

# ROS 101: CREATING A PUBLISHER NODE

by Martin Cote | Sep 23, 2014 | Teaching and Learning | 4 comments



In our [previous post](#), we graduated from driving a Husky, to taking on a Grizzly! Now it's time to get down and dirty with what ROS is really made of, nodes! We will first be creating a workspace to work from, then we will write a simple publisher that will make our virtual Husky drive around randomly. If this is your first time visiting a Clearpath Robotics ROS 101 blog, get started [here](#).

## Creating a Workspace & Package

Before we begin writing a node, we need to create a workspace and a package.

Workspaces are simply directories to store all of your packages. First we will need to create a new directory.

```
mkdir ~/ros101
```

This created a directory in your home folder which we will use as a workspace directory. We now need to create a subdirectory in your workspace directory to store all your source code for your packages

```
mkdir ~/ros101/src
```

The last step to creating the workspace will be to initialize the workspace with `catkin_init_workspace`.

```
cd ~/ros101/src  
catkin_init_workspace
```

Now that our workspace has been created, we will create a package in the src directory we just created. This can be done by navigating to the ~/ros101/src directory, which you should have already done in the last step, and using the catkin\_create\_pkg command followed by what we want the package to be named, and then followed by what other packages our package will be dependent on; this command creates another directory for your new package, and two new configuration files inside that directory with some default settings.

```
catkin_create_pkg random_husky_driver roscpp std_msgs
```

You can see that this created CMakeLists.txt and package.xml inside the random\_husky\_driver directory; this is also where you will store all your source code for your packages. The roscpp and std\_msgs dependencies were added into the CMakeLst.txt and package.xml.

## Writing the publisher

As mentioned in our previous post, a publisher publishes messages to a particular topic. For this tutorial, we will be publishing random commands to the /husky/cmd\_vel topic to make your Husky visualization drive itself. Start by creating a file in your ~/ros101/src/random\_husky\_driver/src directory called random\_driver.cpp, and copy the following code.

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    //Initializes ROS, and sets up a node
    ros::init(argc, argv, "random_husky_commands");
    ros::NodeHandle nh;

    //Ceates the publisher, and tells it to publish
    //to the husky/cmd_vel topic, with a queue size of 100
    ros::Publisher pub=nh.advertise<geometry_msgs::Twist>("husky/cmd_vel", 100);

    //Sets up the random number generator
    srand(time(0));

    //Sets the loop to publish at a rate of 10Hz
    ros::Rate rate(10);

    while(ros::ok()) {
        //Declares the message to be sent
        geometry_msgs::Twist msg;
        //Random x value between -2 and 2
        msg.linear.x=4*double(rand())/double(RAND_MAX)-2;
        //Random y value between -3 and 3
        msg.angular.z=6*double(rand())/double(RAND_MAX)-3;
        //Publish the message
        pub.publish(msg);

        //Delays untill it is time to send another message
        rate.sleep();
    }
}
```

```
    }
}
```

Let's break down this code line by line,

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
```

These lines includes the headers that we are going to need. The `<ros/ros.h>` header is required for ROS functionality and the `<geometry_msgs/Twist.h>` is added so that we can create a message of that type.

```
ros::init(argc, argv, "random_husky_commands");
ros::NodeHandle nh;
```

The first line, `ros::init`, is used to initialize the ROS node, and name it "random\_husky\_commands", while `ros::NodeHandle` starts the node.

```
ros::Publisher pub=nh.advertise<geometry_msgs::Twist>("husky/cmd_vel", 100);
```

Publishing a message is done using `ros::Publisher pub=nh.advertise`, followed by the message type that we are going to be sending, in this case it is a `geometry_msgs::Twist`, and the topic that we are going to be sending it too, which for us is `husky/cmd_vel`.

The 100 is the message queue size, that is, if you are publishing message faster then what `roscpp` can send, 100 messages will be saved in the queue to be sent. The larger the queue, the more delay in robot movement in case of buffering.

Therefore in a real life example, you will want to have a smaller queue in the case of robot movement, where delay in movement commands are undesirable and even dangerous, but dropped messages are acceptable. In the case of sensors, it is recommended to use a larger queue, since delay is acceptable to ensure no data is lost.

```
ros::Rate rate(10)
...
rate.sleep()
```

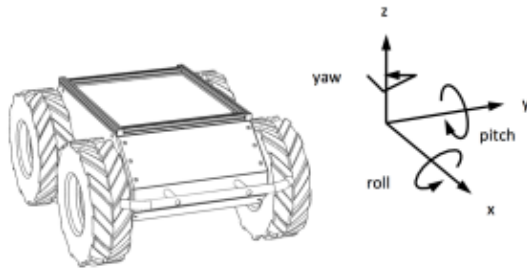
ROS is able to control the loop frequency using `ros::Rate` to dictate how rapidly the loop will run in Hz. `rate.sleep` will delay a variable amount of time such that your loop cycles at the desired frequency. This accounts for time consumed by other parts of the loop. All Clearpath robots require a minimum loop rate of 10Hz.

```
while(ros::ok())
```

The `ros::ok` function will return true unless it receives a command to shut down, either by using the `rostop kill` command, or by the user pushing Ctrl-C in a terminal.

```
geometry_msgs::Twist msg;
```

```
This creates the message we are going to send, msg, of the type geometry_msgs::Twist
msg.linear.x=4*double(rand())/double(RAND_MAX)-2;
msg.angular.z=6*double(rand())/double(RAND_MAX)-3;
```



These lines calculate the random linear x and angular z values that will be sent to Husky.

```
pub.publish(msg)
```

We are finally ready to publish the message! The `pub.publish` adds `msg` to the publisher queue to be sent.

## Compiling the random Husky driver

Compilation in ROS is handled by the catkin build system. The first step would usually be to set up our package dependencies in the `CMakeLists.txt` and `package.xml`. However this has already been done for us when we created the package and specified our dependencies. The next step is then to declare our new node as an executable, this is done by adding the following two lines to the `CMakeLists.txt` files in `~/ros101/src/random_husky_driver`

```
add_executable(random_driver src/random_driver.cpp)
target_link_libraries(random_driver ${catkin_LIBRARIES})
```

The first line creates the executable called `random_driver`, and directs ROS to its source files. The second line specifies what libraries will be used. Now we need to build our workspace using the `catkin_make` command in the workspace directory

```
cd ~/ros101
catkin_make
```

Let's bring up the husky visualization as we did in a previous blog post.

```
roslaunch husky_gazebo husky_empty_world.launch
```

The final step is to source your `setup.bash` file in the workspace you have created. This script allows ROS to find the packages that are contained in your workspace. **Don't forget this process will have to be done on every new terminal instance!**

```
source ~/ros101/devel/setup.bash
```

It's now time to test it out! Make sure you have an instance of `roscore` running in a separate terminal, then start the node.

```
roslaunch random_husky_driver random_driver
```

You should now see Husky drive around! In a new terminal window, we can make sure that our node is publishing to the `/husky/cmd_vel` topic by echoing all messages on this topic

```
rostopic echo /husky/cmd_vel
```

```
linear:
  x: -0.988633466414
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -1.2239008944
---
linear:
  x: -1.66241014919
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.11694186093
---
linear:
  x: 0.497968052746
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.237037784065
---
linear:
  x: 1.92314005267
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.9483107782882
---
```

You should now see a stream of random linear x and angular z values.

Looking for other Clearpath tutorials? [Click here](#) for more ROS tutorials, or visit our [Knowledge Base](#) to browse all of our tutorials.



## RELATED POSTS

**ROBOTS 101 –  
LASERS**

**HOW TO: UNPACK  
GRIZZLY ROBOTIC  
UTILITY VEHICLE**

**WEBINAR  
RECORDING:  
TRANSFORM YOUR**

**HOLIDAY HACK DAY**

## 4 COMMENTS



**Mark Jones** on January 12, 2016 at 3:00 pm

Hey, great tutorials.

Just a note with the line “add\_executable(random\_driver random\_driver.cpp)”, it should be “add\_executable(random\_driver src/random\_driver.cpp)”.

Cheers

[REPLY](#)



**Chris Bogdon** on February 17, 2016 at 1:45 pm

Thanks Mark! And good catch. I've updated the tutorial with the correct line.

[REPLY](#)



**S. Usama** on March 22, 2016 at 4:39 pm

Really helpful.

After that, I tried to use a node to subscribe some data and then publish it to some other topic. I mean I want to use same node for publishing and subscribing. Can you help me with an example?

[REPLY](#)



**Chris Bogdon** on March 24, 2016 at 1:15 pm

Hi S. Usama. Check out our Subscriber tutorial here:

<http://clearpathrobotics.com/guides/ros/Creating%20subscriber.html>

Hope this helps! You can find all of our ROS tutorials at

<http://www.clearpathrobotics.com/guides/ros>

[REPLY](#)

## Trackbacks/Pingbacks

1. [ROS101: Creating a Subscriber Using GitHub](#) - [...] previously learned how to write a publisher node to move Husky randomly. BUT! What good is publishing all these messages...

SEARCH

## Categories

Culture & Community

In the Field: Customer Spotlight

In the News

Robot Forge

ROS Spotlight

Teaching and Learning  
Tradeshows & Conferences

Recent Posts

- SIT Advances Autonomous Navigation Research Using Jackal UGV
- Clearpath Robotics Attends ROSCon & iROS 2017
- MIT Develops Autonomous “Socially Aware” Robot Using Jackal UGV
- From Books to Bots: 2017 Summer Term Co-op
- Clearpath Presents Jorge Cham and More at iROS 2017

SUBSCRIBE TO UPDATES

EMAIL\*

SUBMIT

Search...

ROBOTS	SERVICES	CUSTOMERS	SUPPORT CENTRE	COMPANY
WARTHOG UGV	COMPETENCIES	CUSTOMER STORIES	KNOWLEDGE BASE	OUR STORY
HUSKY UGV	RESEARCH SERVICES		SUBMIT TICKET	OUR TEAM
JACKAL UGV	INDUSTRIAL SERVICES		ROS COMMUNITY	CAREERS
OTTO 1500			PR2 SUPPORT	BLOG
RIDGEBACK			PR2 RESOURCES	NEWS
TURTLEBOT EUCLID			TUTORIALS	PRESS INQUIRIES
TURTLEBOT 2	CONTACT US			
HUMMINGBIRD UAV	Clearpath Robotics Inc.			
PELICAN UAV	1425 Strasburg Rd. Suite 2A			
HERON USV	Kitchener, On N2R 1H2			
ACCESSORIES				
	1 800 301 3863			
	info@clearpathrobotics.com			

TALK TO A HUMAN

