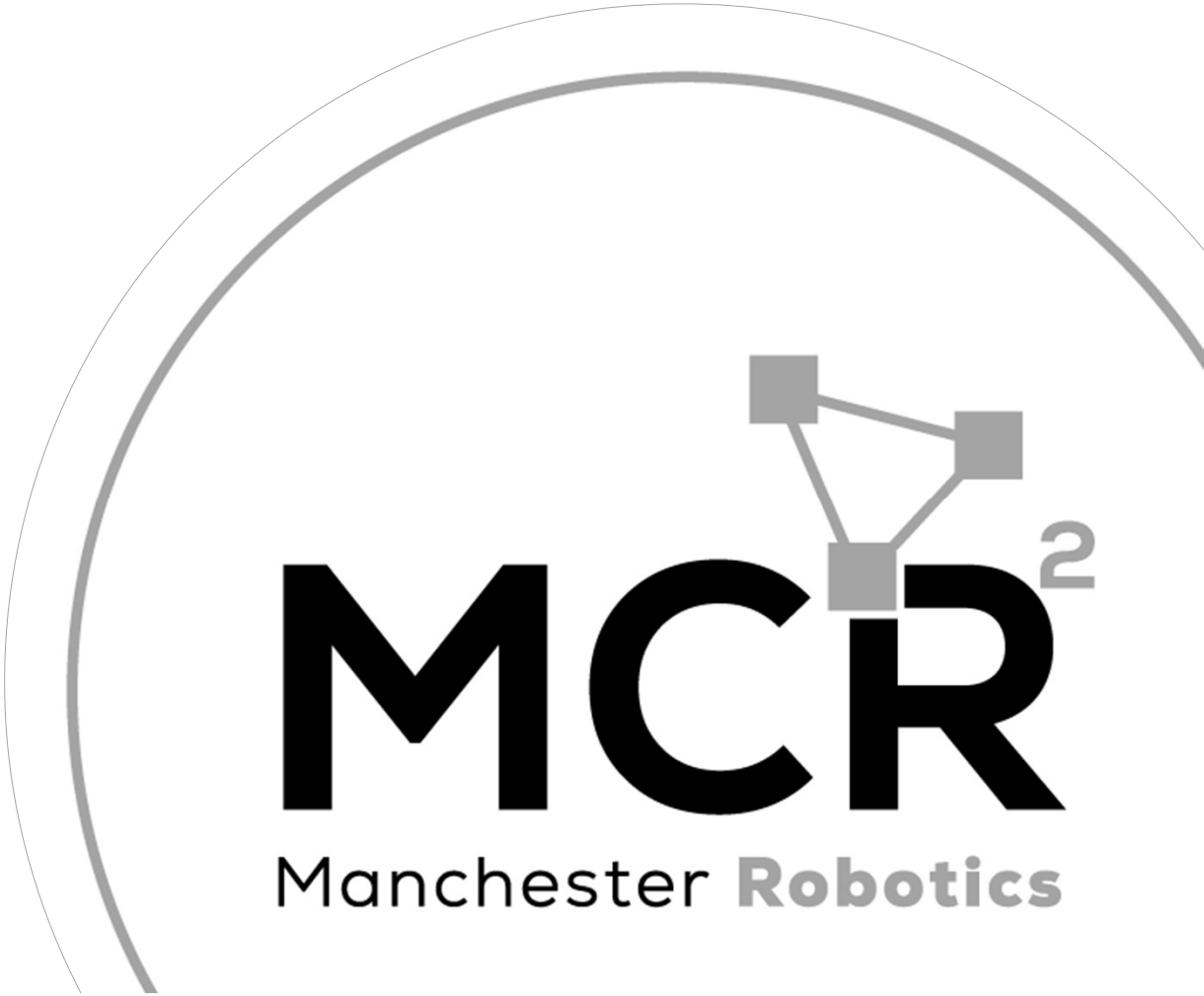
ROS Serial Communication

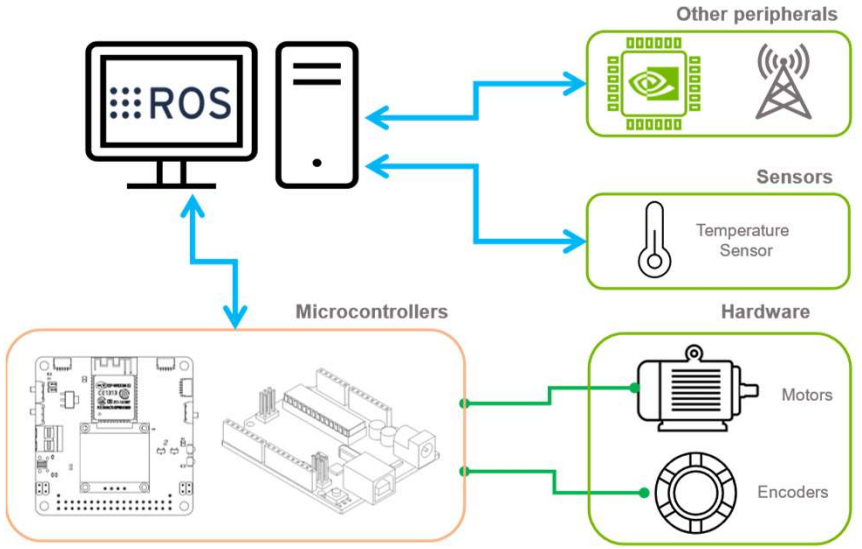
Micro-ROS

{Learn, Create, Innovate};



Micro-ROS

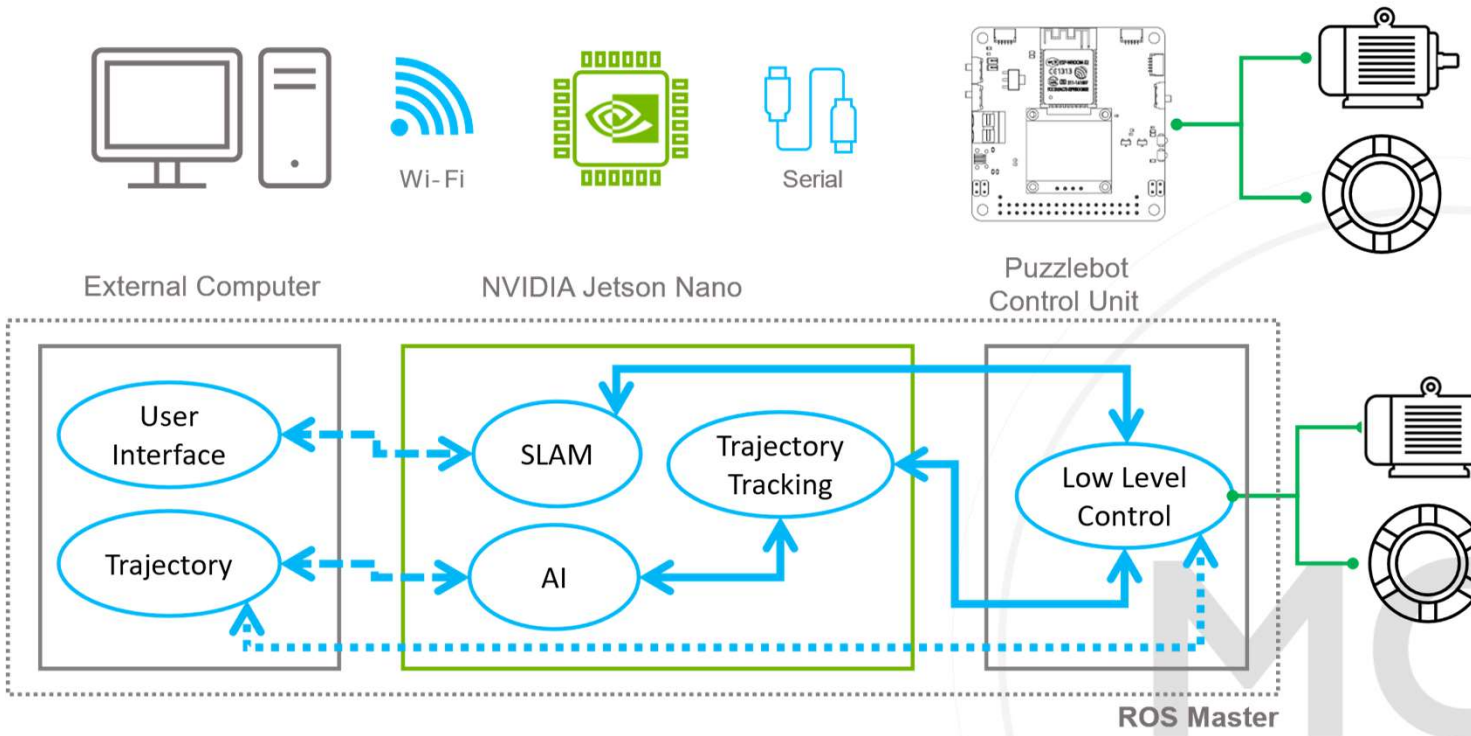
- Bridge the gap between resource-constrained microcontrollers and larger processors in robotic applications that are based on ROS.
- It is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket.
- Allows the hardware to communicate with the rest of the ROS2 system.
- It allows to use topics, services and logging features of ROS2 in microcontrollers.



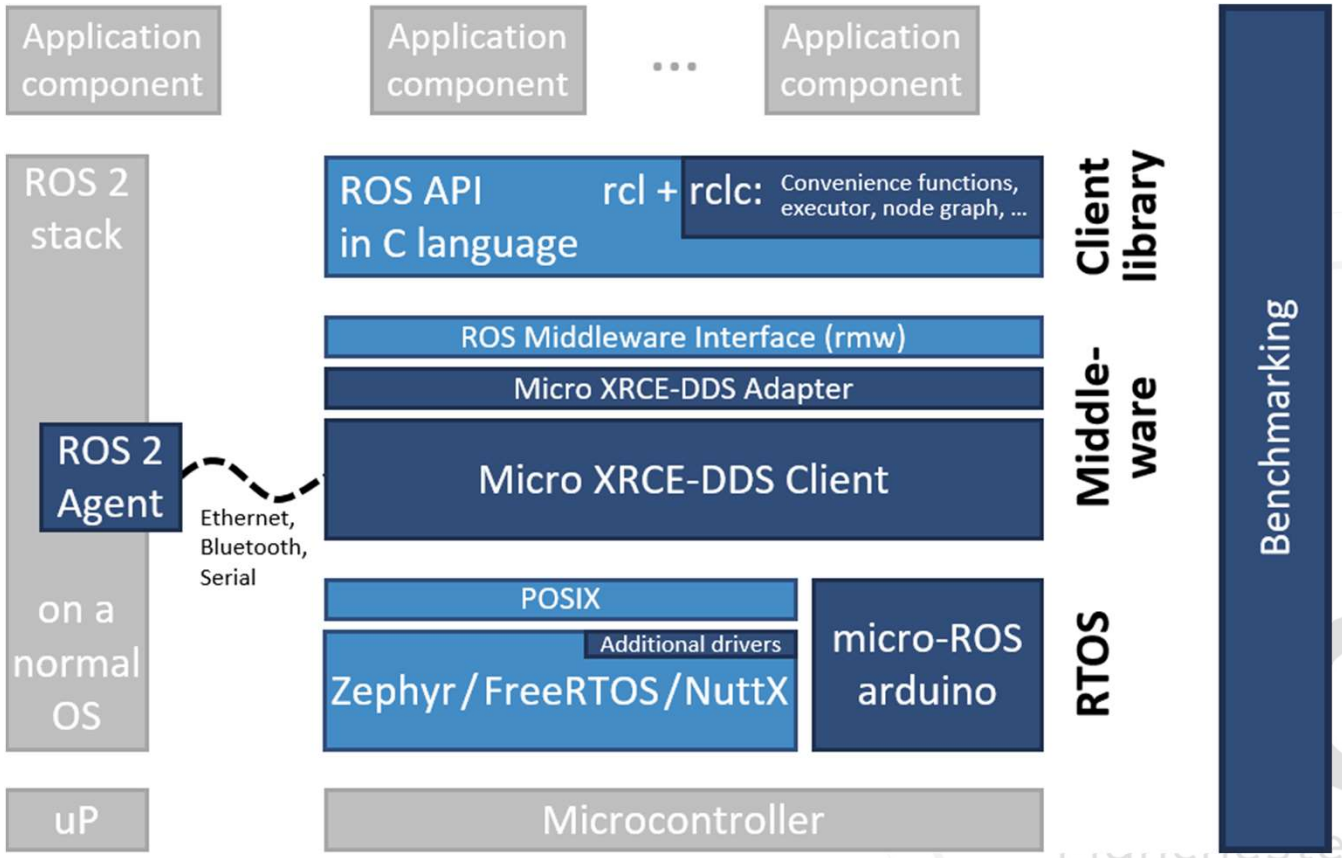
Micro-ROS

- It allows to create ROS Nodes inside microcontrollers or different hardware, allowing communication with different systems inside the ROS network.
- Microcontrollers are used in almost every robotic product. Typical reasons are:
  - Hardware access
  - Hard, low-latency real-time
  - Power saving
- micro-ROS | ROS 2 for microcontrollers

Micro-ROS


Wi-Fi
Serial
External Computer
NVIDIA Jetson Nano
Puzzlebot
Control Unit
User
Interface
Trajectory
SLAM
AI
Trajectory
Tracking
Low Level
Control
ROS Master

Micro-ROS


Application component
Application component
...
Application component
ROS 2 stack
ROS API in C language
rcl +
rclc:
Convenience functions, executor, node graph, ...
Client library
ROS Middleware Interface (rmw)
Micro XRCE-DDS Adapter
Micro XRCE-DDS Client
Middle-ware
ROS 2 Agent
Ethernet, Bluetooth, Serial
on a normal OS
POSIX
Additional drivers
Zephyr / FreeRTOS / NuttX
micro-ROS arduino
RTOS
Benchmarking
uP
Microcontroller

Micro-ROS Compatibility

- One of the most common usages of rosserial is to be used with a microcontroller (as described before) to communicate with sensors, actuators, etc.
- Unlike a computer running with ROS, the dedicated OS of the microcontrollers, allows the user to have more control over the timing functions required for certain hardware and control algorithms.
- Renesas EK RA6M5, ESP32, STM and Arduino are some of the microcontrollers supported by micro-ROS. (No support for Arduino UNO or Arduino MEGA2560)
- Micro-ROS offers support for freeRTOS, Zephyr and NutX

Installing micro-ROS binaries

```
mkdir -p ros2_utilities_ws/src
cd ~/ros2_utilities_ws/src
git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git

sudo apt update -y
sudo apt upgrade -y
sudo apt full-upgrade -y
sudo apt autoremove -y
sudo apt autoclean -y
sudo apt purge -y

cd ~/ros2_utilities_ws/
rosdep update
rosdep install --from-paths src --ignore-src -y

sudo apt install python3-pip -y
cd ~/ros2_utilities_ws/
colcon build
source install/local_setup.sh

cd ~/ros2_utilities_ws/
ros2 run micro_ros_setup create_agent_ws.sh
ros2 run micro_ros_setup build_agent.sh

source install/local_setup.sh
echo "source ~/ros2_utilities_ws/install/local_setup.bash" >> ~/.bashrc
dmesg | grep tty
sudo chmod a+rw /dev/tty*
sudo usermod -a -G dialout $USER
```

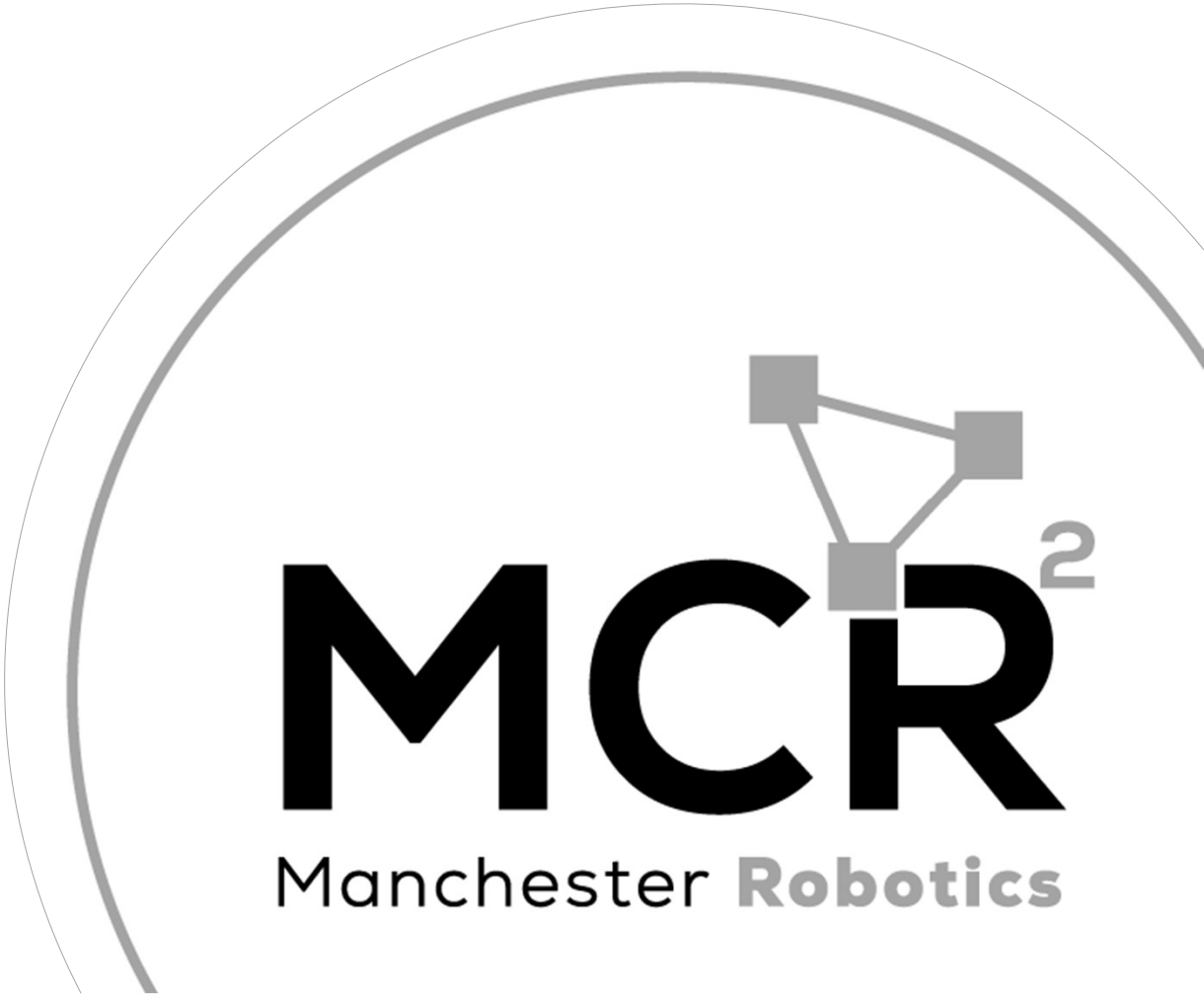
Adding the Arduino libraries

- Download the latest micro-ROS release for Arduino
  - Releases · micro-ROS/micro_ros_arduino (github.com)
  - Select the Humble release
  - Save it in a location that will remain untouched
- Open Arduino and add the library
  - Include it in your project using Sketch -> Include library -> Add .ZIP Library...
- Open the Tools -> Board Manager, and add the ESP32 by Expressif or Arduino SAM board support

Micro-ROS Communication

MCU Program

{Learn, Create, Innovate};



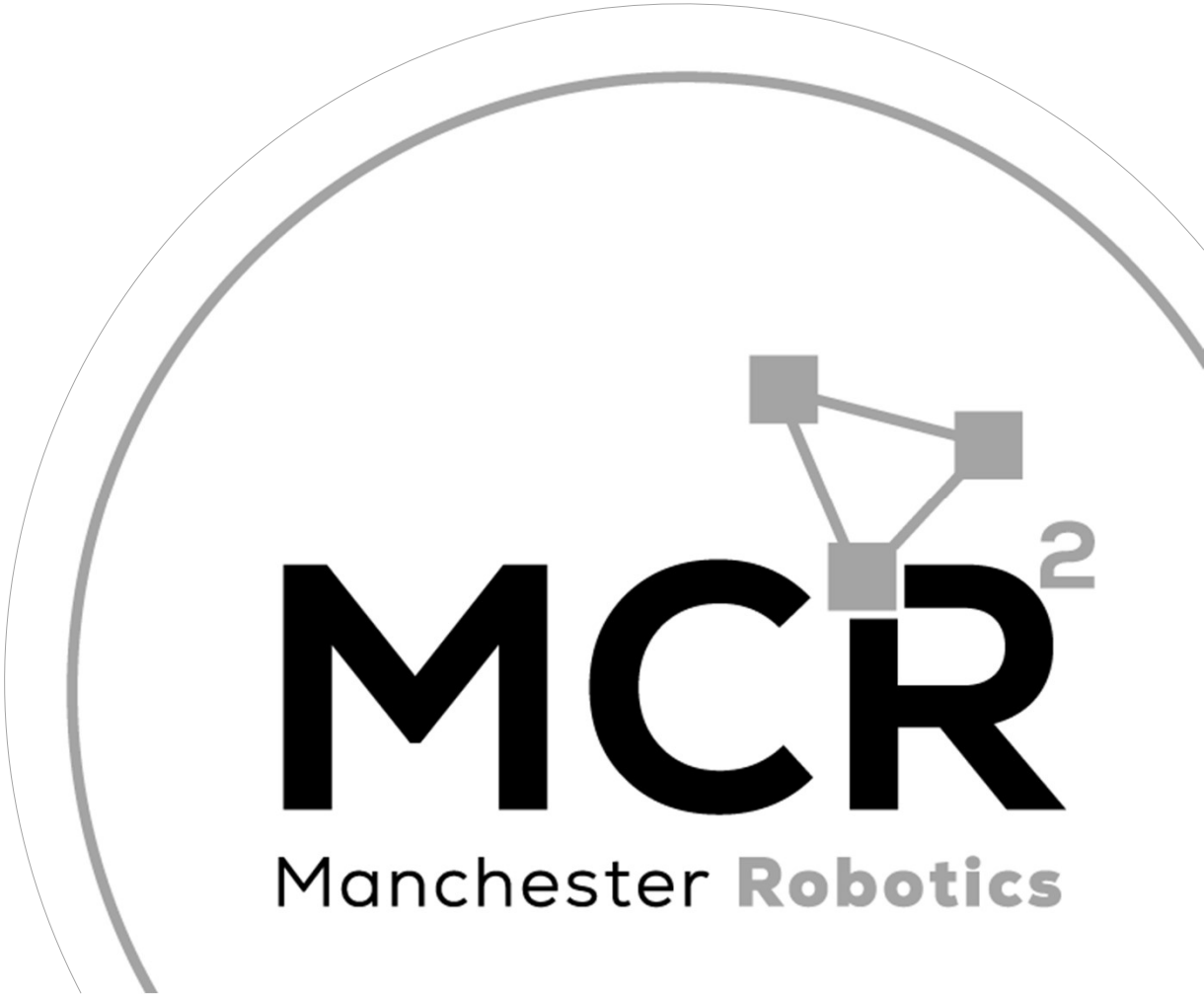
MCU Programming

- As stated before, STM32, Arduino and ESP32 are some of the most used development platforms because of their ease of use..
- Arduino and ESP32 boards can be programmed using the Arduino IDE.
- For all the activities and challenges in this session, the Arduino IDE will be used for programming.
- The activities and challenges shown in this presentation will be performed using a ESP32 WROOM, alongside a DC motor and a motor driver module (L298n) for the challenge.
- Please refer to the prerequisites of this session for the complete list of required components.

Micro-ROS Communication

ROS Sketch Structure

{Learn, Create, Innovate};



micro-ROS Sketch Structure

- Micro-ROS provides a ROS communication protocol that works over UART.
- Allows the board to become a ROS2 node which can directly publish and subscribe to ROS2 messages, publish TF transforms, and get the ROS2 system time.

ROS2 Sketch Structure
Variable declaration

- For every ROS Arduino program, several headers must be included before any other library or ROS message.
- Instantiate the handler, which allows our program to create publishers and subscribers. The node handle also takes care of serial port communications.
- Perform validations of the ROS Client Library
- Instantiate any publishers, subscribers to be used.
- Declare any ROS messages and variables to be used.
- Declare/define callback functions to be used with the subscribers.

ROS2 Sketch Structure

Setup section

- Initialize the ROS node handle. Node initialisation,
- Advertise any topics being published.
- Subscribe to the topics being used.
- Initialise variables, ports, functions, etc.

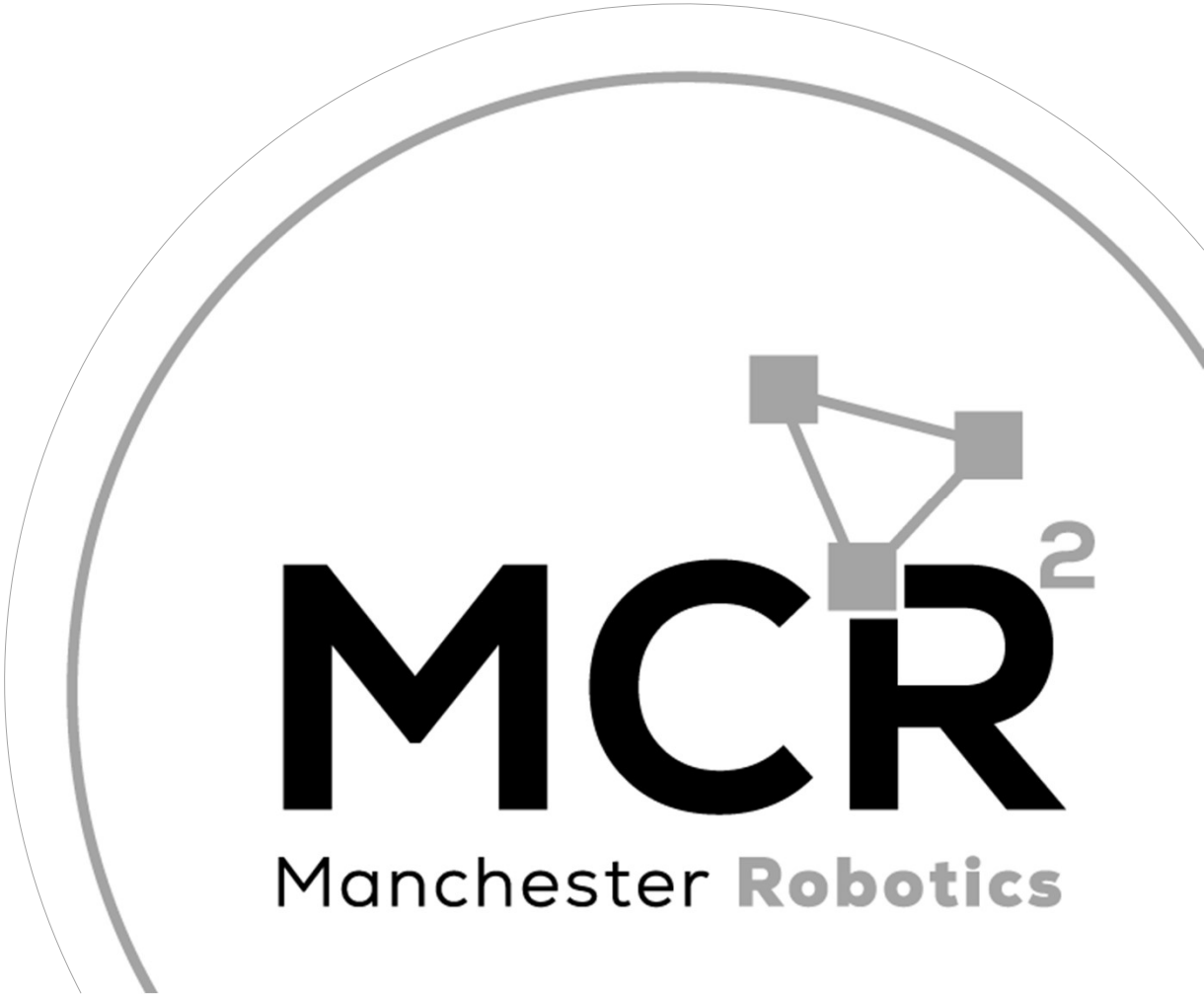
Loop section

- Run the main program.
- Loop and repeat actions.
- Handle callback functions.

Micro-ROS Serial Communication

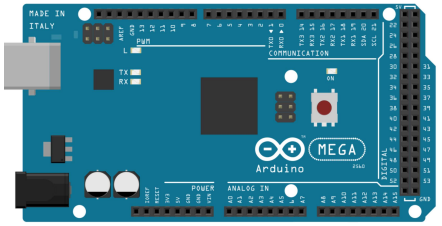
Publisher Activity

{Learn, Create, Innovate};



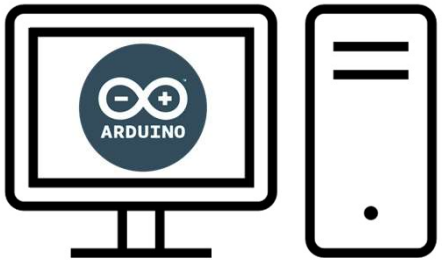
Requirements

- The following activity is based on the example tutorial found in the provided micro-ROS libraries
- This activity requires Arduino IDE to be installed.
- Any of the following boards are supported:
  - Arduino Portenta H7 M7 Core
  - Arduino Nano RP2040 Connect
  - OpenCR
  - Teensy
  - ESP32



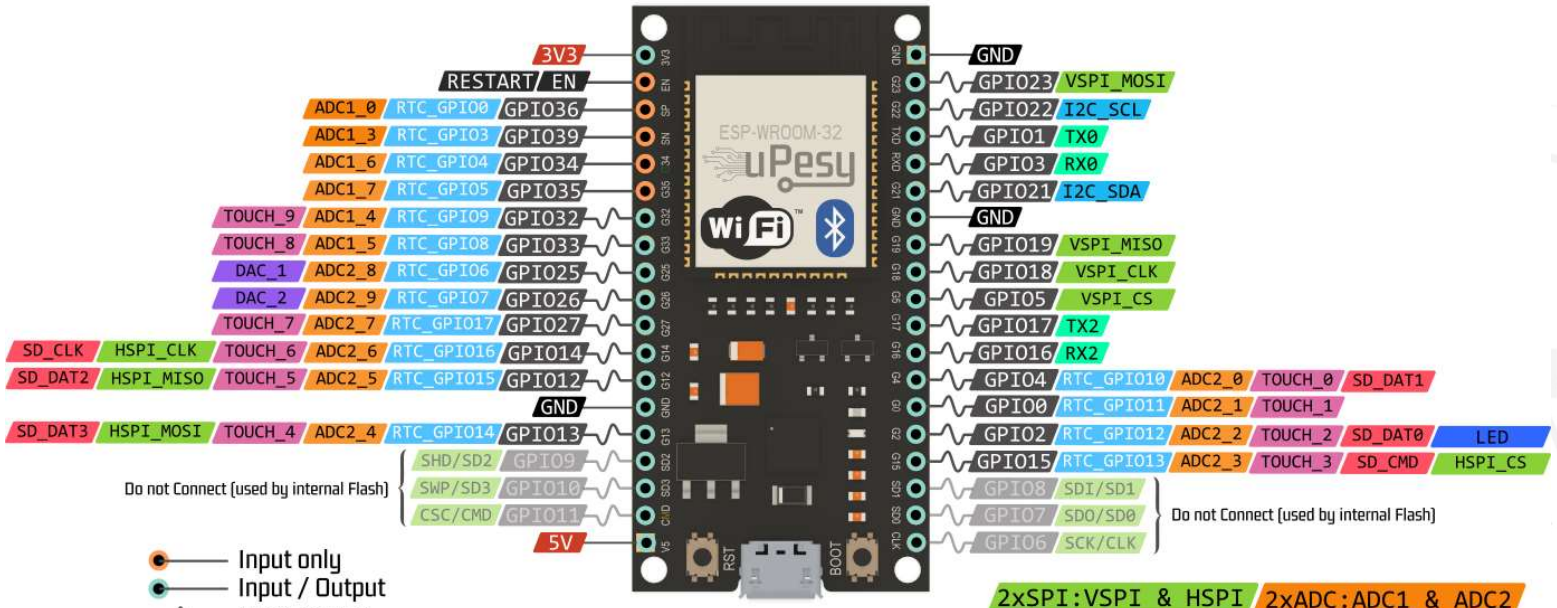
Arduino Due

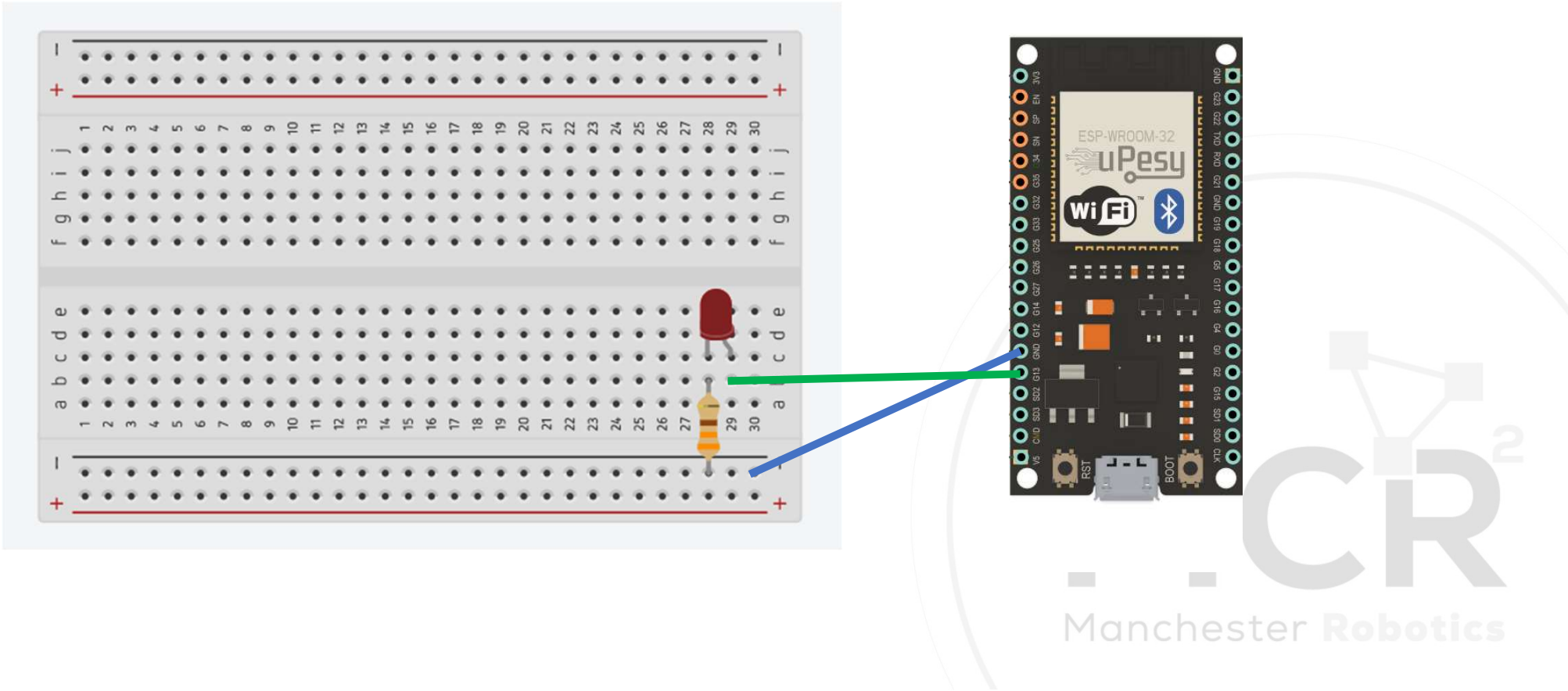

ESP32 board



Computer

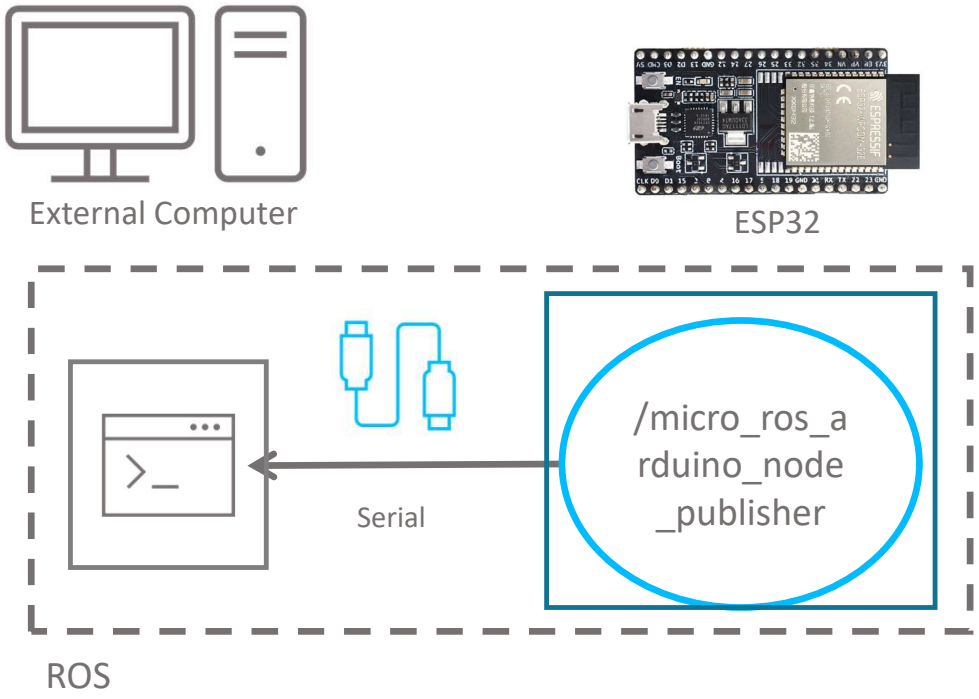
ESP32 Wroom DevKit Full Pinout




ESP-WROOM-32
uPesy
Wi Fi
RST
BOOT

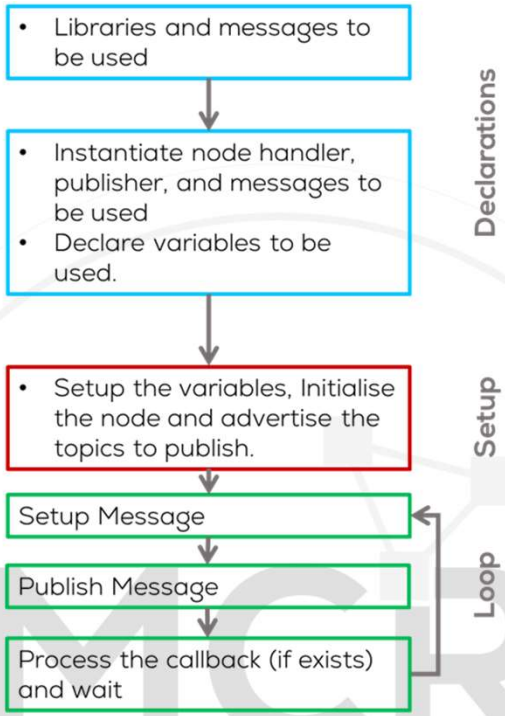
Description

- In this activity, a node running a simple publisher will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will publish a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.



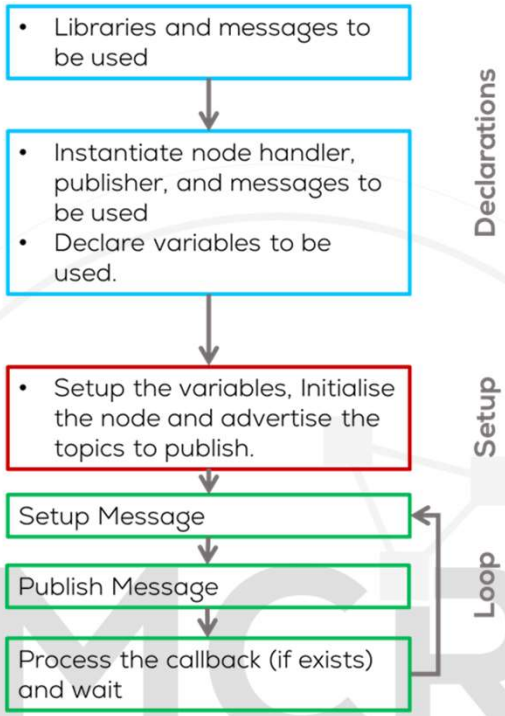
ESP32 Code - Declarations

```
#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>

#include <std_msgs/msg/int32.h>

rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

#define LED_PIN 13

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
```



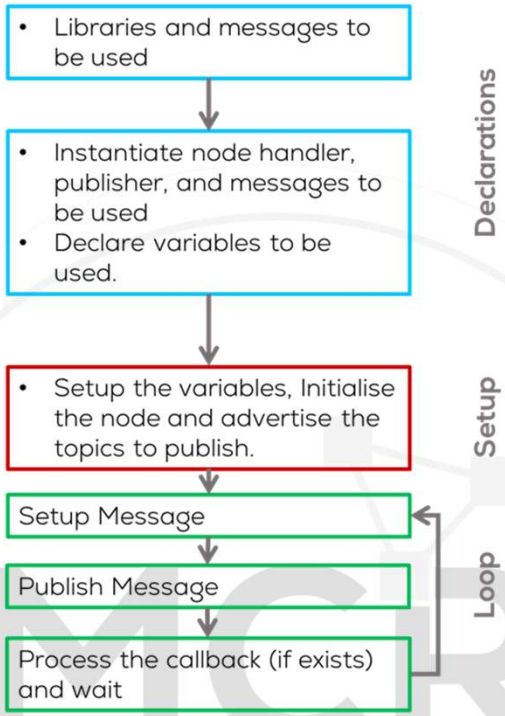
ESP32 Code – Function definition

```
void error_loop(){
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(100);
  }
}

void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
    msg.data++;
  }
}
```



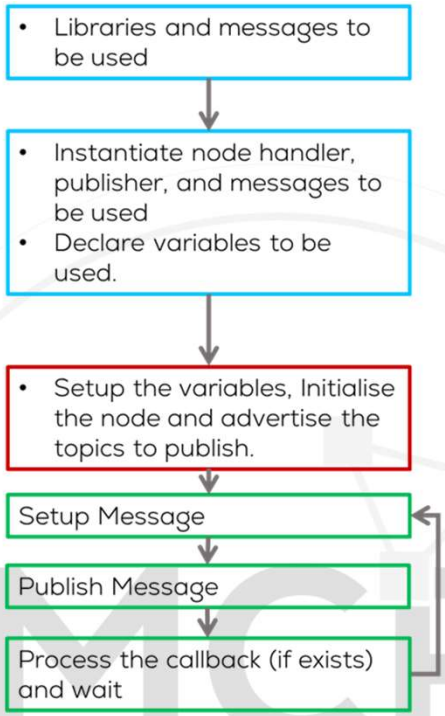
ESP32 Code – Setup

```
void setup() {
  set_microros_transports();

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);

  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(&node, "micro_ros_arduino_node", "", &support));

  // create publisher
  RCCHECK(rclc_publisher_init_default(&publisher, &node, ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "micro_ros_arduino_node_publisher"));

  // create timer,
  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(&timer, &support, RCL_MS_TO_NS(timer_timeout), timer_callback));

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer));

  msg.data = 0;
}
```



ESP32 Code – Loop

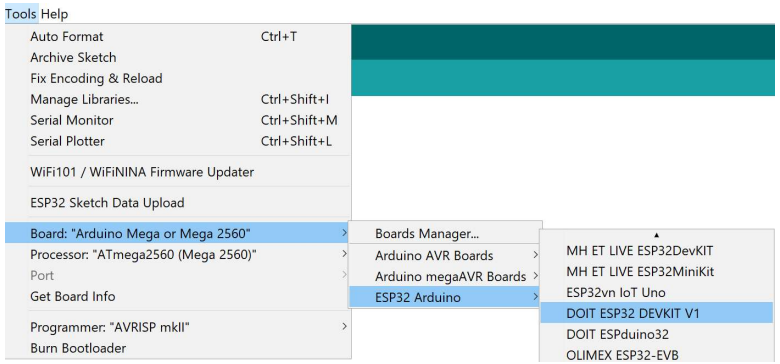
```
void loop() {
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

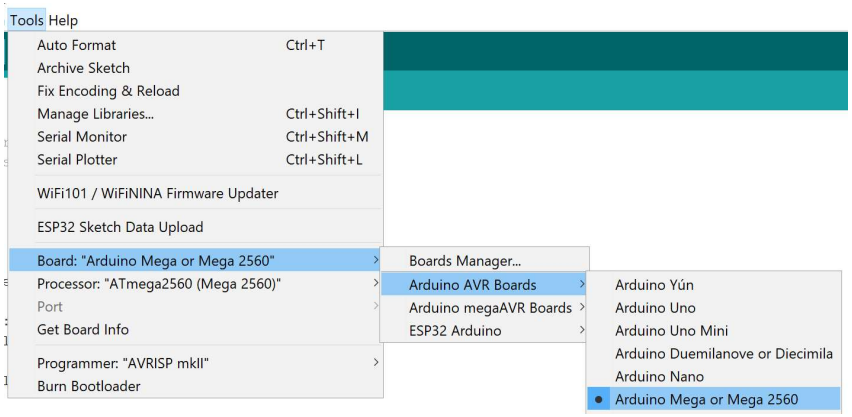


Activity

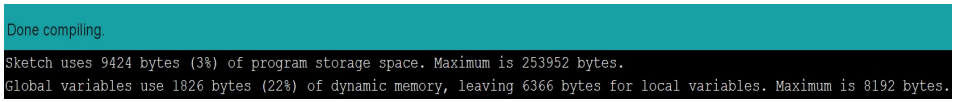
Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Open the example. File -> Examples -> micro_ros_arduino -> micro-ros_publisher
- Select the board to be used Tools -> Board ESP32 or Arduino Due
    - For Arduino Select Arduino SAM Boards>Arduino Mega or Mega 2560
    - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1





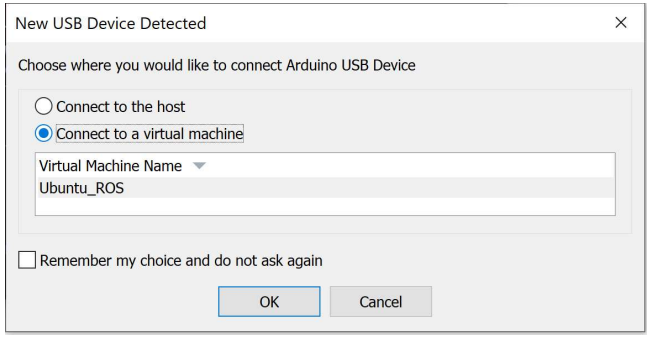
- Compile the code using by clicking check mark button located on the upper left corner.

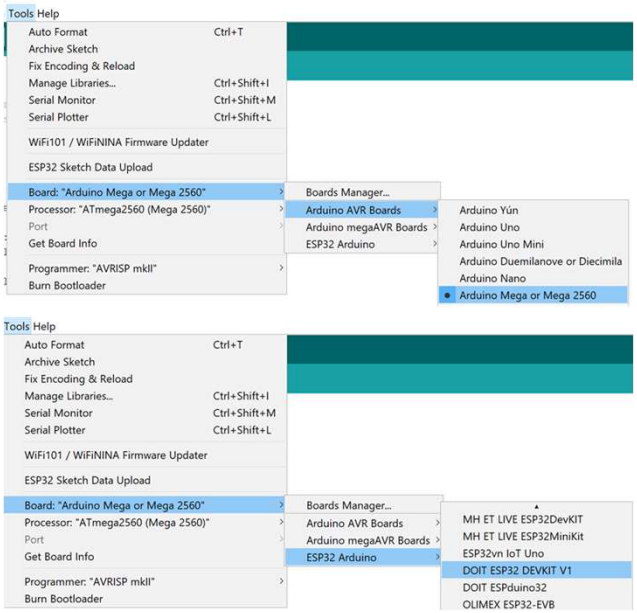


Activity

Uploading (Arduino IDE)

- Connect the board
- Select the port to be used Tools>Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
- Select the board to be used Tools -> Board ESP32 or Arduino Due
  - For Arduino Select Arduino SAM Boards>Arduino Mega or Mega 2560
  - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



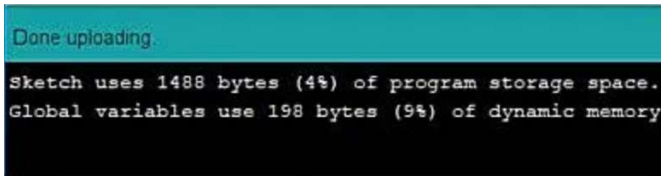


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user.
    - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Terminal

- In a new terminal use the command line tool rosrun and select the port type (USB or ACM). Verify that the agent starts working. Otherwise, press the reset button on your board

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

- In a new terminal subscribe to the topic using the command

```
ros2 topic echo /micro_ros_arduino_node_publisher
```

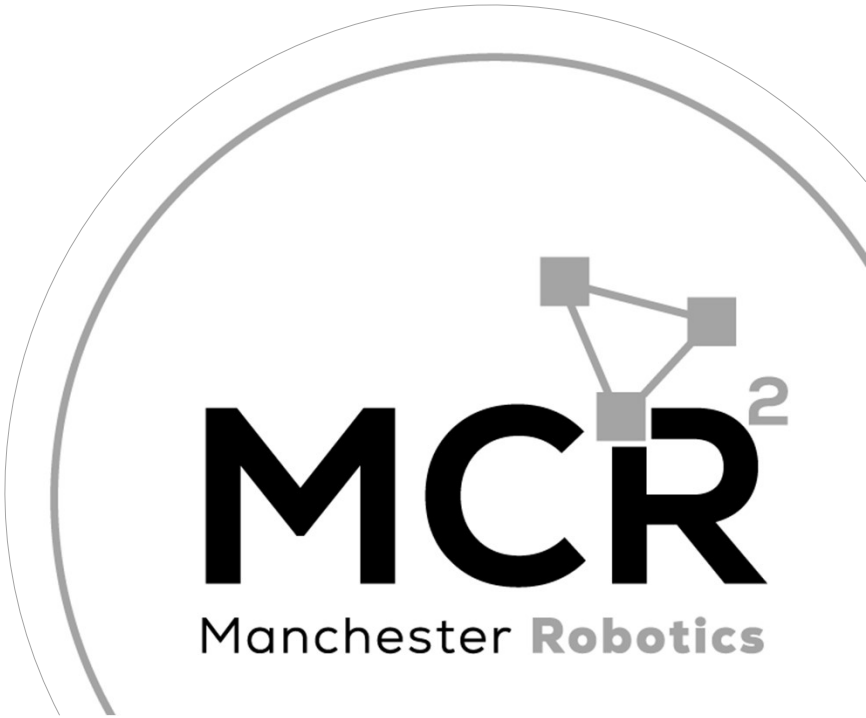
- You should see the following results

```
data: 6
---
data: 7
---
data: 8
---
data: 9
---
data: 10
---
data: 11
---
data: 12
---
data: 13
---
```

Micro-ROS Serial Communication

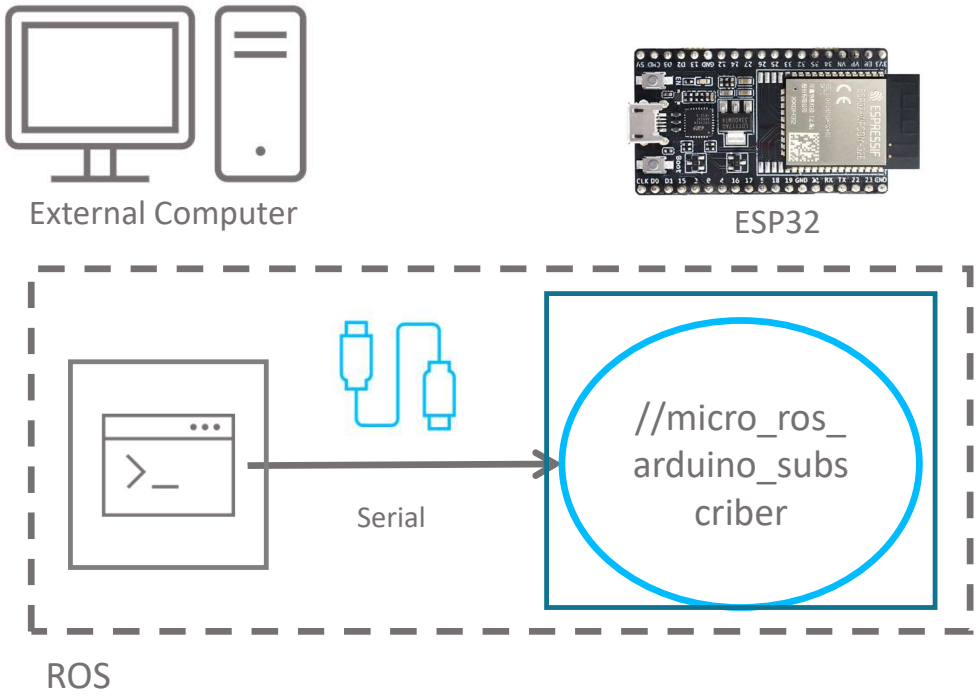
Subscriber Activity

{Learn, Create, Innovate};



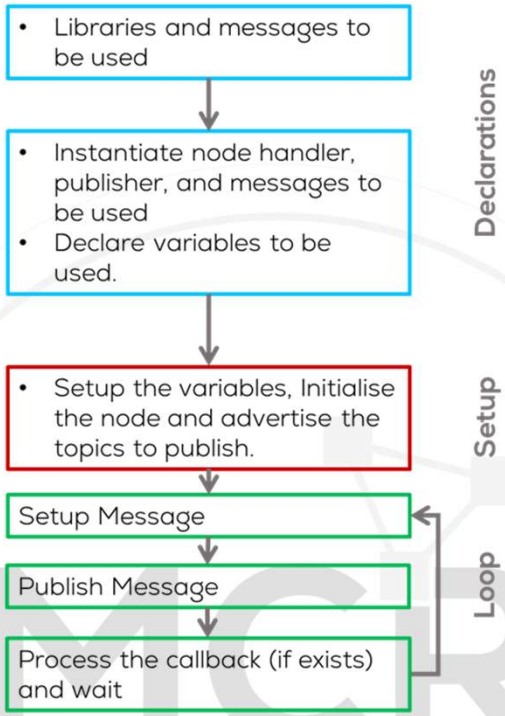
Description

- In this activity, a node running a simple subscriber will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will subscribe to a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.



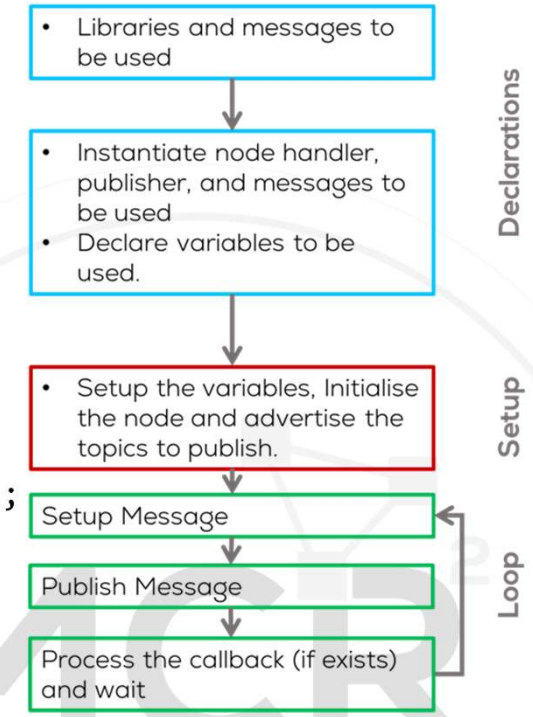
ESP32 Code - Declarations

```
#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>

#include <std_msgs/msg/int32.h>

rcl_subscription_t subscriber;
std_msgs__msg__Int32 msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

#define LED_PIN 13

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
```



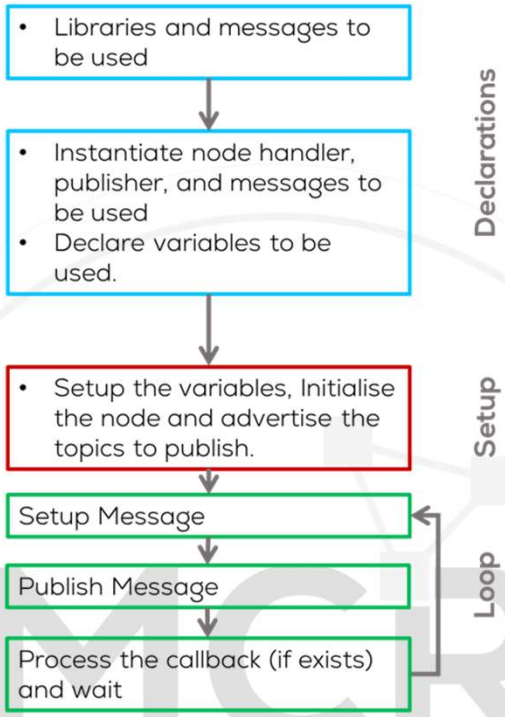
ESP32 Code – Function definition

```
void error_loop(){
  while(1){
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(100);
  }
}

void subscription_callback(const void * msgin)
{
  const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
  digitalWrite(LED_PIN, (msg->data == 0) ? LOW : HIGH);
}
```



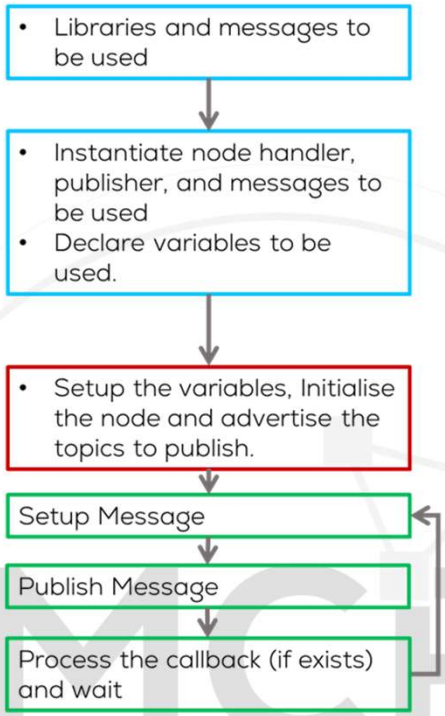
ESP32 Code – Setup

```
void setup() {
  set_microros_transports();

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH);

  delay(2000);

  allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(&node, "micro_ros_arduino_node", "", &support));

  // create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_arduino_subscriber"));

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg, &subscription_callback, ON_NEW_DATA));
}
```



ESP32 Code – Loop

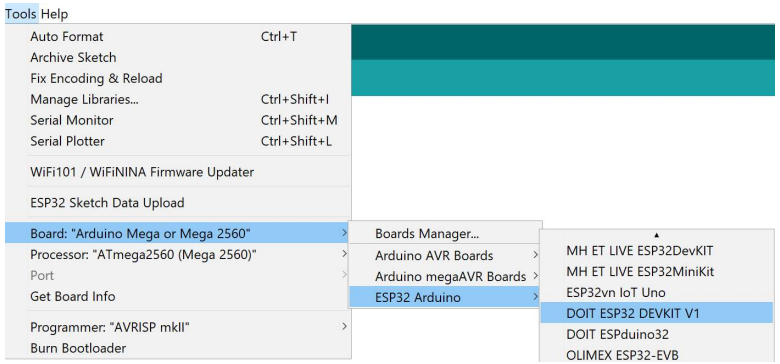
```
void loop() {
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

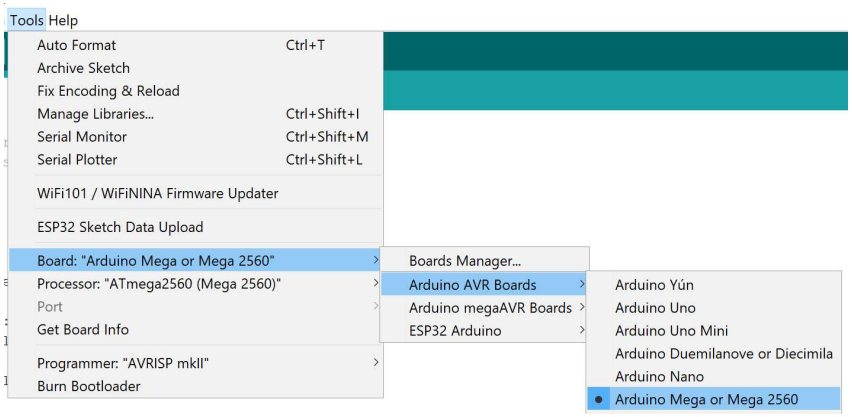


Activity

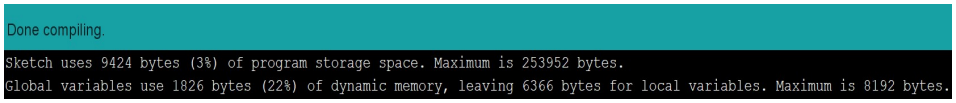
Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Open the example. File -> Examples -> micro_ros_arduino -> micro-ros_subscriber
- Select the board to be used Tools -> Board ESP32 or Arduino Due
  - For Arduino Select Arduino SAM Boards>Arduino Mega or Mega 2560
  - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1





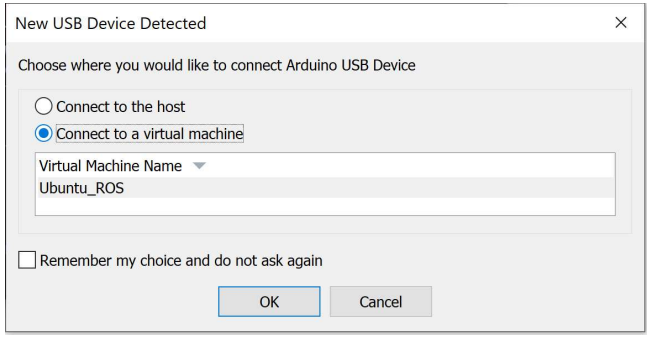
- Compile the code using by clicking check mark button located on the upper left corner.

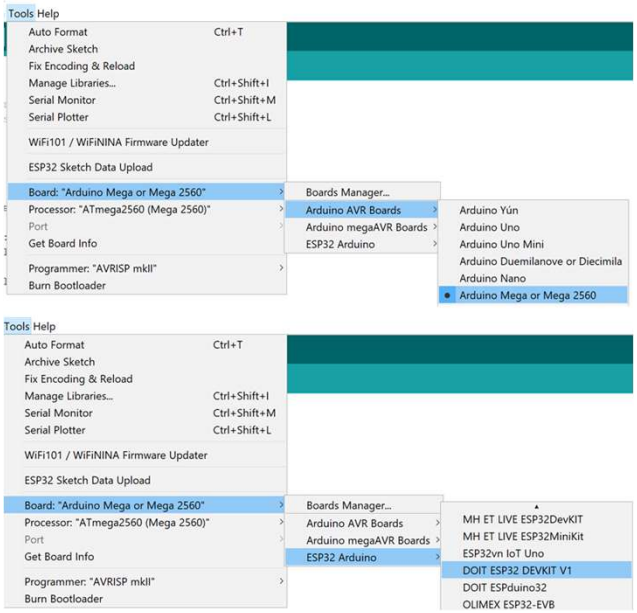


Activity

Uploading (Arduino IDE)

- Connect the board
- Select the port to be used Tools>Port
  - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
- Select the board to be used Tools -> Board ESP32 or Arduino Due
  - For Arduino Select Arduino SAM Boards>Arduino Mega or Mega 2560
  - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



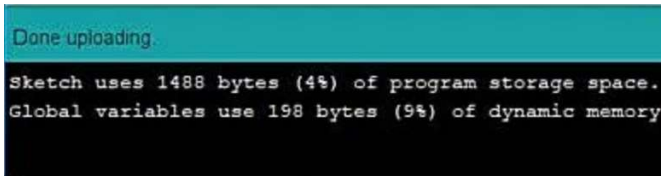


Activity

Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.

- The following message should appear o the IDE



Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user.
    - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*
sudo chmod 666 /dev/ttyUSB*
```

Terminal

- In a new terminal use the command line tool rosrun and select the port type (USB or ACM). Verify that the agent starts working. Otherwise, press the reset button on your board

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

- In a new terminal publish to the topic using the command

```
ros2 topic pub /micro_ros_arduino_subscriber --once std_msgs/msg/Int32 "data: 1"
```

- Change the content of data to 0 (off) or 1 (on) to see changes on the LED.