Activity 4: Creating a simple publisher (Python)

Open a new terminal and remember to source your Ros2 installation so that ros2 commands will work.

1. Creating a package for publisher & subscriber examples

Navigate into the ‘dev_ws’ directory created in Activity 1a.

```
cd ~/dev_ws
```

Recall that packages should be created in the src directory, not the root of the workspace. So, navigate into ‘dev_ws/src’, and run the package creation command to create a new package:

```
cd ~/dev_ws/src
ros2 pkg create --build-type ament_python py_pubsub
```

Your terminal will return a message verifying the creation of your package ‘py_pubsub’ and all its necessary files and folders.

Navigate into ‘dev_ws/src/py_pubsub/py_pubsub’

```
cd ~/dev_ws/src/py_pubsub/py_pubsub
```

Recall that this directory is a Python package with the same name as the ROS 2 package it’s nested in.

Create and open a new python file named ‘publisher_member_function.py’ with the following command:

```
gedit publisher_member_function.py
```

An empty text editor will pop-up and next will be writing a publisher node.

2. Writing and examining the code

The following steps involves filling up the text editor with the following lines of code required to create a publisher node.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
```

The first lines of code after the comments import rclpy so its Node class can be used.

The next statement imports the built-in string message type that the node uses to structure the data that it passes on the topic.

These lines represent the node’s dependencies. Recall that dependencies have to be added to package.xml, which you’ll do in the next section.

Next, the MinimalPublisher class is created, which inherits from (or is a subclass of) Node.

```
class MinimalPublisher(Node):
```

Following is the definition of the class’s constructor. super().__init__ calls the Node class’s constructor and gives it your node name, in this case minimal_publisher.

```
def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0
```

Timer_callback creates a message with the counter value appended, and publishes it to the console with get_logger().info.

```
def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1
```

Lastly, the main function is defined.

```
def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()


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

First the rclpy library is initialized, then the node is created, and then it "spins" the node so its callbacks are called.

3. Add dependencies

Navigate one level back to the dev_ws/src/py_pubsub directory, where the setup.py, setup.cfg, and package.xml files have been created for you.

Open package.xml with your text editor. Make sure to fill in the <description>, <maintainer> and <license> tags:

```
<description>Examples of minimal publisher/subscriber using rclpy</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

After the lines above, add the following dependencies corresponding to your node's import statements:

```
<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

This declares the package needs rclpy and std_msgs when its code is executed.

Make sure to save the file.

4. Add an entry point

Open the setup.py file. Again, match the maintainer, maintainer_email, description and license fields to your package.xml:

```
maintainer='YourName',
maintainer_email='you@email.com',
description='Examples of minimal publisher/subscriber using rclpy',
license='Apache License 2.0',
```

Add the following line within the console_scripts brackets of the entry_points field:

```
entry_points={
        'console_scripts': [
                'talker = py_pubsub.publisher_member_function:main',
        ],
},
```

Don’t forget to save.

5. Check setup.cfg

The contents of the setup.cfg file should be correctly populated automatically, like so:

```
[develop]
script-dir=$base/lib/py_pubsub
[install]
install-scripts=$base/lib/py_pubsub
```

This is simply telling setuptools to put your executables in lib, because ros2 run will look for them there.

You could build your package now, source the local setup files, and run it, but let’s create the subscriber node first so you can see the full system at work.

6. Write the subscriber node

Return to dev_ws/src/py_pubsub/py_pubsub to create the next node.
Enter the following code in your terminal:

```
wget
https://raw.githubusercontent.com/ros2/examples/foxy/rclpy/topics/minimal_subscriber/examples_rclpy_minimal_subscriber/subscriber_member_function.py
```

Now the directory should have these files:

```
__init__.py  publisher_member_function.py  subscriber_member_function.py
```

7. Examine the code

Open the subscriber_member_function.py with your text editor.

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String

class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)

def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()
```

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

The subscriber node’s code is nearly identical to the publisher’s. The constructor creates a subscriber with the same arguments as the publisher. Recall from the topics tutorial that the topic name and message type used by the publisher and subscriber must match to allow them to communicate.

```
self.subscription = self.create_subscription(
    String,
    'topic',
    self.listener_callback,
    10)
```

The subscriber’s constructor and callback don’t include any timer definition, because it doesn’t need one. Its callback gets called as soon as it receives a message.

The callback definition simply prints an info message to the console, along with the data it received. Recall that the publisher defines msg.data = 'Hello World: %d' % self.i

```
def listener_callback(self, msg):
    self.get_logger().info('I heard: "%s"' % msg.data)
```

The main definition is almost exactly the same, replacing the creation and spinning of the publisher with the subscriber.

```
minimal_subscriber = MinimalSubscriber()

rclpy.spin(minimal_subscriber)
```

Since this node has the same dependencies as the publisher, there’s nothing new to add to package.xml. The setup.cfg file can also remain untouched.

8. Add an entry point

Reopen setup.py and add the entry point for the subscriber node below the publisher’s entry point. The entry_points field should now look like this:

```
entry_points={
        'console_scripts': [
                'talker = py_pubsub.publisher_member_function:main',
                'listener = py_pubsub.subscriber_member_function:main',
        ],
},
```

Make sure to save the file, and then your pub/sub system should be ready for use.

9. Build and run

You likely already have the rclpy and std_msgs packages installed as part of your ROS 2 system. It’s good practice to run rosdep in the root of your workspace (dev_ws) to check for missing dependencies before building:

```
rosdep install -i --from-path src --rosdistro <distro> -y
```

Still in the root of your workspace, dev_ws, build your new package:

```
colcon build --packages-select py_pubsub
```

Open a new terminal, navigate to dev_ws, and source the setup files:

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

Now run the talker node:

```
ros2 run py_pubsub talker
```

The terminal should start publishing info messages every 0.5 seconds, like so:

```
[INFO] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [minimal_publisher]: Publishing: "Hello World: 1"
[INFO] [minimal_publisher]: Publishing: "Hello World: 2"
[INFO] [minimal_publisher]: Publishing: "Hello World: 3"
[INFO] [minimal_publisher]: Publishing: "Hello World: 4"
...
```

Open another terminal, source the setup files from inside dev_ws again, and then start the listener node:

```
ros2 run py_pubsub listener
```

The listener will start printing messages to the console, starting at whatever message count the publisher is on at that time, like so:

```
[INFO] [minimal_subscriber]: I heard: "Hello World: 10"
[INFO] [minimal_subscriber]: I heard: "Hello World: 11"
[INFO] [minimal_subscriber]: I heard: "Hello World: 12"
[INFO] [minimal_subscriber]: I heard: "Hello World: 13"
[INFO] [minimal_subscriber]: I heard: "Hello World: 14"
```

Enter Ctrl+C in each terminal to stop the nodes from spinning.