Smart Mobility Engineering Lab (IGS3231)

Jump Together, Fly Farther!

인하대학교 국제학부

Week 5

ISE Department

Prof. Mehdi Pirahandeh

Content

인하대학교 INHA UNIVERSITY

- Writing a simple publisher and subscriber
- Creating custom *msg* and *srv* files
- Implementing custom interfaces
- Using parameters in a class
- Using ros2doctor to identify issues
- Activity session

Introduction to Course

인하대학교 INHA UNIVERSITY

ROS 2 Documentation: Foxy

Search docs

- Installation
- Distributions
- Tutorials
- How-to Guides
- Concepts
- Contact
- The ROS 2 Project
- Related Projects
- Glossary
- Citations

» ROS 2 Documentation Edit on GitHub

You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Humble.

ROS 2 Documentation

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

This site contains the documentation for ROS 2. If you are looking for ROS 1 documentation, check out the ROS wiki.

If you use ROS 2 in your work, please see Citations to cite ROS 2.

Getting started

- Installation
  - Instructions to set up ROS 2 for the first time
- Tutorials
  - The best place to start for new users!
  - Hands-on sample projects that help you build a progression of necessary skills
- How-to Guides

http://docs.ros.org/en/foxy/index.html

Writing a simple publisher and subscriber

Writing a simple publisher and subscriber

- In this practice, you will create nodes that pass information in the form of string messages to each other over a topic.
- The example used here is a simple “talker” and “listener” system.
- One node publishes data and the other subscribes to the topic so it can receive that data.

Writing a simple publisher and subscriber

- **Step 1: Create a package**
  - ros2 pkg create --build-type ament_python py_pubsub
- **Step 2: Write the publisher node**
  - Download the example talker code by entering the following command
    - wget https://raw.githubusercontent.com/ros2/examples/rolling/rclpy/topics/minimal_publisher/examples_rclpy_minimal_publisher/publisher_member_function.py
  - Now there will be a new file named publisher_member_function.py adjacent to __init__.py.

Writing a simple publisher and subscriber

인하대학교 INHA UNIVERSITY

Step 2

Write the publisher node

```python
import rclpy # import rclpy library so its Node class can be used
from rclpy.node import Node

#Imports the built-in string message type that the node uses to structure the data that it passes on the topic.
from std_msgs.msg import String

#Next, the MinimalPublisher class is created, which inherits from (or is a subclass of) Node.
class MinimalPublisher(Node):
#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')
        #create_publisher declares that the node publishes messages of type String
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        #a timer is created with a callback to execute every 0.5 seconds.
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        #self.i is a counter used in the 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):
    #First the rclpy library is initialized, then the node is created
    #and then it "spins" the node so its callbacks are called.
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    minimal_publisher.destroy_node()
    rclpy.shutdown()


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

Writing a simple publisher and subscriber

- **Step 2.2: Write the publisher node**
  - ***Add Dependencies***

- Navigate one level back to the ros2_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.

Writing a simple publisher and subscriber

- **Step 2.2: Write the publisher node**
  - ***Add Dependencies***

- 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>
```

Writing a simple publisher and subscriber

인하대학교 INHA UNIVERSITY

- **Step 2.2: Write the publisher node**
  - ***Add Dependencies***

```
<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.

Writing a simple publisher and subscriber

- **Step 2.3: Write the publisher node**
  - ***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',
```

Writing a simple publisher and subscriber

- **Step 2.3: Write the publisher node**
  - ***Add an entry point***

- 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',
        ],
},
```

Writing a simple publisher and subscriber

- **Step 3: Write the subscriber node**

- Return to *ros2_ws/src/py_pubsub/py_pubsub* to create the next node.

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

Writing a simple publisher and subscriber

- **Step 3: Write the subscriber n ode**

- Enter the following code in your terminal:

```python
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
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


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

Writing a simple publisher and subscriber

- **Step 3.2: Write the publisher node**
  - ***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',
        ],
},
```

Writing a simple publisher and subscriber

- **Step 4: Build and Run**
  - 1: rosdep install -i --from-path src --rosdistro rolling –y
  - 2: colcon build --packages-select py_pubsub
  - 3: . install/setup.bash
  - 4: ros2 run py_pubsub talker
  - 5: ros2 run py_pubsub listener

Writing a simple service and client

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 1: Create a package**
  - Run the following code in your terminal:

```
ros2 pkg create --build-type ament_python py_srvcli --dependencies rclpy example_interfaces
```

Writing a simple service and client

- **Step 1: Update the package**
  - Make sure to add the description, maintainer email and name, and license information to package.xml

```
<description>Python client server tutorial</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 1: Update the package**
  - Add the same information to the setup.py file for the maintainer, maintainer_email, description and license fields:

```
maintainer='Your Name',
maintainer_email='you@email.com',
description='Python client server tutorial',
license='Apache License 2.0',
```

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 2: Enter the following code in your terminal:**

```python
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node


class MinimalService(Node):

    def __init__(self):
        super().__init__('minimal_service')
        self.srv = self.create_service(AddTwoInts, 'add_two_ints', self.add_two_ints_callback)

    def add_two_ints_callback(self, request, response):
        response.sum = request.a + request.b
        self.get_logger().info('Incoming request\na: %d b: %d' % (request.a, request.b))

        return response


def main():
    rclpy.init()

    minimal_service = MinimalService()

    rclpy.spin(minimal_service)

    rclpy.shutdown()


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

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 2.1: Add an entry point**

```
entry_points={
    'console_scripts': [
        'service = py_srvcli.service_member_function:main',
        'client = py_srvcli.client_member_function:main',
    ],
},
```

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 3: Enter the following client code in your terminal:**

```python
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node


class MinimalClientAsync(Node):

    def __init__(self):
        super().__init__('minimal_client_async')
        self.cli = self.create_client(AddTwoInts, 'add_two_ints')
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')
        self.req = AddTwoInts.Request()

    def send_request(self, a, b):
        self.req.a = a
        self.req.b = b
        self.future = self.cli.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()


def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    response = minimal_client.send_request(int(sys.argv[1]), int(sys.argv[2]))
    minimal_client.get_logger().info(
        'Result of add_two_ints: for %d + %d = %d' %
        (int(sys.argv[1]), int(sys.argv[2]), response.sum))

    minimal_client.destroy_node()
    rclpy.shutdown()


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

Writing a simple service and client

인하대학교 INHA UNIVERSITY

- **Step 4: Build and run**

- Check for missing dependencies before building:
  - 1: rosdep install -i --from-path src --rosdistro rolling –y

- Navigate back to the root of your workspace, ros2_ws, and build your n ew package:
  - 2: colcon build --packages-select py_srvcli

- Open a new terminal, navigate to ros2_ws, and source the setup files:
  - 3: . install/setup.bash

Creating custom msg and srv files

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- Goal: Define custom interface .msg and .srv files, and then use them with Python nodes.
- In previous cases you learned how to use predefined interfaces for writing simple publisher/subscriber and service/client using Python.
- But sometimes it is required to write custom methods as well.
- In this lesson we will learn how to build custom interface definition s.

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 1: Create a new package**

  ros2 pkg create --build-type ament_cmake tutorial_interfaces

  – Create separate directories for .msg and .srv
    - mkdir ms and mkdir srv

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 2: Create custom definitions**
  - 2.1 msg definitions
    - Make a new file with one line of code ------ int64 num
    - Make a new file called Sphere.msg -- geometry_msgs/Point center and float64 radius

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 2: Create custom definitions**
  - 2.1 srv definition
    - Back in the tutorial_interfaces/srv directory you just created.
    - Make a new file called AddThreeInts.srv with the following request and response structure: int64 a int64 b int64 --- int64 sum.
    - This is your customer service that requests 3 integers.

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 3: Cmakelists.txt**

  – To convert the interfaces, you defined into language-specific c ode(like C++ and Python) so that they can be used in those lan guages, add the following lines to CMakeLists.txt:

```
find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "msg/Sphere.msg"
  "srv/AddThreeInts.srv"
  DEPENDENCIES geometry_msgs # Add packages that above messages depend on, in this case
)
```

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 4: package.xml**
  - Add the following lines to package.xml

```
<depend>geometry_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>
```

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 5: Build the tutorial_interfaces package**
  - Now that all the parts of your custom interfaces package are i n place, you can build the package.

  - In the root of your workspace (~/ros2_ws), run the following co mmand:
    - colcon build --packages-select tutorial_interfaces

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 6: Confirm msg and srv creation**
  - In a new terminal, run the following command from within yo ur workspace (ros2_ws) to source it:
    - . install/setup.bash
    - To confirm the creation of interface run
      - “ros2 interface show tutorial_interfaces/msg/Num”

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 7: Test the new interfaces**
  - For this step you can use the packages you created in previous tutorials.
  - A few simple modifications to the nodes, CMakeLists and package files will allow you to use your new interfaces.

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 7.1 Testing Num.msg with pub/sub**
  - With some slight modifications to the publisher/subscriber package created in a previous tutorial you can see Num.msg in action.
  - Since you'll be changing the standard string msg to a numerical one, the output will be slightly different.

Creating custom msg and srv files

인하대학교 INHA UNIVERSITY

- **Step 7.1 Testing Num.msg with pub/sub**
  - With some slight modifications to the publisher/subscriber package created in a previous tutorial you can see Num.msg in action.
  - Since you'll be changing the standard string msg to a numerical one, the output will be slightly different.

ACTIVITY SESSION

ROS2 Activity Session

인하대학교 INHA UNIVERSITY

- **Individual Task**

  – Review and practice all the commands in your VM machine.

  - Submit your work via the I-class discussion forum.

    - A GitHub link for the task
    - Write an analysis of your commands
    - Deadline for each activity is the next week Monday at 12:00 PM.

Brief break (if on schedule)

Prof. Mehdi Pirahandeh E-mail : mehdi@inha.ac.kr