Note: This tutorial assumes that you have completed the previous tutorials: creating a ROS msg and srv (/ROS/Tutorials/CreatingMsgAndSrv).

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Writing a Simple Publisher and Subscriber (Python)

Description: This tutorial covers how to write a publisher and subscriber node in python.

Tutorial Level: BEGINNER

Next Tutorial: Examining the simple publisher and subscriber (/ROS/Tutorials /ExaminingPublisherSubscriber)

catkin | rosbuild

Sommaire

1. Writing the Publisher Node

"Node" is the ROS term for an executable that is connected to the ROS network. Here we'll create the publisher ("talker") node which will continually broadcast a message.

Change directory into the beginner_tutorials package, you created in the earlier tutorial, creating a package (/ROS/Tutorials/CreatingPackage):

```
$ roscd beginner_tutorials
```

1.1 The Code

First lets create a 'scripts' folder to store our Python scripts in:

```
$ mkdir scripts
$ cd scripts
```

Then download the example script talker.py (https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/talker.py) to your new `scripts` directory and make it executable:

```
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/
001_talker_listener/talker.py
$ chmod +x talker.py
```

You can view and edit the file with `$ rosed beginner_tutorials talker.py` or just look below.

Afficher/masquer les numéros de lignes

```
#!/usr/bin/env python
# license removed for brevity
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 world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

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

1.2 The Code Explained

Now, let's break the code down.

Afficher/masquer les numéros de lignes

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

Every Python ROS Node (/Nodes) will have this declaration at the top. The first line makes sure your script is executed as a Python script.

Afficher/masquer les numéros de lignes

```
import rospy
from std_msgs.msg import String
```

You need to import `rospy` if you are writing a ROS Node (/Nodes). The `std_msgs.msg` import is so that we can reuse the `std_msgs/String` message type (a simple string container) for publishing.

Afficher/masquer les numéros de lignes

```
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
```

This section of code defines the talker's interface to the rest of ROS. `pub = rospy.Publisher("chatter", String, queue_size=10)` declares that your node is publishing to the `chatter` topic using the message type `String`. `String` here is actually the class `std_msgs.msg.String`. The `queue_size` argument is **New in ROS hydro** and limits the amount of queued messages if any subscriber is not receiving them fast enough. In older ROS distributions just omit the argument.

The next line, `rospy.init_node(NAME, ...)`, is very important as it tells rospy the name of your node -- until rospy has this information, it cannot start communicating with the ROS Master (/Master). In this case, your node will take on the name `talker`. NOTE: the name must be a base name (/Names), i.e. it cannot contain any slashes `"/"`.

`anonymous = True` ensures that your node has a unique name by adding random numbers to the end of `NAME`. Refer to Initialization and Shutdown - Initializing your ROS Node (/rospy/Overview /Initialization%20and%20Shutdown#Initializing_your_ROS_Node) in the `rospy` documentation for more information about node initialization options.

Afficher/masquer les numéros de lignes

```
    rate = rospy.Rate(10) # 10hz
```

This line creates a `Rate` object `rate`. With the help of its method `sleep()`, it offers a convenient way for looping at the desired rate. With its argument of `10`, we should expect to go through the loop 10 times per second (as long as our processing time does not exceed 1/10th of a second!)

Afficher/masquer les numéros de lignes

```
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()
```

This loop is a fairly standard rospy construct: checking the `rospy.is_shutdown()` flag and then doing work. You have to check `is_shutdown()` to check if your program should exit (e.g. if there is a `Ctrl-C` or otherwise). In this case, the "work" is a call to `pub.publish(hello_str)` that publishes a string to our `chatter` topic. The loop calls `rate.sleep()`, which sleeps just long enough to maintain the desired rate through the loop.

(You may also run across `rospy.sleep()` which is similar to `time.sleep()` except that it works with simulated time as well (see Clock (/Clock)).)

This loop also calls `rospy.loginfo(str)`, which performs triple-duty: the messages get printed to screen, it gets written to the Node's log file, and it gets written to rosout (/rosout). rosout (/rosout) is a handy for debugging: you can pull up messages using rqt_console (/rqt_console) instead of having to find the console window with your Node's output.

`std_msgs.msg.String` is a very simple message type, so you may be wondering what it looks like to publish more complicated types. The general rule of thumb is that *constructor args are in the same order as in the `.msg` file*. You can also pass in no arguments and initialize the fields directly, e.g.

```
msg = String()
msg.data = str
```

or you can initialize some of the fields and leave the rest with default values:

```
String(data=str)
```

You may be wondering about the last little bit:

Afficher/masquer les numéros de lignes

```
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

In addition to the standard Python `__main__` check, this catches a `rospy.ROSInterruptException` exception, which can be thrown by `rospy.sleep()` and `rospy.Rate.sleep()` methods when `Ctrl-C` is pressed or your Node is otherwise shutdown. The reason this exception is raised is so that you don't accidentally continue executing code after the `sleep()`.

Now we need to write a node to receive the messages.

2. Writing the Subscriber Node

2.1 The Code

Download the listener.py (https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/listener.py) file into your scripts directory:

```
$ roscd beginner_tutorials/scripts/
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/
001_talker_listener/listener.py
```

The file contents look close to:

Afficher/masquer les numéros de lignes

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

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

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

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

Don't forget to make the node executable:

```
$ chmod +x listener.py
```

2.2 The Code Explained

The code for `listener.py` is similar to `talker.py`, except we've introduced a new callback-based mechanism for subscribing to messages.

Afficher/masquer les numéros de lignes

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

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

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

This declares that your node subscribes to the `chatter` topic which is of type `std_msgs.msgs.String`. When new messages are received, `callback` is invoked with the message as the first argument.

We also changed up the call to `rospy.init_node()` somewhat. We've added the `anonymous=True` keyword argument. ROS requires that each node have a unique name. If a node with the same name comes up, it bumps the previous one. This is so that malfunctioning nodes can easily be kicked off the network. The `anonymous=True` flag tells `rospy` to generate a unique name for the node so that you can have multiple `listener.py` nodes run easily.

The final addition, `rospy.spin()` simply keeps your node from exiting until the node has been shutdown. Unlike roscpp, rospy.spin() does not affect the subscriber callback functions, as those have their own threads.

3. Building your nodes

We use CMake as our build system and, yes, you have to use it even for Python nodes. This is to make sure that the autogenerated Python code for messages and services is created.

Go to your catkin workspace and run `catkin_make`:

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

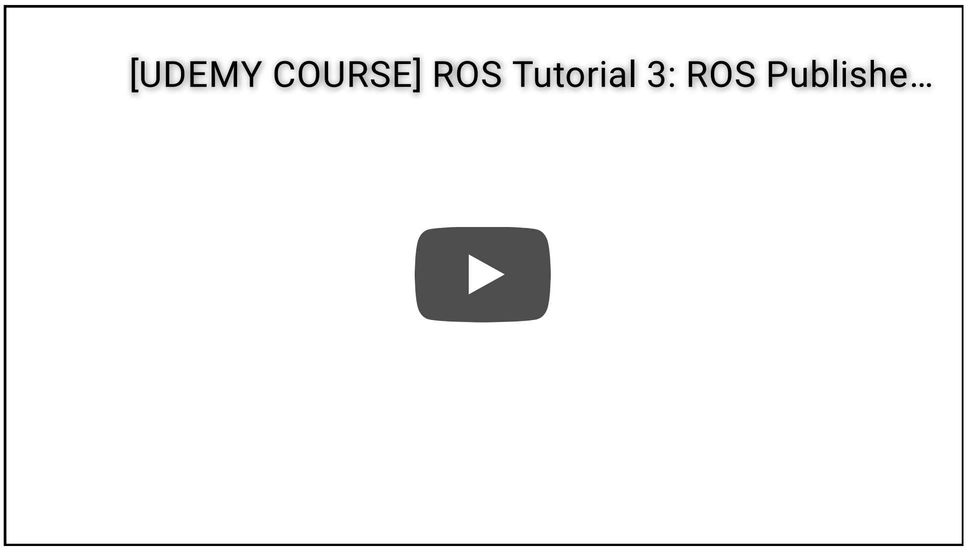
Now that you have written a simple publisher and subscriber, let's examine the simple publisher and subscriber (/ROS/Tutorials/ExaminingPublisherSubscriber).

3.1 Additional Resources

Here are some additional resources contributed by the community.

3.0.1 Video Tutorial

The following video presents a small tutorial explaining how to write and test a publisher and subscriber in ROS with C++ and Python based on the talker/listener example above



Brought to you by: Open Source Robotics Foundation

(http://www.osrfoundation.org)