

## Lab01 - First Publisher and Subscriber

MTRN4231 - UNSW School of Mechanical and Manufacturing Engineering

### Preliminaries

As ROS runs natively in Ubuntu it is good to learn/brush up on bash commands. In particular, make sure you are familiar with the commands 'cd' to navigate through directories and 'ls' to list all files.

You may also find it beneficial to review the object-orientated coding sections of MTRN2500.

### Introduction

In this lab, you will create your first package containing a simple publisher and subscriber. You will then expand on this by changing the message type to accommodate other messages. Instructions are based on the Humble tutorials which can be found: [ROS Humble Tutorials](#).

### Task 1 - RQT

First, let's manually generate a message and publish it to a topic. This can be done through CLI or RQT. RQT allows a user to interface with ROS through a graphical user interface.

REMINDER: A topic can be called anything. A message of a particular type is \*published\* to a topic. Subscribers have call-back functions that are triggered when a new message is published to a topic. A topic can have multiple publishers and subscribers.

Open a new terminal and type:

rqt

Add a message publisher interface by clicking on 'Plugins->Topic->Message Publisher', Then:

- Set Topic to '/first\_topic'
- Set Type to 'std\_msgs/msg/String'
- Hit '+' to create a publisher
- Press the drop-down arrow next to the publisher and fill the expression column with `\hello world"`
- Select the checkbox to start publishing

You should now be publishing a message of type string to the topic '/first\_topic' at a frequency of 1 Hz. NOTE: While we are publishing to the topic you will not see the message until we subscribe to it in task 2.

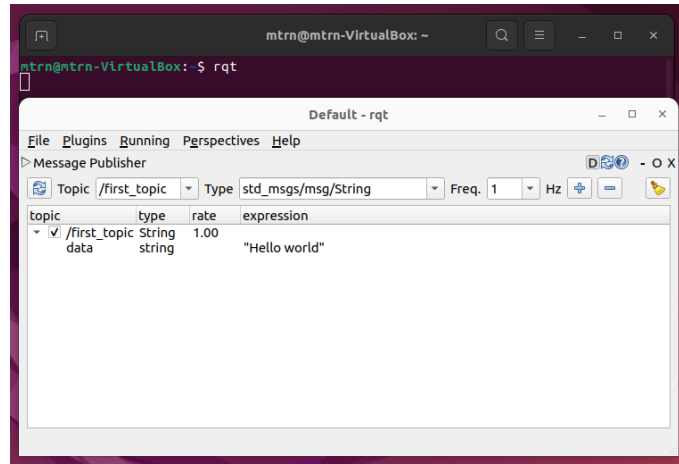


Figure 1: RQT correct setup for task 1.

## Task 2 - CLI Topic Viewer

To verify the message is publishing, we will use the CLI tools to subscribe to `'/first_topic'`.

First, open a new terminal and type the command:

```
ros2 topic list
```

Alternatively, to display topics and related message types use:

```
ros2 topic list -t
```

You should see the following topics listed:

```
/first_topic
/parameter_events
/rosout
```

To subscribe to a topic, run:

```
ros2 topic echo /first_topic
```

To subscribe to a different topic, replace `'/first_topic'`. While still subscribed, edit the expression of the message in RQT to see that the terminal window updates.

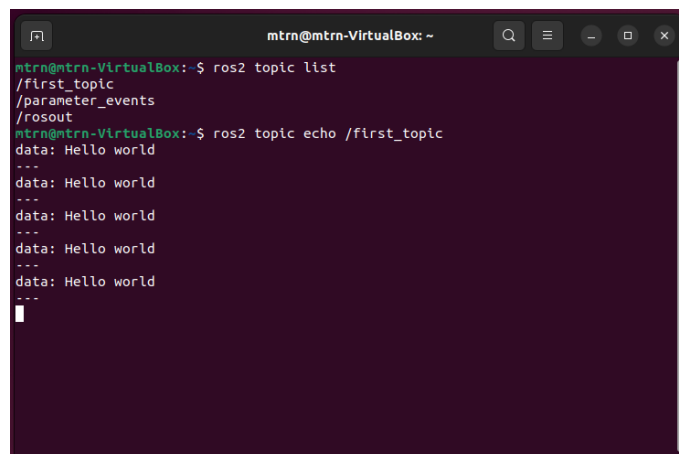


Figure 2: Terminal output for Task2.

## Task 3 - Creating a Package

While the above tasks demonstrate the communication method with ROS2 this was conducted manually. Now let us create a package and two nodes. One to publish the message, the other to subscribe to the message and display it.

REMINDER: A package in ROS contains one or more nodes. Packages are generally created for each separate component in your project (Eg. you may have separate packages for: Vision, Navigation, Localisation., Brain,etc....). While you can have Python and C++ nodes in the same package, build conflicts can be encountered and it is generally not recommended.

Navigate to `/4231/lab1_workspace/src` and create a C++ package using:

```
ros2 pkg create --build-type ament_cmake cpp_pubsub
```

## Task 4 - Creating the Publisher Nodev

Navigate to `/4231/lab1_workspace/src/cpp_pubsub/src` and run the command:

```
wget -O publisher_member_function.cpp https://raw.githubusercontent.com/ros2/examples/humble/rclcpp/topics
```

Please review the lecture and the code for yourself. In summary, First an object is created which inherits from the ROS node class. Then a publisher object is created and attached to a topic. An asynchronous timer is also created which calls a callback function when it ends. The callback function creates a basic message and then publishes it. A full explanation and run-through of the code can be found in the tutorial link listed above.

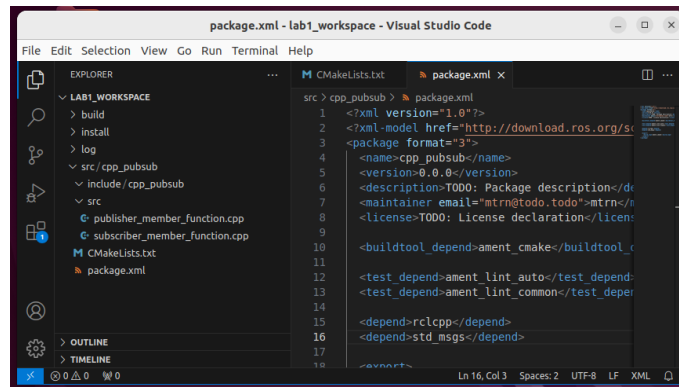


Figure 3: Correct file structure after downloading both talker and listener examples. The workspace has also been compiled.

### Adding Dependencies

To make our node work we should add dependencies, such that the compiler can ensure the required packages are available. In the file `'package.xml'` add the following package dependencies below the `ament_cmake` dependency.

```
<depend>rclcpp</depend>
<depend>std_msgs</depend>
```

### Fixing CMake

To compile our node we need to program the create the executable, and then link required packages to the executable. In the file `'CMakeLists.txt'`, below the existing dependency `find_package(ament_cmake REQUIRED)`, add the lines:

```
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
```

After that, add the executable and name it `talker` so you can run your node using `ros2 run`:

```
add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
```

Finally, add the `install(TARGETS...)` section so `ros2 run` can find your executable:

```
install(TARGETS
  talker
  DESTINATION lib/${PROJECT_NAME})
```

### Building

Navigate to the workspace directory (`/4231/lab1_workspace`). Run the command:

```
colcon build
```

## Sourcing

After a successful build, the workspace needs to be sourced so ROS is able to identify the executables.

**NOTE: For every terminal instance for every desired workspace, this command must be run, get familiar with this command:**

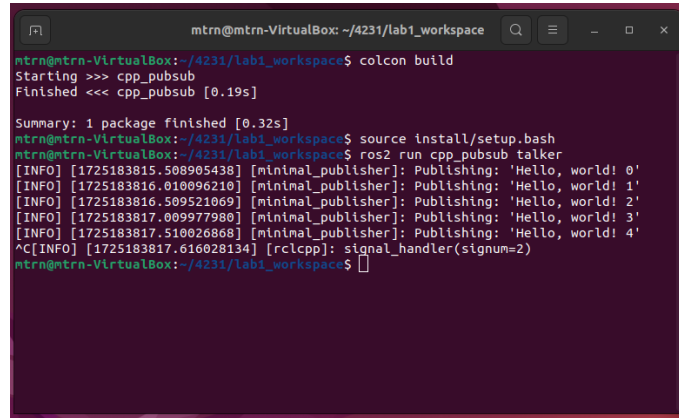
```
source install/setup.bash
```

## Testing

To run the node, source a terminal window and execute the command:

```
ros2 run cpp_pubsub talker
```

In a separate terminal window, subscribe to the topic 'topic' through the command line. You should now see the message being published every second.



```
mtrn@mtrn-VirtualBox: ~/4231/lab1_workspace
mtrn@mtrn-VirtualBox:~/4231/lab1_workspace$ colcon build
Starting >>> cpp_pubsub
Finished <<< cpp_pubsub [0.19s]

Summary: 1 package finished [0.32s]
mtrn@mtrn-VirtualBox:~/4231/lab1_workspace$ source install/setup.bash
mtrn@mtrn-VirtualBox:~/4231/lab1_workspace$ ros2 run cpp_pubsub talker
[INFO] [1725183815.508905438] [minimal_publisher]: Publishing: 'Hello, world! 0'
[INFO] [1725183816.010096210] [minimal_publisher]: Publishing: 'Hello, world! 1'
[INFO] [1725183816.509521069] [minimal_publisher]: Publishing: 'Hello, world! 2'
[INFO] [1725183817.009977980] [minimal_publisher]: Publishing: 'Hello, world! 3'
[INFO] [1725183817.510026868] [minimal_publisher]: Publishing: 'Hello, world! 4'
^C[INFO] [1725183817.616028134] [rclcpp]: signal_handler(signum=2)
mtrn@mtrn-VirtualBox:~/4231/lab1_workspace$
```

Figure 4: Terminal example with compiling, sourcing and running.

## Modifications

To familiarise yourself with publishers, make these changes to your code:

- Change the rate of publishing by editing the timer initialisation
- Edit the contents of the message by changing the string
- Change the name of the topic from 'topic' to something more useful

## Task 5 - Creating the Subscriber

Navigate to /4231/lab1\_workspace/src/cpp\_pubsub/src and run the command:

```
wget -O subscriber_member_function.cpp https://raw.githubusercontent.com/ros2/examples/humble/rclcpp/topic
```

Please review the lecture and the code for yourself. In summary, First an object is created which inherits from the ROS node class. Then a subscriber object is created and attached to a topic. The callback function is bound to the subscriber such that when a message is published to a topic the callback is activated. A full explanation and run-through of the code can be found in the tutorial link listed above. If in Task 4 you updated the name of the topic from 'topic', make sure to also update it in the subscriber\_member\_function.cpp file.

## CMake List

No extra packages are required as such package.xml does not need to be updated. However, the CMake list must be updated to compile the new node. Add the executable and then add the executable to the install targets list:

```
add_executable(listener src/subscriber_member_function.cpp)
ament_target_dependencies(listener rclcpp std_msgs)
```

```
install(TARGETS
  talker
  listener
```

```
DESTINATION lib/${PROJECT_NAME})
```

## Running

First, build the node as mentioned in Task 4, and don't forget to source your workspace. Then run the command:

```
ros2 run cpp_pubsub listener
```

Provided `talker` is running, you should now see the published message displayed on the subscriber terminal.

## Task 6 - Change Message Type to an Integer for Both Publisher and Subscriber

Modify the existing message from a string to the `std_msgs/msg/Int64` message type. As no new packages are needed, only the `.cpp` files are required to change. Modify the publisher and subscriber types, function types, and message type.

The message definition can be found at: [Int64 Message Definition](#)

```
std_msgs::msg::Int64 msg;
msg.data = 1;
```

**NOTE:** Technically `std_msgs` has been replaced by `example_msgs`, however the documentation has not been updated to reflect this yet. As such, use `std_msgs/msg/Int64`.

## Task 7 - Change Message Type to a Pose for Both Publisher and Subscriber

A useful message type is the Pose message, which encodes a position and orientation. Modify the existing message from a string to the `geometry_msgs/msg/Pose` message type. The package `geometry_msgs` is required and must be added to the CMake and package list. The message definition can be found at: [Pose Message Definition](#) and is defined as:

```
geometry_msgs::msg::Pose
```

A data field can be set as such:

```
geometry_msgs::msg::Pose msg;
msg.position.x = 0.0;
```

Refer to the message definition to fill out the remaining fields.

### Include

```
#include "geometry_msgs/msg/pose.hpp"
```

### CMake

```
find_package(geometry_msgs REQUIRED)
```

### Package.xml

```
<depend>geometry_msgs</depend>
```

## Task 8 - Custom Message

When working on a large project, it is better to use custom messages. That way you can easily add extra data fields if required. Follow the tutorial [Custom Messages](#) to create a custom message. Then modify your nodes created in the previous tasks to read and publish your message.

## Extensions

- Extend the message types by implementing a `'pose_stamped'` message
- Make a publisher and subscriber in Python

## Debugging

- Make sure to source your workspace
- Make sure you are building in the right directory
- Make sure you have added the executable in CMake
- Make sure your message types and topic names match