Experiment 2: Introduction ROS2 Programming

Installation:

Step 1:

https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

Step 2:

https://docs.ros.org/en/foxy/Tutorials/Colcon-Tutorial.html

Before creating your first node you need to:

- Create a ROS2 workspace and source it.
- Create a Python package.

ROS organizes the program using packages. A package contains Cpp, Python, setup, compilation, and parameters files. They are:

- package.xml file containing meta-information about the package
- setup.py containing instructions for how to install the package
- setup.cfg is required when a package has executable so that ros2 run can find them
- /<package_name> a directory with the same name as the package, used by ROS2 tools to find the package that contains __init__.py

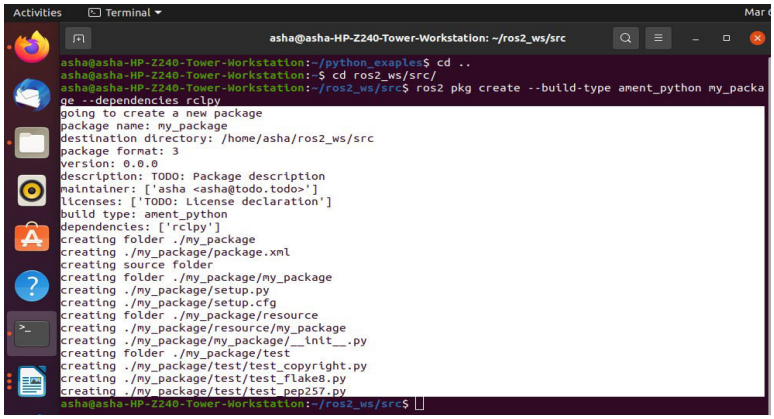
When you want to create packages, you need to work in a particular ROS workspace known as **ROS workspace**. The ROS2 workspace is the directory in your hard disk where your **ROS2. packages reside to be usable by ROS2. Usually, the ROS2 workspace directory is called ros2_ws**

Execute in Terminal #1

```
mkdir -p ~/ros2_ws/src
```

Inside this workspace, there is a directory called **src**. This folder contains all the packages created. Every time you create a package, you have to be in the directory **ros2_ws/src**.

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_package --dependencies rclpy
```



ros pkg create <package_name> --build-type ament_python my_package --dependencies <package_dependencies>

The **package_name** is the name of the package you want to create, and **package_dependencies** are the names of other ROS packages that your package depends on.

Execute in Terminal #1

gedit ~/.bashrc

source /opt/ros/foxy/setup.bash
source ~/ros2_ws/install/setup.bash
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash

Execute in Terminal #1

ros2 pkg list
ros2 pkg list | grep my_package

ros2 pkg list: Gives you a list with all packages in your ROS system.

ros2 pkg list | grep my_package: Filters, from all of the packages located in the ROS system, the package is named my_package.

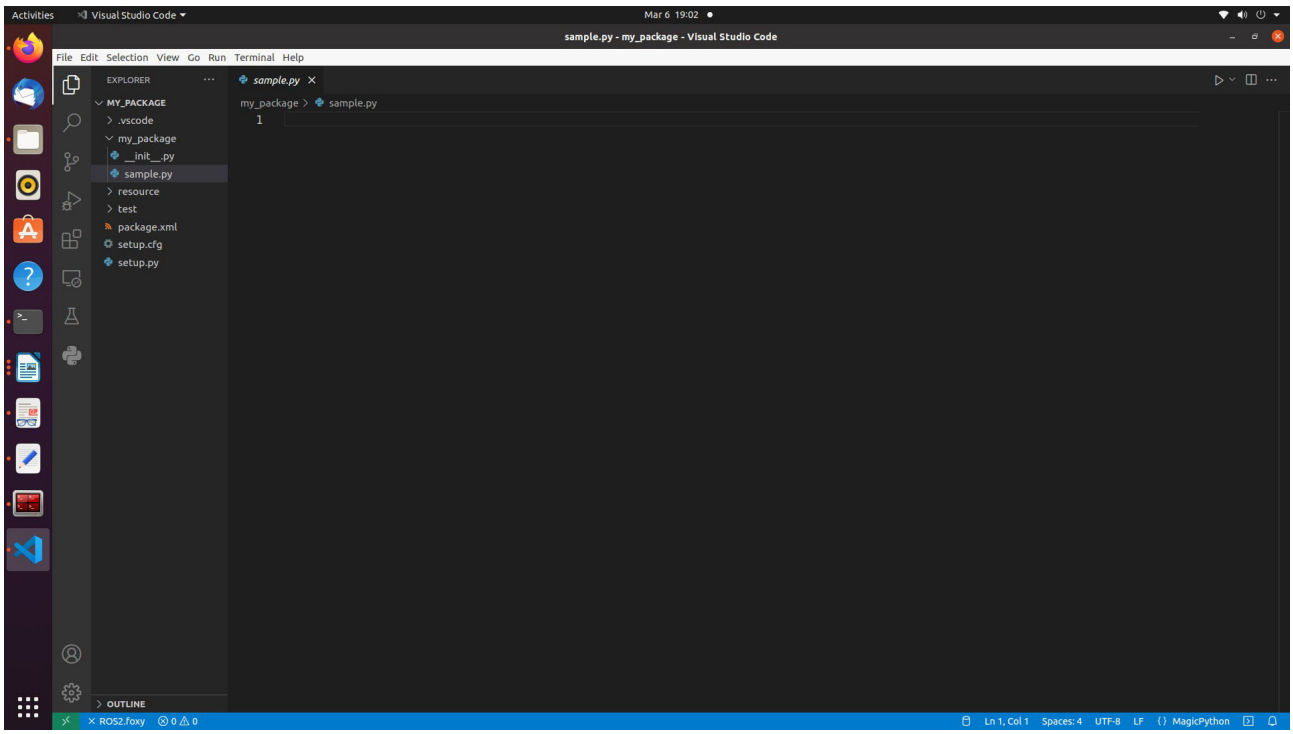
cd ~/ros2_ws
colcon build
colcon build --packages-select <package_name>
colcon build --packages-select my_package

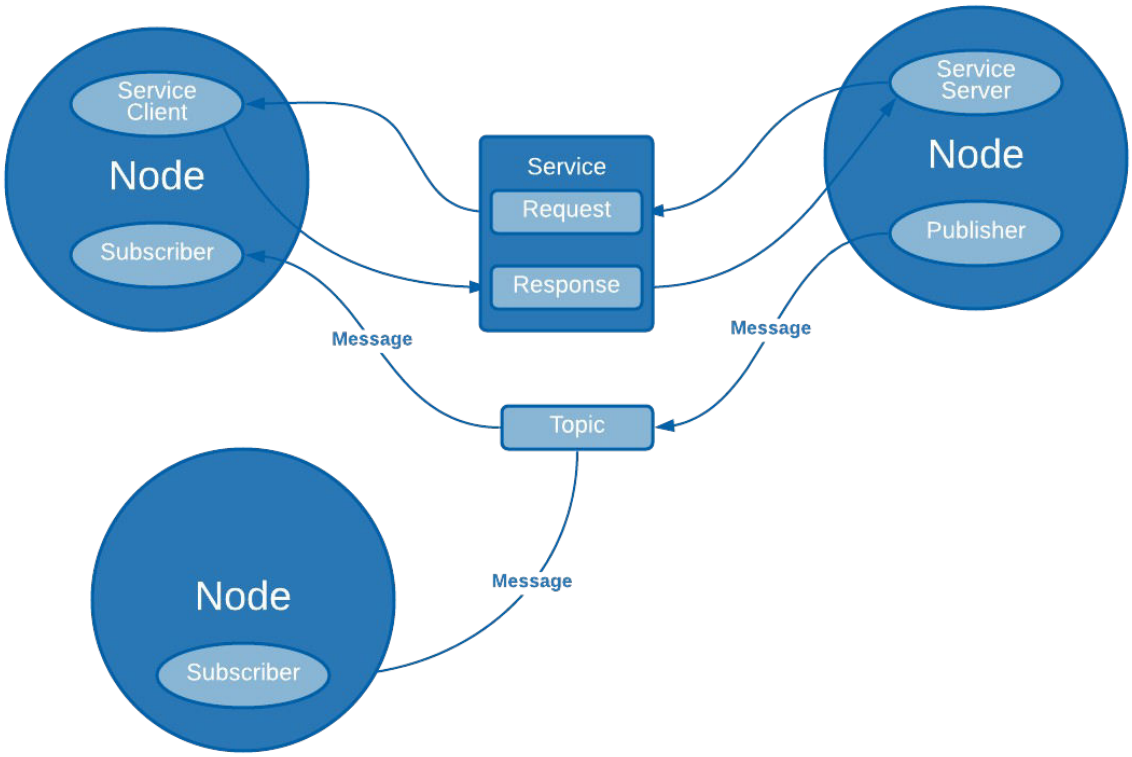
1. Create a Python file that will be executed in the **my_package** (all Python scripts) directory inside **my_package** folder.

Execute in Terminal #1

src/my_package/my_package
touch sample.py
chmod +x sample.py

Right click on the folder my_package and open with Visual Studio Application





Type the following code

```
#!/usr/bin/env python3
import rclpy
```

```
from rclpy.node import Node

class MyNode(Node):  #MIDIFY NAME OF THE CLASS
    def __init__(self):
        # call super() in the constructor in order to initialize the Node object
        # the parameter we pass is the node name
        super().__init__('sample') #MIDIFY NAME OF THE NODE
        # create a timer sending two parameters:
        # - the duration between 2 callbacks (0.2 seeconds)
        # - the timer function (timer_callback)
        self.create_timer(0.2, self.timer_callback)

    def timer_callback(self):
        # print a ROS2 log on the terminal with a great message!
        self.get_logger().info("Congratulation for starting your Robot Operating Syatem
Lab!!")


def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)
    # declare the node constructor
    node = MyNode() #MIDIFY NAME OF THE NODE
    # pause the program execution, waits for a request to kill the node (ctrl+c)
    rclpy.spin(node)
    # shutdown the ROS communication
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Execute in Terminal #2

```
cd ~/ros2_ws/src/my_package
mkdir launch
cd launch
touch my_package_launch_file.launch.py
chmod +x my_package_launch_file.launch.py
```

Type the following in the my_package_launch_file.launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(
            package='my_package',
```

```
            executable='sample',
            output='screen'),
    ])
```

Modify the setup.py file to generate an executable from the Python file you just created.

```
from setuptools import setup
from glob import glob
import os

package_name = 'my_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name), glob('launch/*')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        'sample = my_package.sample:main'
        ],
    },
)
```

The main objective of this code is to add an entry point to the script you created a few moments ago. To do that, work with a dictionary named entry_points. Inside it, you find an array called console_scripts. Add the node information to generate the executable. The objective of this code is to install the launch files. For example, with the package named "my_package", this will install all the launch files from the launch/folder, into ~/ros2_ws/install/my_python_pkg/share/my_python_pkg/launch/.

```
import os
from glob import glob
from setuptools import setup
package_name = 'my_package'
setup(

    #code
```

```
    ...


    data_files=[
            ('share/ament_index/resource_index/packages',
                ['resource/' + package_name]),
            ('share/' + package_name, ['package.xml']),
            (os.path.join('share', package_name), glob('launch/*.launch.py'))
        ],

    #code

)
```

Compile your package file as was previously explained.

Execute in Terminal #1

```
cd ~/ros2_ws
colcon build
source ~/ros2_ws/install/setup.bash
```

Finally, Now you must execute it. Use the following command that you already know. Execute the roslaunch command in the terminal to launch your program.

Execute in Terminal #1

```
ros2 launch my_package my_package_launch_file.launch.py

ctrl+C
```

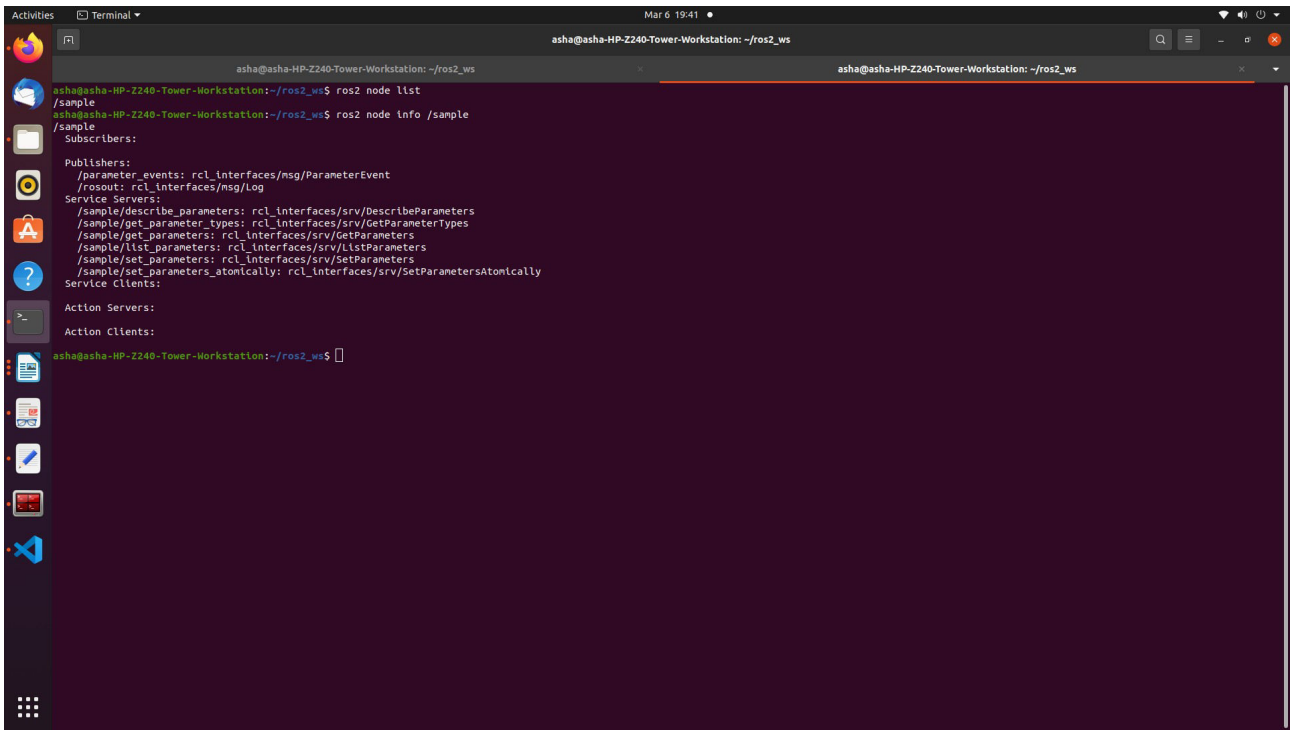
Execute in Terminal #1

```
ros2 run  my_package sample
```

In main function declare the MyNode class, which you will use to start the node as such. Within this, you have the __init__method and the timer_callback method.
super().__init__('<node_name>')
timer_callback method creates a message with the counter value appended and publishes it to the console with get_logger().info.
self.get_logger().info(""your message to be printed")

Execute in Terminal #2

```
ros2 node list
ros2 node info <node_name>
ros2 node info /sample
```



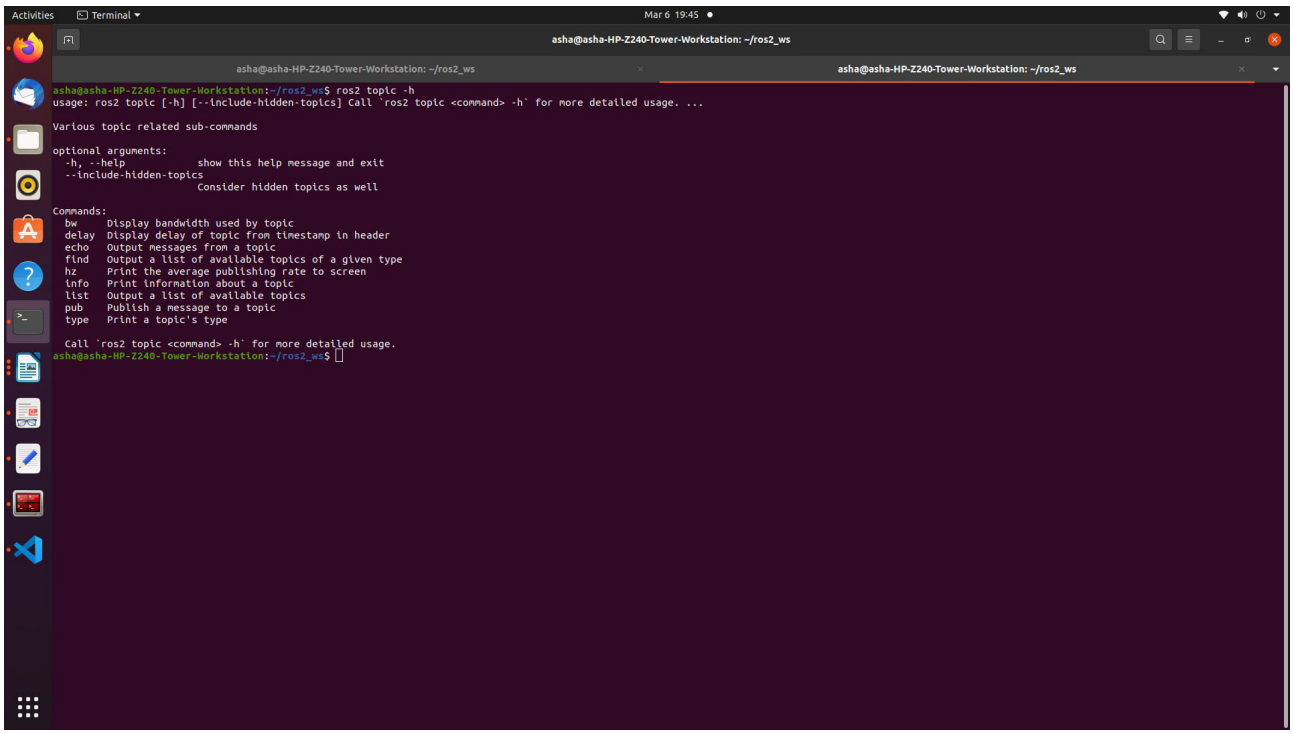
Understanding ROS2 Topics: Publishers & Subscribers

What will you learn about in this unit?

- Topics
- Basic topic commands
- Topic publishers
- Topic subscribers

Execute in Terminal #1

source /opt/ros/foxy/setup.bash
ros2 topic -h



OOP Python Code Template for Nodes

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node


class MyCustomNode(Node): # MODIFY NAME
        def __init__(self):
                super().__init__("node_name") # MODIFY NAME


def main(args=None):
        rclpy.init(args=args)
        node = MyCustomNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
            main()
```

Execute in Terminal #1

```
cd ros2_ws/src/my_package/my_package/
touch robot_publisher.py
chmod +x robot_publisher.py
ros2 interface show example_interfaces/msg/String
```

open the my_package using Visual Studio and edit the file robot_publisher.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String

class RobotPublisher(Node):

    def __init__(self):
        super().__init__("robot_publisher")
        self.robot_name_="ROBOT"
        self.publisher_ = self.create_publisher(String, "robot_news", 10)
        self.timer_ = self.create_timer(0.5, self.publish_news)
        self.get_logger().info("Node Started")


    def publish_news(self):
        msg = String()
        msg.data = "Hello " + str(self.robot_name_)
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = RobotPublisher()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Edit the setup.py

```
from setuptools import setup

package_name = 'my_package '

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
```

```
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
             'sample = my_package.sample:main',
            'robot_publisher = my_package.robot_publisher:main'
        ],
    },
)
```

Execute in Terminal #1

```
colcon build --packages-select my_package –symlink-install
```

Execute in Terminal #2

```
source ~/.bashrc
ros2 run my_package robot_publisher
```

Execute in Terminal #3

```
source ~/.bashrc
ros2 topic echo /robot_news
```

Subscriber node

Execute in Terminal #1

```
cd ros2_ws/src/my_package/my_package /
touch robot_subscriber.py
chmod +x robot_subscriber.py
```

Edit the file robot_subscriber.py using Visual Studio Editor

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String

class RobotSubscriber(Node):
    def __init__(self):
        super().__init__("robot_subscriber")
        self.subscriber_ = self.create_subscription(String, "robot_news",
self.callback_robot_news, 10)
        self.get_logger().info("robot_subscriber Node Started")

    def callback_robot_news(self, msg):
        self.get_logger().info(msg.data)
```

```
def main(args=None):
    rclpy.init(args=args)
    node = RobotSubscriber()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

Edit the setup.py

```
from setuptools import setup
package_name = 'my_package '

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
             'sample = my_package.sample:main',
            "robot_publisher = my_package.robot_publisher:main",
            'robot_subscriber = my_package.robot_subscriber:main'
        ],
    },
)
```

Execute in Terminal #1

```
colcon build --packages-select my_package  --symlink-install
```

Execute in Terminal #2

```
source ~./bashrc
ros2 run my_package robot_publisher
```

Execute in Terminal #3

```
source ~./bashrc
ros2 run my_package robot_subscriber
```

Execute in Terminal #4

```
ros2 node list
ros2 topic list
```

Try: Modify the subscriber code to publish number at 1Hz on the topic /number

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String, Int32
global count

class RobotSubscriber(Node):
    def __init__(self):
        super().__init__("robot_subscriber")
        self.count_ = 0
        self.subscriber_ = self.create_subscription(String, "robot_news",
self.callback_robot_news, 10)
        self.publisher_ = self.create_publisher(Int32, "robot_number", 10)
        self.timer_ = self.create_timer(1, self.send_number)
        self.get_logger().info("robot_subscriber and publisher Node Started")

    def callback_robot_news(self, msg):
        self.get_logger().info(msg.data)

    def send_number(self):
        number = Int32()
        number.data = self.count_
        self.count_ +=1
        self.publisher_.publish(number)

def main(args=None):
    rclpy.init(args=args)
    node = RobotSubscriber()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

Post Lab Exercises - Marks: 4 [CO – 1, LO – 1, 2, 12, PO- 1,2,3, BL – 3,4,5]

Exercise 1: Write a launch file pub_sub.launch.py to run the publisher and subscriber node.

Exercise 2: Create 2 nodes from scratch. First node has 1 publisher, the second has 1 publisher & 1 subscriber.

- The number_publisher node publishes a number on the "/number" topic, with the existing type example_interfaces/msg/Int32.
- The number_counter node subscribes to the "/number" topic. It keeps a counter variable. Every time a new number is received, it's added to the counter. The node also has a publisher on the "/number_count" topic. When the counter is updated, the publisher directly publishes the new value on the topic.

A few hints:

- Check what to put into the example_interfaces/msg/Int32 with the "ros2 interface show" command line tool.
- Use the order as follows: first create the number_publisher node, check that the publisher is working with "ros2 topic". Then create the number_counter, focus on the subscriber. And finally create the last publisher. In the number_counter node, the publisher will publish messages directly from the subscriber callback.

More About Launch File

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        launch_ros.actions.Node(
            package='turtlebot3_teleop',
            executable='teleop_keyboard',
            output='screen'),
    ])
```

```
from launch import LaunchDescription
import launch_ros.actions

def generate_launch_description():
    return LaunchDescription([
        launch_ros.actions.Node(
            package='teleop_twist_keyboard',
            executable='teleop_twist_keyboard',
            output='screen'),
    ])
```

Within the LaunchDescription object, generate a node where you will provide the following parameters:

1. package='package_name' Name of the package that contains the code of the ROS program to execute
2. executable='cpp_executable_name' Name of the cpp executable file that you want to execute
3. output='type_of_output' Through which channel you will print the output of the program

Create Custom Message

To publish the data that contains multiple data types, one can create a new one.
Create a custom interface in a CMake package and then use it in a Python node.
To create a new message, do the following:

1. Create a directory in the src folder
2. Create a directory named **msg** inside your package Inside the directory, create a file named Name_of_message.msg. Modify the CMakeLists.txt file
3. Modify package.xml file
4. Compile and source
5. Use in code

Create an interface to send the Manufacture date of the robot.

Execute in Terminal #1

```
cd ros2_ws/src
ros2 pkg create my_robot_interface
ls
cd my_robot_interface/
rm -rf include/
rm -rf src/
mkdir msg
cd msg
touch ManufactureDate.msg
```

open src with visual studio application:

Enter the following data in ManufactureDate.msg (**It should have the pattern '^[A-Z][A-Za-z0-9]*$')

```
int32 date
string month
int64 year
```

In CmakeLists.txt

Edit two functions inside CMakeLists.txt:

find_package()
This is where all the packages required to COMPILE the messages for the topics, services, and actions go. In package.xml, state them as **build_depend** and **exec_depend**.

```
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)
```

```
rosidl_generate_interfaces()
This function includes all of the messages for this package (in the msg folder) to be
compiled. The function should look similar to the following:
rosidl_generate_interfaces(${PROJECT_NAME}
"msg/ManufactureDate.msg"
)
```

```
cmake_minimum_required(VERSION 3.5)
project(my_robot_interface)

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(my_robot_interface
"msg/ManufactureDate.msg"
)


ament_package()
```

Modify package.xml

Add the following lines to the package.xml

```
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

```
<?xml version="1.0"?>
```

```
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_robot_interface</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="asha@todo.todo">asha</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <build_depend>rosidl_default_generators</build_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Execute in Terminal #1

```
cd ~/ros2_ws
colcon build --packages-select my_robot_interface
cd install/my_robot_interface/lib/python3.8/site-packages/my_robot_interface/msg
```

This executes this bash file that sets, among other things, the newly generated messages created through the colcon build. If you don't do this, it might give you an import error, saying it doesn't find the message generated.

```
source install/setup.bash
ros2 interface show my_robot_interface/msg/ManufactureDate
```

Modify robot_publisher.py to transmit the manufacturing date to the subscriber node

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interface.msg import ManufactureDate

class RobotDatePublisher(Node):
```

```
    def __init__(self):
        super().__init__("robot_date_publisher")
        self.robot_name_="ROBOT"
        self.publisher_ = self.create_publisher(ManufactureDate,
"robot_manufacturing_date", 10)
        self.timer_ = self.create_timer(0.5, self.publish_news)
        self.get_logger().info("Node Started")


    def publish_news(self):
        msg = ManufactureDate()
        msg.date = 12
        msg.month = "March"
        msg.year = 2022
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = RobotDatePublisher()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Execute in Terminal #1

```
colcon build --packages-select my_package
```

Execute in Terminal #2

```
ros2 run my_package robot_publisher
```

Execute in Terminal #3

```
ros2 run my_package robot_subscriber
```

Edit package.xml

```
 <depend>rclpy</depend>
 <depend>example_interfaces</depend>
 <depend>my_robot_interface</depend>
```

Edit robot_subscriber.py code

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interface.msg import ManufactureDate

class RobotDateSubscriber(Node):
    def __init__(self):
        super().__init__("robot_date_subscriber")
        self.subscriber_ = self.create_subscription(ManufactureDate,
"robot_manufacturing_date", self.callback_robot_news, 10)
        self.get_logger().info("robot_subscriber Node Started")

    def callback_robot_news(self, msg):
        information ="Manufacturing Date of the ROBOT is " + str(msg.date) + " " +
str(msg.month) + " " + str(msg.year)
        self.get_logger().info(information)


def main(args=None):
    rclpy.init(args=args)
    node = RobotDateSubscriber()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```