

Guide to MicroRos Usage

Daniele Genovese

August 9, 2024

1 Prerequisites

1.1 ROS

MicroRos is a version of ROS2 designed to be used on microcontrollers. Its aim is to bring ROS capabilities to systems with a limited processing power, memory and storage. Since MicroRos allows the communication between ROS networks and embedded systems, a basic knowledge about ROS/ROS2 is required. In this document, it is assumed that a ROS2 distro is already installed on the computer.

1.2 Required HW/SW

1.2.1 HW

- Microcontroller (MC) (in this case, an ESP32 MC will be used)
- USB-MicroUSB cable

1.2.2 SW

- Visual Studio Code (VSCode)
- Integrated Development Environment extension for VSCode (in this case, PlatformIO)
- MicroRos agent
- MicroRos application

2 PlatformIO Setup

2.1 VSCode

Open the terminal and write the following line:

```
sudo apt install code --classic
```

Once the installation has ended, open VSCode by writing:

```
code .
```

2.2 PlatformIO extension

When VSCode opened, go to the top left bar and open "Extensions"



Figure 1: Extensions submenu

Search for PlatformIO extension and install it.

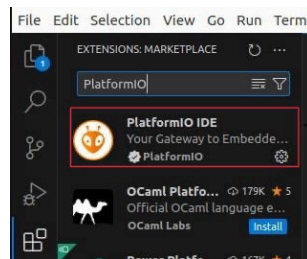


Figure 2: PlatformIO installation

2.3 PlatformIO new project

Once installed, the submenu will appear on the left of the VSCode window:



Figure 3: PlatformIO submenu

Click on it, select *Create New Project* and then, *+ New Project*. Now, name your project, select your board and the framework. After that, deselect the option *Use default location* and choose your ROS2 workspace. This is an example of configuration:

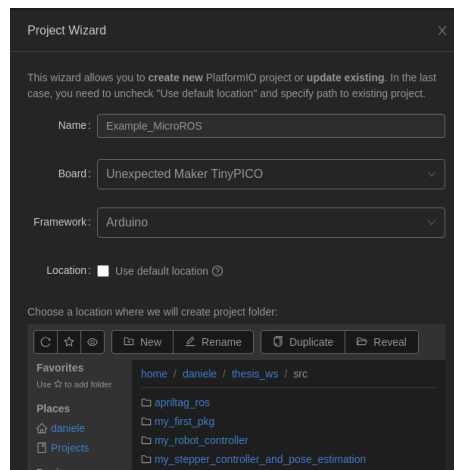


Figure 4: PlatformIO project configuration

The two most important files are `main.cpp` (the application file) and `platformio.ini` (the configuration file). Now, open the main file inside the directory `/path/to/your/workspace/Example_MicroROS/src`. The aim of this example is to run a simple publisher on the MC. To achieve this, we can copy the publisher example from [Here](#) into `main.cpp`.

After that, the file `platformio.ini` must be configured. The field `platform`, `board` and `framework` are configured by default. The library dependencies must be added. In this case, only `micro_ros_platformio` library must be included:

`https://github.com/micro-ROS/micro_ros_platformio`

Moreover, the type of transport must be chosen. In this case, the serial transport is used. The file must look like this:

```
Example_MicroROS > platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:tinypico]
12 platform = espressif32
13 board = tinypico
14 framework = arduino
15 board_microros_transport = serial
16 lib_deps =
17 | https://github.com/micro-ROS/micro\_ros\_platformio
18
```

Figure 5: Platformio.ini file

Pay attention, in this case, ROS2 iron is being using, that is the default distro for MicroRos. In case of using another distro, the following line must be included inside the Platformio.ini file:

`board_microros_distro = < your_distro >`

Then, save the file and wait for the update of metadata.

Once the metadata are updated, the application is ready to be flashed onto the MC. Using the USB-microUSB cable, connect the board to the computer. Now, compile the file using the bottom-left menu.

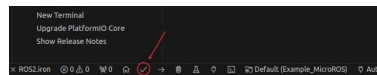


Figure 6: Compilation button

If the compilation went well, the output should look like this:

```
esptool.py v4.5.1
Creating esp32 image...
Merged 2 ELF sections
Successfully created esp32 image.
===== [SUCCESS] Took 6.70 seconds =====
Terminal will be reused by tasks, press any key to close it.
```

Figure 7: Correct compilation output

Finally, upload the application onto the board using the button on the lower left menu.

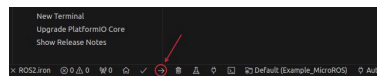


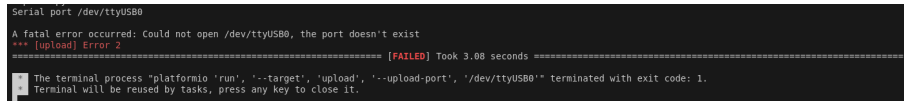
Figure 8: Upload button

The obtained output should look like this:

```
writing at 0x000344eb... (46 %)
writing at 0x000399df... (53 %)
writing at 0x000404a9... (61 %)
writing at 0x00046888... (69 %)
writing at 0x00050db1... (76 %)
writing at 0x00050d6a... (84 %)
writing at 0x0005e52b... (92 %)
writing at 0x00063c8b... (100 %)
Wrote 349488 bytes (280928 compressed) at 0x00010000 in 4.8 seconds (effective 576.8 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 10.56 seconds =====
Terminal will be reused by tasks, press any key to close it.
```

Figure 9: Correct upload output

If this type of error is encountered:



```
Serial port /dev/ttyUSB0
A fatal error occurred: Could not open /dev/ttyUSB0, the port doesn't exist
*** [upload] Error 2
===== [FAILED] Took 3.08 seconds =====
The terminal process "platformio 'run', '--target', 'upload', '--upload-port', '/dev/ttyUSB0'" terminated with exit code: 1.
Terminal will be reused by tasks, press any key to close it.
```

Figure 10: Common error

Type the following line on the terminal to ensure that the user is in the dialout group and try to upload again the application (reboot the system before uploading):

```
sudo usermod -aG dialout $USER
```

3 MicroROS agent

3.1 MicroROS installation

Once the application has been uploaded, the microROS build system must be installed.

- Create a workspace for microROS and download the MicroRos tools

```
source /opt/ros/$ROS_DISTRO/setup.bash
mkdir microros_ws
cd microros_ws
git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git
src/micro_ros_setup
```
- Update the dependencies

```
sudo apt update && rosdep update
rosdep install --from-paths src --ignore-src -y
```
- Install pip

```
sudo apt-get install python3-pip
```
- Build and source MicroROS tools

```
colcon build
source install/local_setup.bash
```

3.2 MicroROS agent run

Now, the microROS agent must be created. To do this, follow the next steps:

- Download the package

```
ros2 run micro_ros_setup create_agent_ws.sh
```

- Build the package

```
ros2 run micro_ros_setup build_agent.sh
```

```
source install/local_setup.bash
```

- Find the serial device name

```
ls /dev/serial/by-id/
```

- Run the agent

```
ros2 run micro_ros_agent micro_ros_agent serial -dev [device]
```

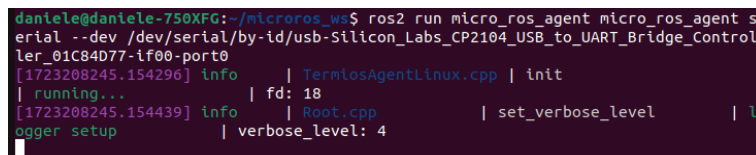
(In this case the serial name is :

```
/dev/serial/by-id/usb-Silicon_Labs_CP2104_USB_to_UART_Bridge_Controller_01C84D77-if00-port0
```

so the command becomes :

```
ros2 run micro_ros_agent micro_ros_agent serial -dev /dev/serial/by-id/usb-Silicon_Labs_CP2104_USB_to_UART_Bridge_Controller_01C84D77-if00-port0)
```

After that, you will obtain something like this:



```
daniele@daniele-750XFG:~/microros_ws$ ros2 run micro_ros_agent micro_ros_agent s
erial --dev /dev/serial/by-id/usb-Silicon_Labs_CP2104_USB_to_UART_Bridge_Control
ler_01C84D77-if00-port0
[1723208245.154296] info | TermiosAgentLinux.cpp | init
| running... | fd: 18
[1723208245.154439] info | Root.cpp | set_verbose_level | 1
ogger setup | verbose_level: 4
```

Figure 11: Agent starting output

At this point, remove and insert USB cable of MC again and the agent has been created:

```
[1723208269.463897] info | TermiosAgentLinux.cpp | init
| running... | fd: 18
[1723208271.664216] info | Root.cpp | create_client | c
reate | client_key: 0x0E86C4A7, session_id: 0x81
[1723208271.664298] info | SessionManager.hpp | establish_session | s
ession established | client_key: 0x0E86C4A7, address: 0
[1723208271.693246] info | ProxyClient.cpp | create_participant | p
articipant created | client_key: 0x0E86C4A7, participant_id: 0x000(1)
[1723208271.709909] info | ProxyClient.cpp | create_topic | t
opic created | client_key: 0x0E86C4A7, topic_id: 0x000(2), participant_
id: 0x000(1)
[1723208271.719583] info | ProxyClient.cpp | create_publisher | p
ublisher created | client_key: 0x0E86C4A7, publisher_id: 0x000(3), particip
ant_id: 0x000(1)
[1723208271.730415] info | ProxyClient.cpp | create_datawriter | d
atawriter created | client_key: 0x0E86C4A7, datawriter_id: 0x000(5), publish
er_id: 0x000(3)
```

Figure 12: Agent final output

And this is what the node is publishing:

```
daniele@daniele-750XFG:~/microros_ws$ ros2 topic list
/micro_ros_platformio_node_publisher
/parameter_events
/rosout
daniele@daniele-750XFG:~/microros_ws$ ros2 topic echo /micro_ros_platformio_node
_publisher
data: 424
---
data: 425
---
data: 426
---
data: 427
---
data: 428
---
data: 429
---
data: 430
---
data: 431
---
data: 432
---
data: 433
---
data: 434
---
```

Figure 13: Application output