Lecture 07.01 Publishing to topics

advertise

New topics must first be registered with big Other `roscore`, which will thereafter *advertise* this topic. In `rospy`, the syntax is as follows.

```
pub = rospy.Publisher(<topic name string>,<message_type>)
```

message type

The first argument is the name of the topic and the second is the *message type* (all messages on a topic have the same type). This registers the topic name.

Later, we will learn to create our own message types, but for now we'll stick to the standard message types defined by the ROS package *std_msgs*. For a list of available message types in `std_msgs`, see

std_msgs

wiki.ros.org/std_msgs.

07.01.1 Creating a simple publisher node

The code accompanying the text has a simple publisher node in the `rico_topics` package. You should use `catkin_create_pkg` to create a parallel package in your own code repository, as follows.

```
catkin_create_pkg my_topics \
rospy std_msgs message_runtime message_generation
```

We'll need the dependencies listed above. Create a new Python file in `my_topics/src` with the following.

```
touch my_topics/src/topic_publisher.py
```

Open the empty `topic_publisher.py` in a text editor. You'll want to enter here the same code as appears in the sample `topic_publisher.py` from `robotics-book-code/rico_topics/src`, which is listed in Figure 07.1.

shebang

executable

Since this is the first `rospy` node we've written, it's worth considering it in detail. The first line is called a *shebang* and indicates the file is *executable* and the relevant interpreter (in this case, `python`). One more step is actually required to make your new file executable in Ubuntu: you must change its *permissions* to be executable, as follows.

permissions

```
chmod u+x my_topics/src/topic_publisher.py
```

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32 # standard int

# Setup: initialize node, register topic, set rate
rospy.init_node( # initialize node
  'topic_publisher' # node default name
)
pub = rospy.Publisher( # register topic w/roscore
  'counter', # topic name
  Int32, # topic type
  queue_size=5 # queue size
)
rate = rospy.Rate(2) # adaptive rate in Hz

# Loop: publish, count, sleep
count = 0
while not rospy.is_shutdown(): # until ctrl-c
    pub.publish(count) # publish count
    count += 1 # increment
    rate.sleep() # set by rospy.Rate above
```

Figure 07.1: `rico_topics/src/topic_publisher.py` listing.

07.01.2 Setting up the node

Back to Figure 07.1, following the shebang, there's the loading of packages via Python's package **`import`** mechanism. Note that we're using both `rospy` and `std_msgs`, which we included in our `package.xml` when we used `catkin_create_pkg`. Then follows the initalization of a ROS node via *`rospy.init_node`*. For more details on initializing nodes, see

`wiki.ros.org/rospy/Overview/Initialization and Shutdown`.

`rospy.init_node`

We then register a topic `counter` of type `Int32` (from `std_msgs`) and *queue size* of `5` via *`rospy.Publisher`*. Queue size is how many buffered messages should be stored on the topic. The general guidance is: use more than you need. For more on selecting queue size, see

`wiki.ros.org/rospy/Overview/Publishers and Subscribers`.

queue size

`rospy.Publisher`

Finally, we use `rospy.Rate` to specify our desired loop timing. This powerful mechanism will be used in a moment to adaptively maintain a looping rate.

07.01.3 Publishing to the topic

The **`while`** loop in Figure 07.1 is pretty simple: while the node isn't shut down,

1. publish the count to topic `counter` via the `publish` method of the object `pub` created by `rospy.Publisher`,
2. increment the count, and
3. wait until the `sleep` method says to iterate.

The `Rate` object `rate` can use its `sleep` method to adaptively attempt to keep the loop running at the specified rate. This timing mechanism is quite convenient.

07.01.4 Running and verifying the node

First, we need to `catkin_make` the workspace to make our new package available. Navigate (`cd`) in Terminal to your workspace root directory.

```
catkin_make
```

If you have an error involving the Python packages `em`, `yaml`, or `catkin_pkg`, try installing them with the following.

```
pip install empy pyyaml catkin_pkg
```

Once your `catkin_make` finishes successfully, source the workspace.

```
source devel/setup.bash
```

Now open a new Terminal and start a `roscore` service. Now we can `rosrun` the new node!

```
rosrun my_topics topic_publisher.py
```

Our node is running! Let's check the current topics to see if `counter` is being advertised. A nice tool for this is `rostopic`.

```
rostopic list
```

```
/counter
/rosout
/rosout_agg
```

So it is. We can ignore the other topics, which always appear. Let's see what is being published to the topic.

```
rostopic echo counter -n 3
```

```
data: 17
---
data: 18
---
data: 19
---
```

The `-n 3` option/value shuts down `rostopic` after three messages. Otherwise it would continue until we Ctrl+C.

We can also see how the successful our `sleep` method is at maintaining our desired loop rate. (We have to Ctrl+C to stop this one.)

```
rostopic hz counter
```

```
subscribed to [/counter]
average rate: 2.001
    min: 0.500s max: 0.500s std dev: 0.00000s window: 2
average rate: 1.999
    min: 0.500s max: 0.501s std dev: 0.00051s window: 4
average rate: 2.000
    min: 0.498s max: 0.501s std dev: 0.00095s window: 6
average rate: 2.000
    min: 0.498s max: 0.501s std dev: 0.00088s window: 7
```

Not too bad!