep update
```

9. Install roserial (communication with Arduino): [🔗](#)

```
1 sudo apt install ros-noetic-roserial ros-noetic-roserial-arduino ros-noetic-roserial-python -y
```

Verify [🔗](#)

Run the following command to start the ROS master service:

```
1 roscore
```

If everything is correct, it should start without errors. In case of an error indicating that `roslaunch` is missing, run:

```
1 sudo apt install ros-noetic-roslaunch -y
```

Now in Arduino IDE [↗](#)

Installing the Library in Arduino IDE

- Open the `Arduino` IDE.
- Go to the menu: **Sketch** → **Include Library** → **Manage Libraries...**
- In the search bar, type "**rosserial**".
- Select the "**Rosserial Arduino Library**", choose version **0.7.8**, and click **Install**.

Rosserial Arduino Library by Michael...
0.7.8 installed
Use an Arduino as a ROS publisher/subscriber Works with <http://wiki.ros.org/rosserial,...>
[More info](#)
0.7.8 ▼ **REMOVE**

Connect USB to WSL [↗](#)

Install `usbipd`

Once `usbipd` is installed, follow these steps:

1. Run the following command to see the list of connected USB devices and note down the bus ID of the USB device to be attached to WSL: [↗](#)

```
1 usbipd list
2
3 Connected:
4 BUSID  VID:PID  DEVICE                                STATE
5 1-2    1a86:7523  USB-SERIAL CH340 (COM3)              Not shared
6 2-1    04f3:0c00  ELAN WBF Fingerprint Sensor         Not shared
7 2-2    0bda:b00c  Realtek Bluetooth 5 Adapter         Not shared
8 2-4    04f2:b75e  HP TrueVision HD Camera, Camera DFU Device Not shared
9
10 Persisted:
11 GUID                                DEVICE
```

2. Use this command to share the USB device, replacing `<busid>` with the bus ID we previously noted: [↗](#)

```
1 usbipd bind --busid <busid>
```

3. Verify that the device is shared: [↗](#)

```
1 usbipd list
2
3 Connected:
4 BUSID  VID:PID  DEVICE                                STATE
5 1-2    1a86:7523  USB-SERIAL CH340 (COM3)              Shared
6 2-1    04f3:0c00  ELAN WBF Fingerprint Sensor         Not shared
7 2-2    0bda:b00c  Realtek Bluetooth 5 Adapter         Not shared
8 2-4    04f2:b75e  HP TrueVision HD Camera, Camera DFU Device Not shared
```

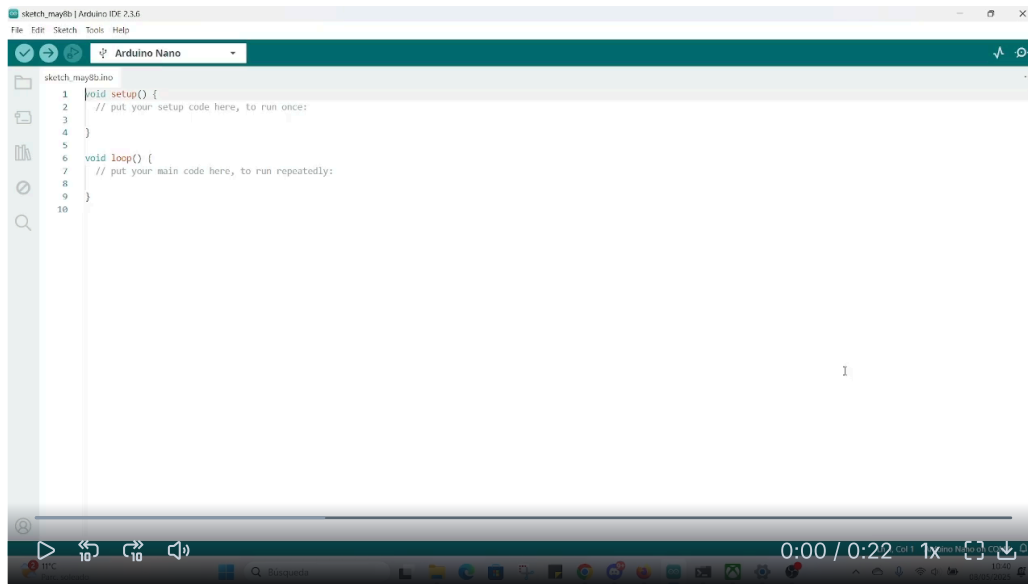
4. Run the following command: [↗](#)

```
1 usbipd attach --wsl --busid 1-2
2
3 usbipd: info: Using WSL distribution 'Ubuntu-20.04' to attach; the device will be available in all WSL 2 distribu
4 usbipd: info: Detected networking mode 'nat'.
5 usbipd: info: Using IP address 172.27.144.1 to reach the host.
```

⚠ Make sure to have a WSL window running while you execute the command

“Hello world!” example [🔗](#)

How to import the code: [🔗](#)



Code: [🔗](#)

```
1  /*
2   * roserial Publisher Example
3   * Prints "hello world!"
4   */
5
6  #include <ros.h>
7  #include <std_msgs/String.h>
8
9  ros::NodeHandle  nh;
10
11  std_msgs::String str_msg;
12  ros::Publisher chatter("chatter", &str_msg);
13
14  char hello[13] = "hello world!";
15
16  void setup()
17  {
18    nh.initNode();
19    nh.advertise(chatter);
20  }
21
22  void loop()
23  {
24    str_msg.data = hello;
25    chatter.publish( &str_msg );
26    nh.spinOnce();
27    delay(1000);
28  }
29
```

Code explanation: [↗](#)

Include libraries [↗](#)

```
1 #include <ros.h>
2 #include <std_msgs/String.h>
```

- `ros.h`: the main ROS communication library for Arduino. It allows communication with a ROS master.
- `std_msgs/String.h`: this is the standard message type in ROS for sending text strings.

Global declarations [↗](#)

```
1 ros::NodeHandle nh;
```

- `nh` is an instance of `NodeHandle`, which manages communication with the ROS system (e.g., initialization, publishing, subscribing).

```
1 std_msgs::String str_msg;
2 ros::Publisher chatter("chatter", &str_msg);
```

- `str_msg` is an object of type `String` from `std_msgs`. It holds the string message to be published.
- `chatter` is a ROS publisher that will publish `str_msg` messages to the ROS topic "chatter".

```
1 char hello[13] = "hello world!";
```

- A C-style character array holding the message "hello world!". The array has space for 13 characters: 12 letters + 1 null terminator (`\0`).

Setup function [↗](#)

```
1 void setup() {
2   nh.initNode();
3   nh.advertise(chatter); }
```

- `nh.initNode()`: initializes the ROS node, establishing communication with the ROS master.
- `nh.advertise(chatter)`: tells ROS that this node will be publishing messages to the "chatter" topic.

Loop function [↗](#)

```
1 void loop() {   str_msg.data = hello;
2   chatter.publish( &str_msg );
3   nh.spinOnce();
4   delay(1000); }
```

This is the main loop that repeatedly runs:

1. `str_msg.data = hello;`
 - Assigns the string "hello world!" to the message's `data` field.
2. `chatter.publish(&str_msg);`
 - Publishes the message on the "chatter" topic.
3. `nh.spinOnce();`
 - Handles any ROS communication, such as publishing or receiving messages.
4. `delay(1000);`
 - Waits for 1 second before repeating the loop. This means a new message is sent every second.

Launching ROS communication with Arduino [↗](#)

Here, run the following command to verify that the device is correctly connected:

```
1 lsusb
2 Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
3 Bus 001 Device 002: ID 1a86:7523 QinHeng Electronics HL-340 USB-Serial adapter
4 Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Now that the device is connected to `WSL`, run the code previously uploaded to `Arduino Nano`. To do this, open three terminal windows:

- In the first window, which will act as the `ROS` master server, run the following command:

```
1 roscore
2
3 ... logging to /home/andrei/.ros/log/3e968780-3173-11f0-8d2f-5fa14544ce13/roslaunch-DESKTOP-R1R7VHK-1076.log
4 Checking log directory for disk usage. This may take a while.
5 Press Ctrl-C to interrupt
6 Done checking log file disk usage. Usage is <16B.
7
8 started roslaunch server http://DESKTOP-R1R7VHK:46611/
9 ros_comm version 1.17.0
10
11
12 SUMMARY
13 =====
14
15 PARAMETERS
16 * /rostdistro: noetic
17 * /rosversion: 1.17.0
18
19 NODES
20
21 auto-starting new master
22 process[master]: started with pid [1086]
23 ROS_MASTER_URI=http://DESKTOP-R1R7VHK:11311/
24
25 setting /run_id to 3e968780-3173-11f0-8d2f-5fa14544ce13
26 process[rosout-1]: started with pid [1096]
27 started core service [/rosout]
```

This command starts the `ROS` core, which is necessary for `ROS` nodes to communicate with each other.

- In the second window, which will act as the publisher, run the following command:

```
1 rosrun roserial_python serial_node.py /dev/ttyUSB0
2
3 [INFO] [1747303310.966436]: ROS Serial Python Node
4 [INFO] [1747303310.972126]: Connecting to /dev/ttyUSB0 at 57600 baud
5 [INFO] [1747303313.082193]: Requesting topics...
6 [INFO] [1747303318.173192]: Note: publish buffer size is 280 bytes
7 [INFO] [1747303318.174942]: Setup publisher on led_control [std_msgs/Int32MultiArray]
8 [INFO] [1747303318.190249]: Note: subscribe buffer size is 280 bytes
9 [INFO] [1747303318.193192]: Setup subscriber on led_control [std_msgs/Int32MultiArray]
```

This command establishes the connection between the `Arduino` and `ROS` through the serial port and allows the `Arduino` to publish messages to `ROS`. Make sure to use the correct port, which is verifiable with the following command:

```
1 ls /dev/ttyUSB*
2
3 /dev/ttyUSB0
```

This command will list all the available ports and help you identify the one corresponding to the targeted `Arduino`.

- In the third window, which will act as the subscriber, run the following command:

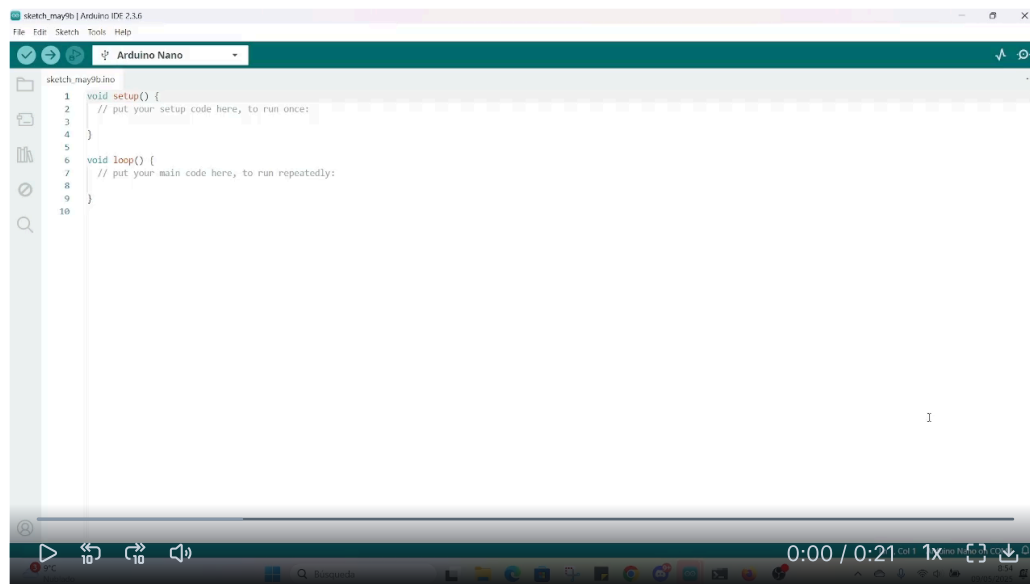
```
1 rostopic echo chatter
```

This command subscribes to the `chatter` topic and allows to view the messages that the `Arduino` is publishing to `ROS` through the `roserial` node.

```
1 data: "hello world!"
2 ---
3 data: "hello world!"
4 ---
5 data: "hello world!"
6 ---
7 data: "hello world!"
8 ---
9 data: "hello world!"
10 ---
11 data: "hello world!"
12 ---
```

ROSSerial LED Toggle Example [↗](#)

How to import the code: [↗](#)



Code: [↗](#)

```
1 /*
2  * roserial Subscriber Example
3  * Blinks an LED on callback
4  */
5
6 #include <ros.h>
7 #include <std_msgs/Empty.h>
```

```

8
9  ros::NodeHandle nh;
10
11 void messageCb( const std_msgs::Empty& toggle_msg){
12     digitalWrite(13, HIGH-digitalRead(13));  // blink the led
13 }
14
15 ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );
16
17 void setup()
18 {
19     pinMode(13, OUTPUT);
20     nh.initNode();
21     nh.subscribe(sub);
22 }
23
24 void loop()
25 {
26     nh.spinOnce();
27     delay(1);
28 }

```

Code Explanation [↗](#)

This `Arduino` sketch demonstrates how to use `ROSserial` to control an onboard `LED` via `ROS` topics.

Overview: [↗](#)

The sketch subscribes to a `ROS` topic named `"toggle_led"`. Every time a message is received on this topic, it toggles the state of the built-in `LED` (pin 13).

Key Components: [↗](#)

- **`ros::NodeHandle nh`**

Initializes the `ROSserial` communication between `Arduino` and a `ROS` master.

- **`void messageCb(const std_msgs::Empty& toggle_msg)`**

This is the callback function triggered when a message is received on the `"toggle_led"` topic. It toggles the `LED` by inverting its current state using:

```
1 digitalWrite(13, HIGH - digitalRead(13));
```

- **`ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb);`**

Creates a subscriber to the `"toggle_led"` topic. It listens for messages

of type `std_msgs::Empty`, which don't carry any data—just a trigger signal.

`setup()` function: [↗](#)

- Sets pin 13 as an output.
- Initializes the `ROS` node.
- Registers the subscriber.

`loop()` function: [↗](#)

- Calls `nh.spinOnce()` to check for incoming messages.
- Adds a small delay to avoid overwhelming the serial buffer.

Launching ROS Communication with Arduino [↗](#)

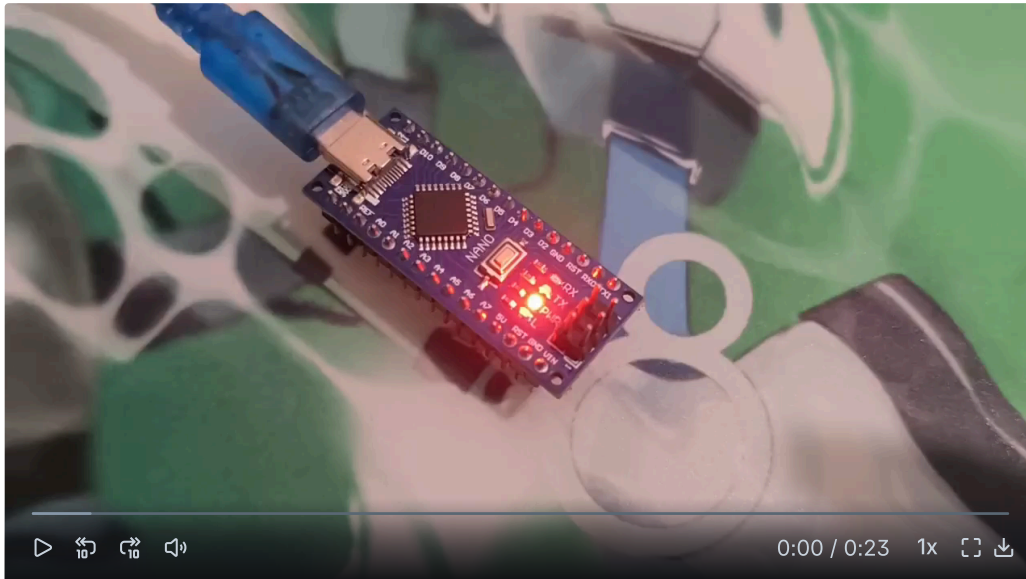
[-] The process for this example is exactly the same as the previous one except for the command you will need to execute in the terminal that acts as the subscriber.

- In this case you need to execute this command:

```
1 rostopic pub toggle_led std_msgs/Empty --once
```

This command publishes a single empty message to the `toggle_led` topic. It is used to send a one-time trigger to the `Arduino` through the `rosserial` node, typically to toggle the state of a `LED` (e.g., turning it on or off). The `std_msgs/Empty` message type signifies that no data is being sent—just the event itself.

```
1 publishing and latching message for 3.0 seconds
```



Code: [🔗](#)


```
1 #include <ros.h>
2 #include <std_msgs/Int32MultiArray.h>
3
4 // Create an instance of the ROS node handle
5 ros::NodeHandle nh;
6
7 // Pin for the red LED
8 const int RED_LED_PIN = 13; // Red LED pin, set to pin 13 on the Arduino
9
10 // Callback function declaration (it must be declared before it is used)
11 void messageCallback(const std_msgs::Int32MultiArray& msg) {
12     // Extract values from the received message
13     int greenLedTime = msg.data[0]; // The first value controls the duration the red LED stays on
14     int waitTime = msg.data[1];     // The second value controls how long to wait before the next action
15
16     // Turn the red LED on
17     digitalWrite(RED_LED_PIN, HIGH);
18
19     // Wait for the amount of time specified by greenLedTime (multiplied by 1000 to convert to milliseconds)
20     delay(greenLedTime * 1000); // Multiply by 1000 to convert from seconds to milliseconds
21
22     // Turn the red LED off
23     digitalWrite(RED_LED_PIN, LOW);
24 }
```

```

25 // Wait for the amount of time specified by waitTime (multiplied by 1000 to convert to milliseconds)
26 delay(waitTime * 1000);
27 }
28
29 // Subscriber (after declaring the callback function)
30 ros::Subscriber<std_msgs::Int32MultiArray> sub("led_control", &messageCallback); // Subscribe to the "led_contr
31
32 // Publisher
33 std_msgs::Int32MultiArray msg; // Create an Int32MultiArray message
34 ros::Publisher pub("led_control", &msg); // Publish to the "led_control" topic
35
36 // Initialize an array to store the data for the message
37 int32_t data[2] = {2, 1}; // {red LED on time (2 seconds), wait time (1 second)}
38
39 void setup() {
40     pinMode(RED_LED_PIN, OUTPUT); // Set the red LED pin as an output
41
42     // Initialize ROS node
43     nh.initNode();
44
45     // Advertise the publisher and subscribe to the subscriber
46     nh.advertise(pub); // Advertise the publisher to the ROS network
47     nh.subscribe(sub); // Subscribe to the topic to listen for messages
48
49     // Initialize the message with the initial data values
50     msg.data = data;
51     msg.data_length = 2; // Set the length of the message data (2 values)
52 }
53
54 void loop() {
55     // Update message with current data values
56     msg.data[0] = data[0]; // Set the first element (time the LED stays on)
57     msg.data[1] = data[1]; // Set the second element (wait time before next action)
58
59     // Publish the message to the ROS network
60     pub.publish(&msg);
61
62     // Quick blink of the red LED to indicate activity
63     digitalWrite(RED_LED_PIN, HIGH); // Turn the red LED on
64     delay(300); // Wait for 300 milliseconds (LED stays on)
65     digitalWrite(RED_LED_PIN, LOW); // Turn the red LED off
66     delay(300); // Wait for 300 milliseconds (LED stays off)
67
68     // Wait for 5 seconds before repeating
69     delay(5000); // Wait for 5 seconds before sending the next message
70
71     // Handle any incoming messages or requests
72     nh.spinOnce(); // Process any incoming messages or ROS commands
73 }
74

```

Launching ROS Communication with Arduino [↗](#)

-  The process for this example is exactly the same as the previous ones except for the output on the terminal that acts as publisher. Command is required to be executed on another terminal acting as the subscriber.

Publisher output:

```

1 rosrunc rosserial_python serial_node.py /dev/ttyUSB0
2
3 [INFO] [1747303310.966436]: ROS Serial Python Node
4 [INFO] [1747303310.972126]: Connecting to /dev/ttyUSB0 at 57600 baud
5 [INFO] [1747303313.082193]: Requesting topics...
6 [INFO] [1747303318.173192]: Note: publish buffer size is 280 bytes
7 [INFO] [1747303318.174942]: Setup publisher on led_control [std_msgs/Int32MultiArray]
8 [INFO] [1747303318.190249]: Note: subscribe buffer size is 280 bytes
9 [INFO] [1747303318.193192]: Setup subscriber on led_control [std_msgs/Int32MultiArray]

```

In the subscriber terminal execute the following command:

```

1 rostopic echo /led_control

```

This command displays the messages published to the `/led_control` topic in real-time, allowing to verify communication with the Arduino.

```

1 layout:
2   dim: []
3   data_offset: 0
4 data: [2, 1]
5 ---

```

The output shows the data sent to the Arduino from ROS, in this case, the values `[2, 1]`, which represent control parameters for the LED, such as the time_ON (time which the LED is turned on) and wait time (sleeping time before checking again the topic).

