

ROS TUTORIAL

INTRODUCTION: ROS stands for Robot Operating System. It is an open-source software framework for robotics. ROS provides a common platform for developing robot software, making it easier for researchers and developers to share code and collaborate on projects. ROS is used by a wide range of organizations, from academic research labs to industrial companies.

ROS is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

Here are some of the benefits of using ROS:

- **Open source:** ROS is free and open source software, which means that it is available to everyone to use and modify. This makes it a great platform for collaboration and innovation.
- **Cross-platform:** ROS can be used on a variety of platforms, including Linux, macOS, and Windows. This makes it a great choice for developing robots that need to be portable.
- **Modular:** ROS is a modular framework, which means that it is easy to add new features and functionality. This makes it a great choice for developing complex robots with a wide range of capabilities.
- **Active community:** ROS has a large and active community of users and developers. This means that there is always help available if you need it.

If you are interested in developing robots, ROS is a great place to start. It is a powerful and flexible platform that can be used to build a wide variety of robots.

Prerequisites: To make the most of this tutorial, it is recommended that you have basic knowledge of ROS, including installing ROS on your system. If you are new to ROS, you can refer to the ROS documentation (link provided in the resources section) for installation instructions and beginner-level tutorials.

QUICK-START: Here are the steps on how to get started with ROS:

Choose a platform. ROS can be installed on a variety of platforms, including Linux, macOS, and Windows. The official ROS website has installation instructions for each platform.

Install ROS. Once you have chosen a platform, you can install ROS. The installation process is different for each platform, so be sure to follow the instructions carefully.

Setting up the environment: If you just installed ROS from `apt` on Ubuntu then you will have to source the `setup.*sh` files by running the following line of command in the terminal:

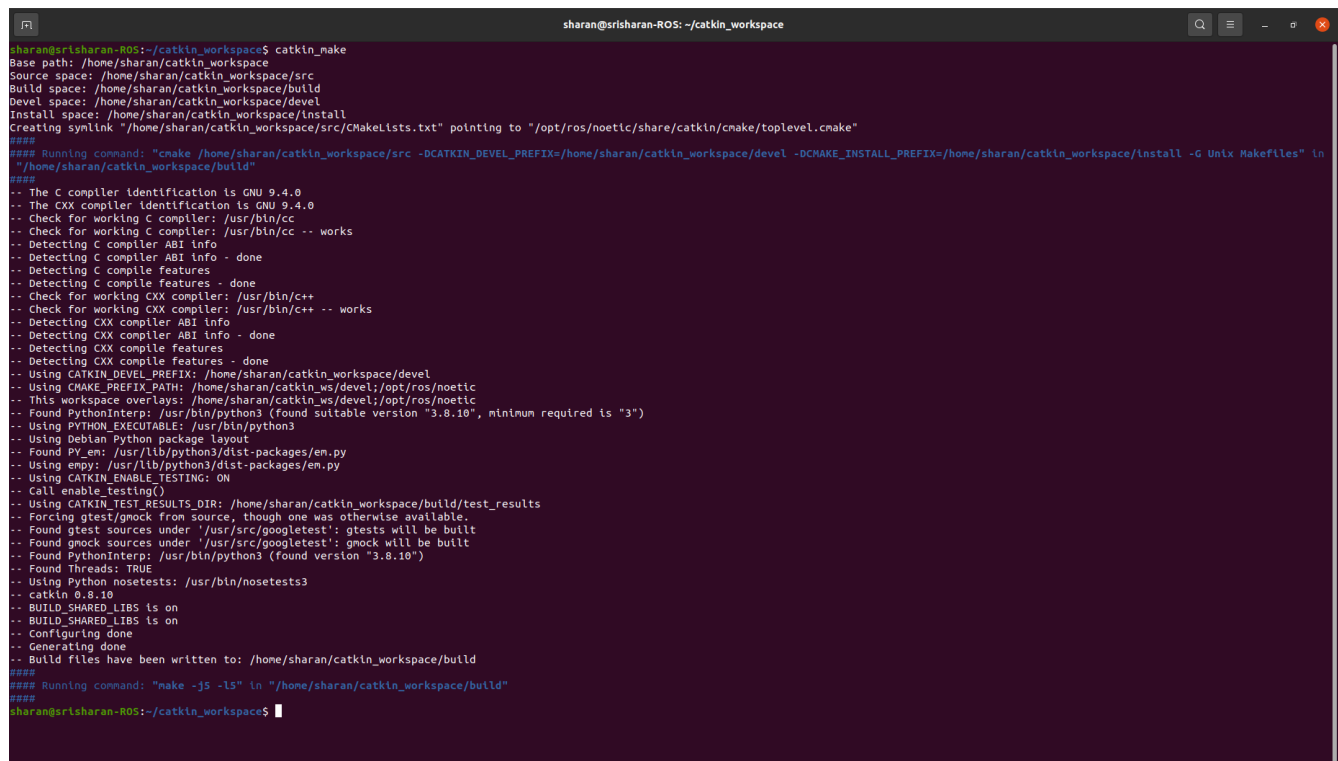
```
source /opt/ros/noetic/setup.bash
```

Create a ROS workspace. A ROS workspace is a directory that contains all of your ROS projects. To create a ROS workspace, run the following command in a terminal:

```
mkdir -p ~/catkin_ws/src
```

The above line of code creates a directory named ‘catkin_ws’ and a folder inside it named ‘src’
Now to build this directory into a catkin workspace run the following line in the command terminal:

```
cd ~/catkin_ws/  
catkin_make
```



```
sharan@srisharan-ROS:~/catkin_workspace$ catkin_make
Base path: /home/sharan/catkin_workspace
Source space: /home/sharan/catkin_workspace/src
Build space: /home/sharan/catkin_workspace/build
Devel space: /home/sharan/catkin_workspace/devel
Install space: /home/sharan/catkin_workspace/install
Creating symlink "/home/sharan/catkin_workspace/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/sharan/catkin_workspace/src -DCATKIN_DEVEL_PREFIX=/home/sharan/catkin_workspace/devel -DCMAKE_INSTALL_PREFIX=/home/sharan/catkin_workspace/install -G Unix Makefiles" in
"/home/sharan/catkin_workspace/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/sharan/catkin_workspace/devel
-- Using CMAKE_PREFIX_PATH: /home/sharan/catkin_ws/devel:/opt/ros/noetic
-- This workspace overlays: /home/sharan/catkin_ws/devel:/opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Found PY_en: /usr/lib/python3/dist-packages/em.py
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/sharan/catkin_workspace/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Found Threads: TRUE
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sharan/catkin_workspace/build
####
#### Running command: "make -j5 -l5" in "/home/sharan/catkin_workspace/build"
####
sharan@srisharan-ROS:~/catkin_workspace$
```

The above line of code takes you into the catkin_ws directory and executes the command catkin_make. catkin_make is a command-line tool that builds ROS packages. It is a wrapper around the CMake build system, and it takes care of all of the details of building ROS packages, such as finding dependencies and generating build files. Once you run the catkin_make command you should see something like the figure displayed on your command terminal.

Now, if you go into the catkin_ws director you will be able to see two more folders generated along with the “src” i.e. “devel” and “build”.

Setting up a ROS package: In this section, we will walk you through the process of setting up a ROS package, which will serve as the foundation for your ROS development projects. A ROS package is a directory that contains ROS-specific files and configurations. Let's get started!

Run the following command to create a new package named "beginner_package":

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg beginner_package rospy std_msgs
```

The above command creates a package named "beginner_package" and includes the "rospy" dependency, which is necessary for Python-based ROS nodes. std_msgs is a ROS package that provides a set of common message types. These message types are used to represent data that is commonly used in robotics, such as geometry, time, and sensor data. Feel free to replace "beginner_package" with your desired package name. You can also add other dependencies like roscpp if you are comfortable with C++ rather than python but for this tutorial let us limit ourselves to Python based ROS nodes.

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Once the package is created, always run the below command-line to source the package.

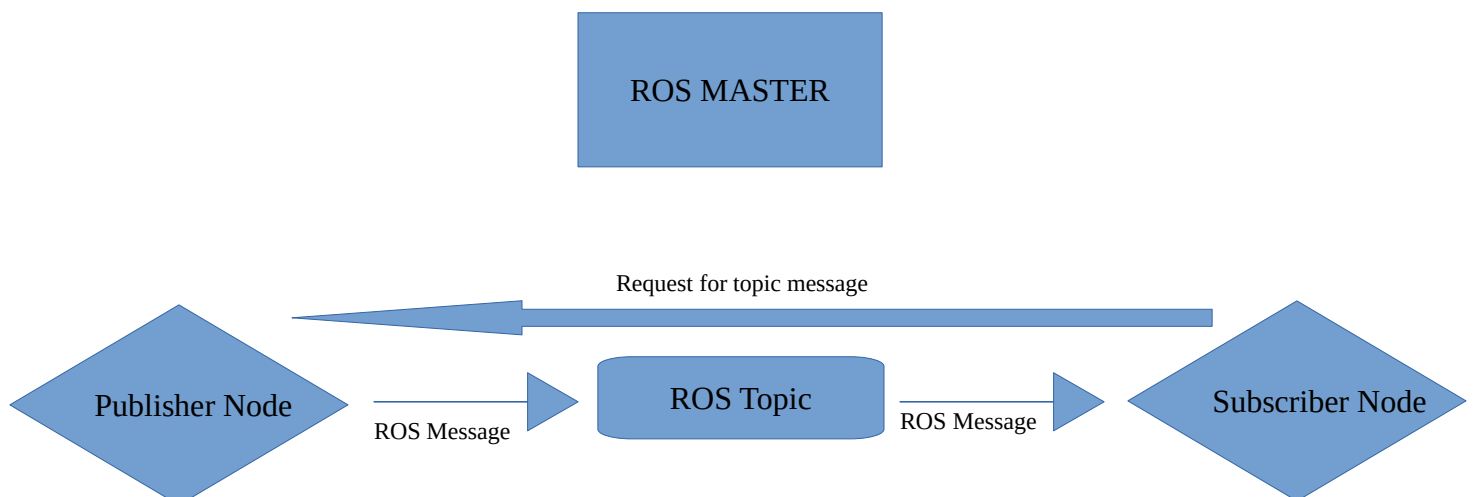
```
source /opt/ros/noetic/setup.bash
```

Sourcing has to be done every time you create a new package. By sourcing the workspace, you make the package and its dependencies available to the ROS environment.

Now run the catkin_make command again to build the package just like how you built the workspace.

```
cd ~/catkin_ws/  
catkin_make
```

ROS Publishing and Subscribing: Before we dive into the practical implementation, let's briefly understand the concepts of publishing and subscribing in ROS. ROS follows a publish-subscribe messaging model, where nodes communicate with each other by publishing and subscribing to topics. Publishers send messages to a specific topic, and subscribers receive those messages from the same topic. This decoupled communication mechanism allows for modular and flexible robot software development.



Creating a Publisher Node: Create a new Python script named “talker.py” inside the “src” folder of the package that you have created. This script will publish messages to a topic. Here's an example of a simple publisher node that publishes a string message:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello this message is from the publisher %s" %
rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

The first line of code is a shebang line. It tells the operating system which interpreter to use to run the script.

The next two lines import the ROS libraries `rospy` and `std_msgs`.

The function called `talker()` creates a publisher object, initializes a ROS node, creates a rate object, and publishes a message on the topic `chatter` at a rate of 10Hz.

The last 4 lines of the code, checks if the script is being run as the main script. If it is, it calls the `talker()` function. If the script is not being run as the main script, it does nothing.

You can use this sample code to customize your publisher node and publish anything that you desire. But the architecture of the Publisher node code remains the same.

Creating a Subscriber Node: In the same folder create another Python script named `listener.py`. This script will subscribe to the topic and receive the messages published by the publisher node. Here's an example of a simple subscriber node:

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard from the talker_listener %s', data.data)

def listener():

    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber('chatter_2', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

The first line of code is a shebang line. It tells the operating system which interpreter to use to run the script.

The next two lines import the ROS libraries `rospy` and `std_msgs`.

The `def callback(data):` function is a callback function. It is called whenever a message is received on the topic "chatter_2".

The `rospy.loginfo()` function logs the message to the ROS log.

The `listener()` function initializes a ROS node, creates a subscriber object, and subscribes to the topic `chatter_2`.

`rospy.init_node()` function initializes a ROS node. The name of the node is `subscriber_node`.

The `rospy.Subscriber()` function creates a subscriber object. The subscriber object is used to subscribe to a topic. The topic name is "chatter_2". The message type is `String`. The callback function is `callback`. The `rospy.spin()` function blocks the main thread of the node and waits for messages to be received on the subscribed topics.

You can use this sample code to customize your subscriber node and publish anything that you desire. But the architecture of the Subscriber node code remains the same.

Creating a node that is both a publisher and a subscriber: In the same folder create another Python script named `talker_listener.py`. This script will subscribe to the topic and receive the messages published by the publisher node and also publish a message to a new topic. Here's an example of a simple subscriber node:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String, Int32MultiArray

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard from the talker %s',
data.data)

def listener_talker():
    rospy.init_node('talker_listener', anonymous=True)

    rospy.Subscriber('chatter', String, callback)

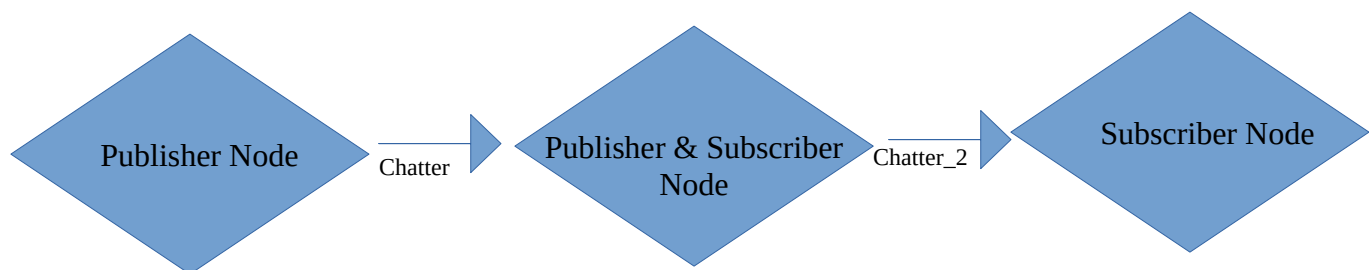
    # spin() simply keeps python from exiting until this node is stopped
    #rospy.spin()

    pub = rospy.Publisher('chatter_2', String, queue_size=10)

    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "This message is from the publisher_subscriber %s" %
rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        listener_talker()
    except rospy.ROSInterruptException:
        pass
```

This is a node that works as both a subscriber node and publisher node.



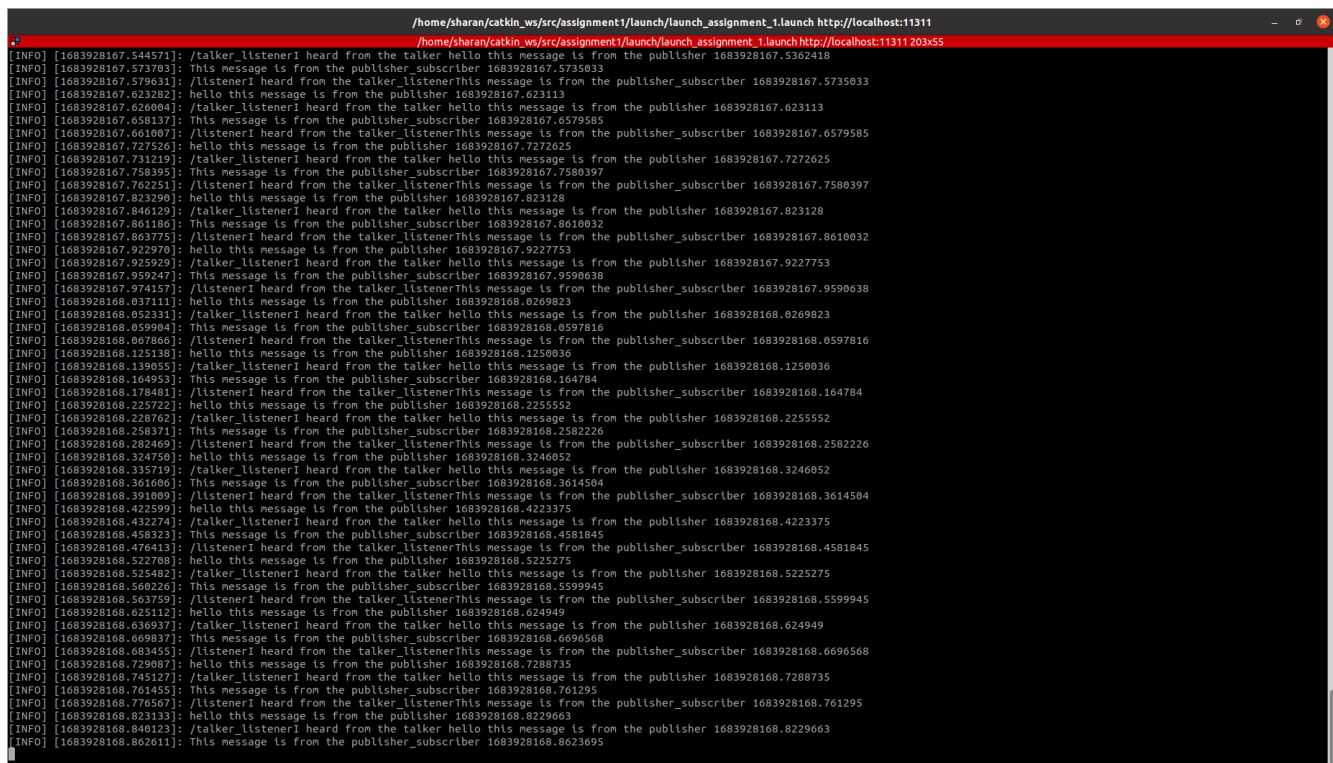
Launch File: Now let us create a launch file to run all of these node simultaneously and see it working. A launch file in ROS is a text file that describes how to start up a ROS application. It can be used to start up multiple nodes, set parameters, and create a namespace. Below is the launch file code for running the above created nodes. Create an xml file name `launch_assignment.launch` with the below code:

```
<launch>
  <node pkg="assignment1" type="talker.py" name="talker"
output="screen"></node>
  <node pkg="assignment1" type="talker_listener.py"
name="talker_listener" output="screen"></node>
  <node pkg="assignment1" type="listener.py" name="listener"
output="screen"></node>
</launch>
```

Now to run this launch file run the below line of code.

```
roslaunch beginner_tutorial
```

You should see something like below when you run the code.



```
/home/sharan/catkin_ws/src/assignment1/launch/launch_assignment_1.launch http://localhost:11311
/home/sharan/catkin_ws/src/assignment1/launch/launch_assignment_1.launch http://localhost:11311 20x55
[INFO] [1683928167.544571]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928167.5362418
[INFO] [1683928167.573703]: This message is from the publisher_subscriber 1683928167.5735033
[INFO] [1683928167.5796931]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928167.5735033
[INFO] [1683928167.623282]: hello this message is from the publisher 1683928167.623113
[INFO] [1683928167.626084]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928167.623113
[INFO] [1683928167.658137]: This message is from the publisher_subscriber 1683928167.6579585
[INFO] [1683928167.661007]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928167.6579585
[INFO] [1683928167.727526]: hello this message is from the publisher 1683928167.7272625
[INFO] [1683928167.732249]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928167.7272625
[INFO] [1683928167.758395]: This message is from the publisher_subscriber 1683928167.7580397
[INFO] [1683928167.762251]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928167.7580397
[INFO] [1683928167.823290]: hello this message is from the publisher 1683928167.823128
[INFO] [1683928167.846129]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928167.823128
[INFO] [1683928167.861186]: This message is from the publisher_subscriber 1683928167.8610032
[INFO] [1683928167.863775]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928167.8610032
[INFO] [1683928167.922970]: hello this message is from the publisher 1683928167.9227753
[INFO] [1683928167.925929]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928167.9227753
[INFO] [1683928167.959247]: This message is from the publisher_subscriber 1683928167.9590638
[INFO] [1683928167.974157]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928167.9590638
[INFO] [1683928168.037111]: hello this message is from the publisher 1683928168.0269823
[INFO] [1683928168.052331]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.0269823
[INFO] [1683928168.059984]: This message is from the publisher_subscriber 1683928168.0597816
[INFO] [1683928168.067866]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.0597816
[INFO] [1683928168.125133]: hello this message is from the publisher 1683928168.1250036
[INFO] [1683928168.139055]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.1250036
[INFO] [1683928168.164953]: This message is from the publisher_subscriber 1683928168.164784
[INFO] [1683928168.178481]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.164784
[INFO] [1683928168.225722]: hello this message is from the publisher 1683928168.2255552
[INFO] [1683928168.228762]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.2255552
[INFO] [1683928168.258371]: This message is from the publisher_subscriber 1683928168.2582226
[INFO] [1683928168.282469]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.2582226
[INFO] [1683928168.324750]: hello this message is from the publisher 1683928168.3246052
[INFO] [1683928168.335719]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.3246052
[INFO] [1683928168.361080]: This message is from the publisher_subscriber 1683928168.3614504
[INFO] [1683928168.391089]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.3614504
[INFO] [1683928168.422599]: hello this message is from the publisher 1683928168.4223375
[INFO] [1683928168.432274]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.4223375
[INFO] [1683928168.458323]: This message is from the publisher_subscriber 1683928168.4581845
[INFO] [1683928168.476413]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.4581845
[INFO] [1683928168.522780]: hello this message is from the publisher 1683928168.5225275
[INFO] [1683928168.525482]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.5225275
[INFO] [1683928168.560226]: This message is from the publisher_subscriber 1683928168.5599945
[INFO] [1683928168.563759]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.5599945
[INFO] [1683928168.625112]: hello this message is from the publisher 1683928168.624949
[INFO] [1683928168.636937]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.624949
[INFO] [1683928168.669837]: This message is from the publisher_subscriber 1683928168.6696568
[INFO] [1683928168.683455]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.6696568
[INFO] [1683928168.729087]: hello this message is from the publisher 1683928168.7288735
[INFO] [1683928168.745127]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.7288735
[INFO] [1683928168.761455]: This message is from the publisher_subscriber 1683928168.761295
[INFO] [1683928168.776507]: /listenerI heard from the talker_listenerThis message is from the publisher_subscriber 1683928168.761295
[INFO] [1683928168.823133]: hello this message is from the publisher 1683928168.8229663
[INFO] [1683928168.840123]: /talker_listenerI heard from the talker hello this message is from the publisher 1683928168.8229663
[INFO] [1683928168.862611]: This message is from the publisher_subscriber 1683928168.8623695
```

CONCLUSION:

Congratulations! You have successfully learned the basics of publishing and subscribing in ROS. By following this tutorial, you have created a publisher node that publishes messages to a topic and a subscriber node that receives those messages. This fundamental communication mechanism forms the backbone of ROS applications, allowing different nodes to exchange data seamlessly. Throughout this tutorial, you have gained insights into the ROS publish-subscribe model, set up a ROS package, created a publisher node, and created a subscriber node. You have also tested the publisher and subscriber to verify the communication between them.

Now that you have a solid understanding of ROS publishing and subscribing, you can explore more advanced topics such as message types, custom message creation, and working with multiple publishers and subscribers.

Feel free to experiment with different message types, publish data from sensors or robots, and build more complex ROS systems by connecting nodes with various functionalities.

Remember, ROS provides a vast ecosystem of libraries and tools to aid in robot development, so make sure to explore the official ROS documentation and community resources to expand your knowledge and skills further.

Keep coding and have fun building awesome ROS-powered robots!

Resources:

- ROS Documentation: <http://wiki.ros.org/>
- ROS Wiki - Publisher and Subscriber:
<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>