Note: This tutorial assumes that you have completed the previous tutorials: understanding ROS services and params (/ROS/Tutorials/UnderstandingServicesParams).

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.

1. Writing a Simple Publisher and Subscriber

Description: This demo will walk you through creating a ROS package as well as creating two simple rospy nodes. The "talker" node will broadcast a message on topic "chatter", while the "listener" node will receive and print that message.

Tutorial Level: BEGINNER

Next Tutorial: Writing a simple service and client (/rospy_tutorials/Tutorials/WritingServiceClient)

catkin	rosbuild

Contents

1.1 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
```

1.1.1 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/indigo-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/indigo-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.

Toggle line numbers

```
 1 #!/usr/bin/env python
 2 # license removed for brevity
 3 import rospy
 4 from std_msgs.msg import String
 5
 6 def talker():
 7     pub = rospy.Publisher('chatter', String, queue_size=10)
 8     rospy.init_node('talker', anonymous=True)
 9     rate = rospy.Rate(10) # 10hz
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         rate.sleep()
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

1.1.2 The Code Explained

Now, let's break the code down.

Toggle line numbers

```
1 #!/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.

Toggle line numbers

```
3 import rospy
4 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.

Toggle line numbers

```
7     pub = rospy.Publisher('chatter', String, queue_size=10)
8     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 the 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 `"/"`.

Toggle line numbers

```
9     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!)

Toggle line numbers

```
10     while not rospy.is_shutdown():
11         hello_str = "hello world %s" % rospy.get_time()
12         rospy.loginfo(hello_str)
13         pub.publish(hello_str)
14         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(String(str))` that publishes to our `chatter` topic using a newly created `String` message. The loop calls `r.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:

Toggle line numbers

```
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         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.

1.2 Writing the Subscriber Node

1.2.1 The Code

Download the listener.py (https://raw.github.com/ros/ros_tutorials/indigo-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/indigo-devel/rospy_tutorials/001_talker_listener/listener.py
```

The file contents look close to:

Toggle line numbers

```
 1 #!/usr/bin/env python
 2 import rospy
 3 from std_msgs.msg import String
 4
 5 def callback(data):
 6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
 7
 8 def listener():
 9
10     # In ROS, nodes are uniquely named. If two nodes with the same
11     # node are launched, the previous one is kicked off. The
12     # anonymous=True flag means that rospy will choose a unique
13     # name for our 'listener' node so that multiple listeners can
14     # run simultaneously.
15     rospy.init_node('listener', anonymous=True)
16
17     rospy.Subscriber("chatter", String, callback)
18
19     # spin() simply keeps python from exiting until this node is stopped
20     rospy.spin()
21
22 if __name__ == '__main__':
23     listener()
```

Don't forget to make the node executable:

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

1.2.2 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.

Toggle line numbers

```
  15     rospy.init_node('listener', anonymous=True)
  16
  17     rospy.Subscriber("chatter", String, callback)
  18
  19     # spin() simply keeps python from exiting until this node is sto
pped
  20     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.

1.3 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
```

1.4 Running the nodes

Running nodes requires you have a ROS core started. A ROS core is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate. Open a new shell, and type:

```
$ roscore
```

`roscore` will output something similar to this:

```
... logging to /u/nkoenig/ros-jaunty/ros/log/d92b213a-90d4-11de-9344-0
0301b8246bf/roslaunch-ncq-11315.log
... loading XML file [/u/nkoenig/ros-jaunty/ros/tools/roslaunch/roscor
e.xml]
Added core node of type [rosout/rosout] in namespace [/]
started roslaunch server http://ncq:60287/

SUMMARY
========

NODES

starting new master (master configured for auto start)
process[master]: started with pid [11338]
ROS_MASTER_URI=http://ncq:11311/
setting /run_id to d92b213a-90d4-11de-9344-00301b8246bf
+PARAM [/run_id] by /roslaunch
+PARAM [/roslaunch/uris/ncq:60287] by /roslaunch
process[rosout-1]: started with pid [11353]
started core service [/rosout]
+SERVICE [/rosout/get_loggers] /rosout http://ncq:36277/
+SERVICE [/rosout/set_logger_level] /rosout http://ncq:36277/
+SUB [/time] /rosout http://ncq:36277/
+PUB [/rosout_agg] /rosout http://ncq:36277/
+SUB [/rosout] /rosout http://ncq:36277/
```

Now everything is set to run talker/listener. Open a new shell and type:

```
$ rosrun beginner_tutorials talker.py
```

Now in the original shell type:

```
$ rosrun beginner_tutorials listener.py
```

`rosrun` is just a convenience script. You're also free to just type `./talker.py`.

Talker should begin outputting text similar to:

```
[INFO] [WallTime: 1394915011.927728] hello world 1394915011.93
[INFO] [WallTime: 1394915012.027887] hello world 1394915012.03
[INFO] [WallTime: 1394915012.127884] hello world 1394915012.13
[INFO] [WallTime: 1394915012.227858] hello world 1394915012.23
...
```

And listener should begin outputting text similar to:

```
[INFO] [WallTime: 1394915043.555022] /listener_9056_1394915043253I hea
rd hello world 1394915043.55
[INFO] [WallTime: 1394915043.654982] /listener_9056_1394915043253I hea
rd hello world 1394915043.65
[INFO] [WallTime: 1394915043.754936] /listener_9056_1394915043253I hea
rd hello world 1394915043.75
[INFO] [WallTime: 1394915043.854918] /listener_9056_1394915043253I hea
rd hello world 1394915043.85
...
```

Now that you've written your first listener node, you should also know that ROS comes with its own generic listener for any topic called rostopic (/rostopic). If you run `rostopic echo topic_name`, you get output similar to what you wrote with `listener.py`:

```
$ rostopic echo chatter
```

```
data: hello world 1394915083.35
---
data: hello world 1394915083.45
---
data: hello world 1394915083.55
---
data: hello world 1394915083.65
---
data: hello world 1394915083.75
---
...
```

Congratulations! You've just run your first ROS python nodes. For more example code, see the rospy_tutorials (/rospy_tutorials) package, or move on to the next tutorial rospy_tutorials/Tutorials/WritingServiceClient (/rospy_tutorials/Tutorials/WritingServiceClient).

Wiki: rospy_tutorials/Tutorials/WritingPublisherSubscriber (last edited 2015-10-22 19:17:06 by RoeeSefi (/RoeeSefi))