

# Publish topic

Publish topic

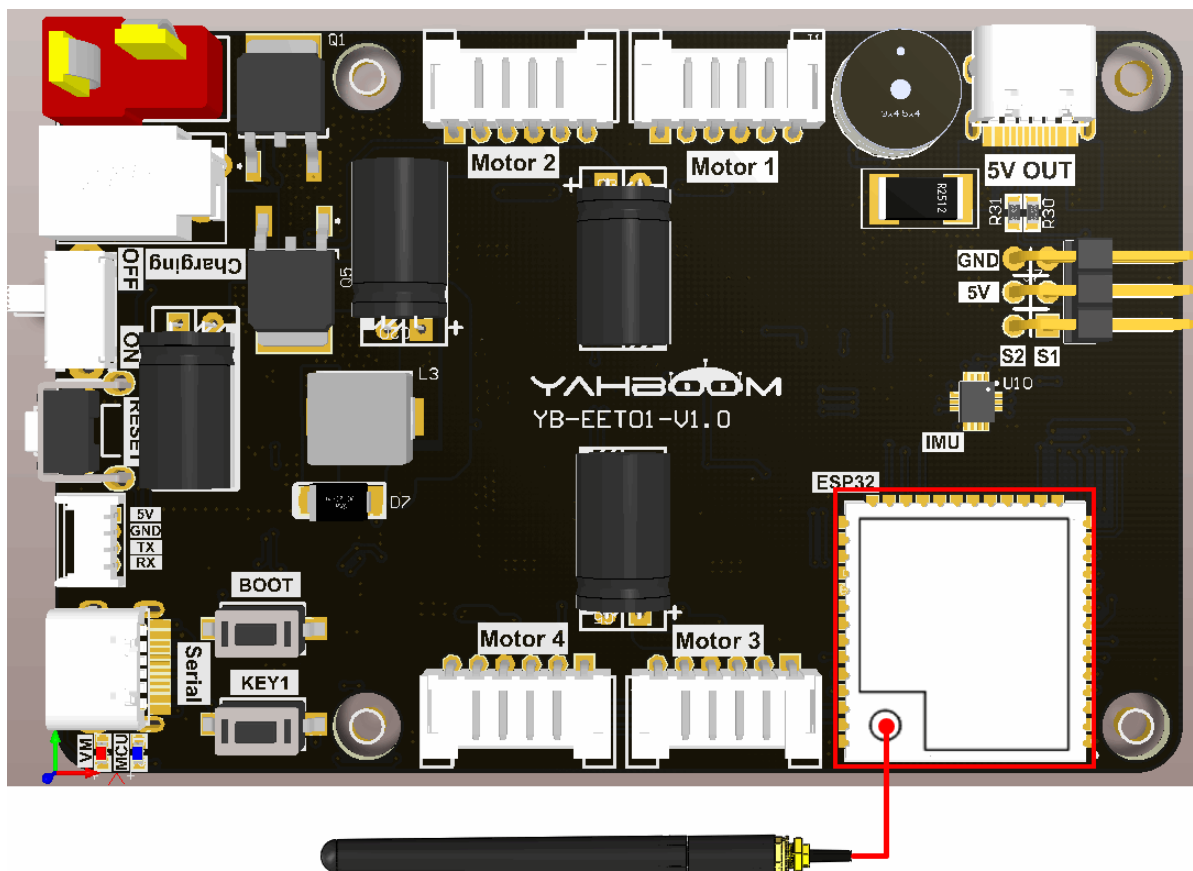
1. Experimental purpose
2. Hardware connection
3. Core code analysis
4. Compile, download and flash firmware
5. Experimental results

## 1. Experimental purpose

Learn ESP32-microROS components, access the ROS2 environment, and publish int32 topics.

## 2. Hardware connection

As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM core module, which has its own wireless WiFi function. The ESP32-S3 core module needs to be connected to an antenna, and a type-C data cable needs to be connected to the computer and the microROS control board as Burn firmware function.



### 3. Core code analysis

The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/microros_samples/publisher
```

First, get the WiFi name and password to connect from the IDF configuration tool.

```
#define ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY  CONFIG_ESP_MAXIMUM_RETRY
```

The `uros_network_interface_initialize` function will connect to WiFi hotspots based on the WiFi configuration in IDF.

```
ESP_ERROR_CHECK(uros_network_interface_initialize());
```

Then obtain `ROS_NAMESPACE`, `ROS_DOMAIN_ID`, `ROS_AGENT_IP` and `ROS_AGENT_PORT` from the IDF configuration tool.

```
#define ROS_NAMESPACE      CONFIG_MICRO_ROS_NAMESPACE
#define ROS_DOMAIN_ID      CONFIG_MICRO_ROS_DOMAIN_ID
#define ROS_AGENT_IP        CONFIG_MICRO_ROS_AGENT_IP
#define ROS_AGENT_PORT      CONFIG_MICRO_ROS_AGENT_PORT
```

Initialize the configuration of microROS, in which `ROS_DOMAIN_ID`, `ROS_AGENT_IP` and `ROS_AGENT_PORT` are modified in the IDF configuration tool according to actual needs.

```
rcl_allocator_t allocator = rcl_get_default_allocator();
rclc_support_t support;

// 创建rcl初始化选项
// Create init_options.
rcl_init_options_t init_options = rcl_get_zero_initialized_init_options();
RCCHECK(rcl_init_options_init(&init_options, allocator));
// 修改ROS域ID
// change ros domain id
RCCHECK(rcl_init_options_set_domain_id(&init_options, ROS_DOMAIN_ID));

// 初始化rmw选项
// Initialize the rmw options
rmw_init_options_t *rmw_options =
rcl_init_options_get_rmw_init_options(&init_options);

// 设置静态代理IP和端口
// Setup static agent IP and port
RCCHECK(rmw_uros_options_set_udp_address(ROS_AGENT_IP, ROS_AGENT_PORT,
rmw_options));
```

Try to connect to the proxy. If the connection is successful, go to the next step. If the connection to the proxy is unsuccessful, you will always be in the connected state.

```

while (1)
{
    ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
    state_agent = rcl_support_init_with_options(&support, 0, NULL,
&init_options, &allocator);
    if (state_agent == ESP_OK)
    {
        ESP_LOGI(TAG, "Connected agent: %s:%s", ROS_AGENT_IP,
ROS_AGENT_PORT);
        break;
    }
    vTaskDelay(pdMS_TO_TICKS(500));
}

```

Create the node "esp32\_publisher", in which ROS\_NAMESPACE is empty by default and can be modified in the IDF configuration tool according to actual conditions.

```

rcl_node_t node;
RCCHECK(rcl_node_init_default(&node, "esp32_publisher", ROS_NAMESPACE,
&support));

```

To create the publisher "publisher", you need to specify the publisher information as std\_msgs/msg/Int32 type.

```

RCCHECK(rcl_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "publisher"));

```

Create a timer for the publisher with a publishing frequency of 10HZ.

```

rcl_timer_t timer;
const unsigned int timer_timeout = 100;
RCCHECK(rcl_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

```

Create an executor, where the three parameters are the numbers controlled by the executor, which should be greater than or equal to the number of subscribers and publishers added to the executor. and add the publisher's timer to the executor

```

rcl_executor_t executor;
int handle_num = 1;
RCCHECK(rcl_executor_init(&executor, &support.context, handle_num,
&allocator));

RCCHECK(rcl_executor_add_timer(&executor, &timer));

```

The function of publishing information is executed in the publisher timer callback. In order to facilitate viewing, the value of the current msg.data is printed. After the publishing is completed, msg.data is automatically incremented by 1.

```
void timer_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL)
    {
        printf("Publishing: %d\n", (int)msg.data);
        RCSoftCHECK(rcl_publish(&publisher, &msg, NULL));
        msg.data++;
    }
}
```

Call rclc\_executor\_spin\_some in the loop to make microros work normally.

```
while (1)
{
    rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
    usleep(1000);
}
```

## 4. Compile, download and flash firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

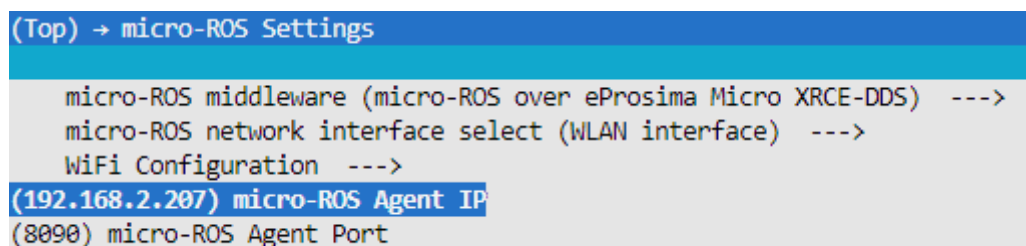
Enter the project directory

```
cd ~/esp/Samples/microros_samples/publisher
```

Open the ESP-IDF configuration tool.

```
idf.py menuconfig
```

Open micro-ROS Settings, fill in the IP address of the agent host in micro-ROS Agent IP, and fill in the port number of the agent host in micro-ROS Agent Port.



The screenshot shows the 'micro-ROS Settings' menu in the ESP-IDF configuration tool. The menu items are: 'micro-ROS middleware (micro-ROS over eProsima Micro XRCE-DDS) --->', 'micro-ROS network interface select (WLAN interface) --->', and 'WiFi Configuration --->'. Below these, the 'micro-ROS Agent IP' is set to '(192.168.2.207)' and the 'micro-ROS Agent Port' is set to '(8090)'. The IP address and port number are highlighted with blue boxes.

Open micro-ROS Settings->WiFi Configuration in sequence, and fill in your own WiFi name and password in the WiFi SSID and WiFi Password fields.

```
(Top) → micro-ROS Settings → WiFi Configuration
(YAHBOOM) WiFi SSID
(12345678) WiFi Password
(5) Maximum retry
```

Open the micro-ROS example-app settings. The Ros domain id of the micro-ROS defaults to 20. If multiple users are using it at the same time in the LAN, the parameters can be modified to avoid conflicts. Ros namespace of the micro-ROS is empty by default and does not need to be modified under normal circumstances. If non-empty characters (within 10 characters) are modified, the namespace parameter will be added before the node and topic.

```
(Top) → micro-ROS example-app settings
(16000) Stack the micro-ROS app (Bytes)
(5) Priority of the micro-ROS app
(20) Ros domain id of the micro-ROS
() Ros namespace of the micro-ROS
```

After modification, press S to save, and then press Q to exit the configuration tool.

Compile, burn, and open the serial port simulator.

```
idf.py build flash monitor
```

If you need to exit the serial port simulator, press **Ctrl+]**.

## 5. Experimental results

After powering on, ESP32 tries to connect to the WiFi hotspot, and then tries to connect to the proxy IP and port.

If the agent is not turned on in the virtual machine/computer terminal, please enter the following command to turn on the agent. If the agent is already started, there is no need to start the agent again.

```
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```

```
root@micro-ros:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privile
ged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1705475406.254095] info      | UDPv4AgentLinux.cpp | init      |
running...                | port: 8090
[1705475406.254622] info      | Root.cpp            | set_verbose_level | 1
ogger setup                | verbose_level: 4
```

After the connection is successful, a node and a publisher are created. The publishing frequency is 10hz and the number is increasing.

```
I (5485) MAIN: Connecting agent: 192.168.2.207:8090
I (5491) main_task: Returned from app_main()
I (5519) MAIN: Connected agent: 192.168.2.207:8090
Publishing: 0
Publishing: 1
Publishing: 2
Publishing: 3
Publishing: 4
Publishing: 5
Publishing: 6
Publishing: 7
Publishing: 8
Publishing: 9
Publishing: 10
```

At this time, you can open another terminal in the virtual machine/computer and view the /esp32\_publisher node.

```
ros2 node list
ros2 node info /esp32_publisher
```

```
~$ ros2 node list
/esp32_publisher
~$ ros2 node info /esp32_publisher
/esp32_publisher
Subscribers:

Publishers:
  /publisher: std_msgs/msg/Int32
Service Servers:

Service Clients:

Action Servers:

Action Clients:
```

Subscribe to/publisher topic data

```
ros2 topic echo /publisher
```

Press Ctrl+C to end the command.

```
~$ ros2 topic echo /publisher
data: 0
---
data: 1
---
data: 2
---
data: 3
---
data: 4
```

The frequency of viewing the /publisher topic is about 10hz which is normal.

```
ros2 topic hz /publisher
```

Press Ctrl+C to end the command

```
root@ubuntu:~$ ros2 topic hz /publisher
average rate: 9.905
min: 0.091s max: 0.109s std dev: 0.00515s window: 11
average rate: 9.980
min: 0.070s max: 0.126s std dev: 0.00973s window: 22
average rate: 9.993
min: 0.070s max: 0.126s std dev: 0.00854s window: 33
average rate: 10.003
min: 0.070s max: 0.126s std dev: 0.00758s window: 44
average rate: 9.916
min: 0.070s max: 0.126s std dev: 0.00822s window: 54
average rate: 9.986
min: 0.070s max: 0.126s std dev: 0.00852s window: 65
```