You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Iron.

Understanding topics

Goal: Use rqt_graph and command line tools to introspect ROS 2 topics.

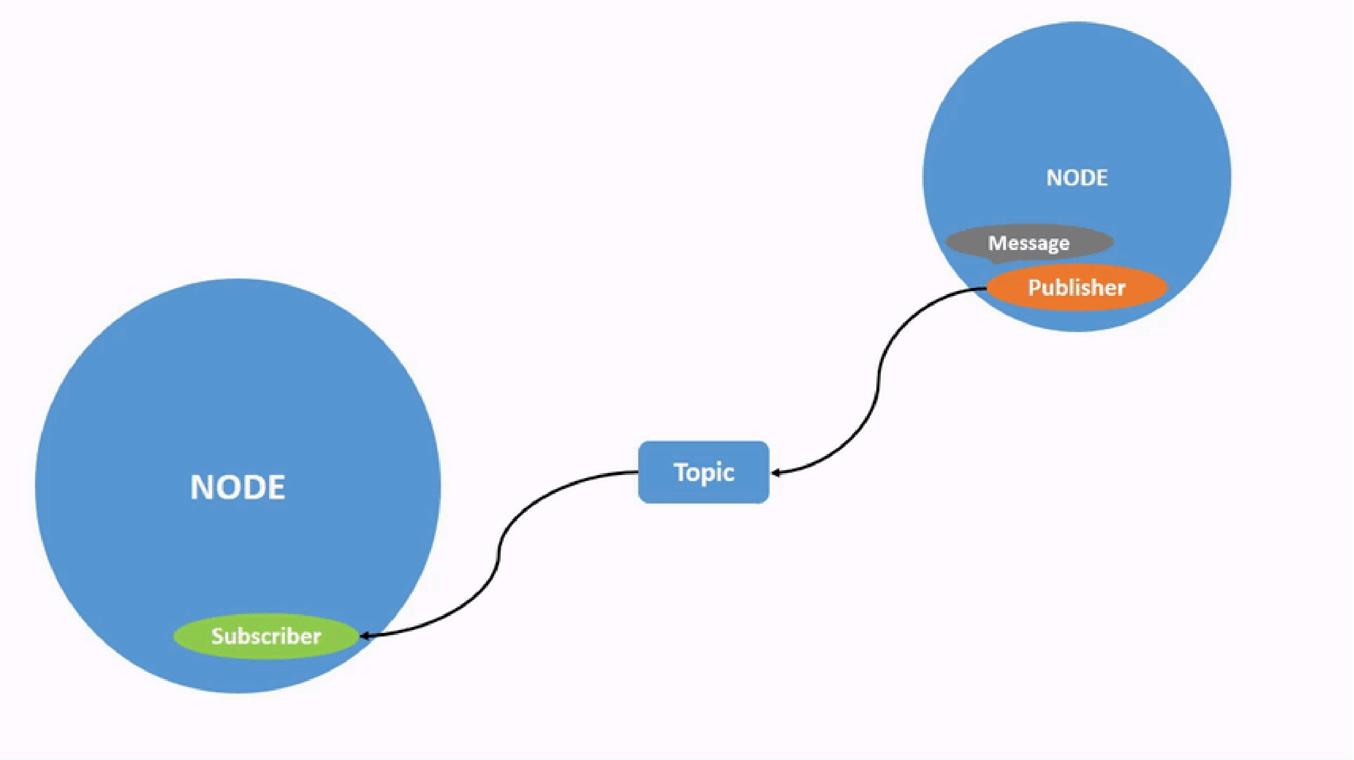
Tutorial level: Beginner

Time: 20 minutes

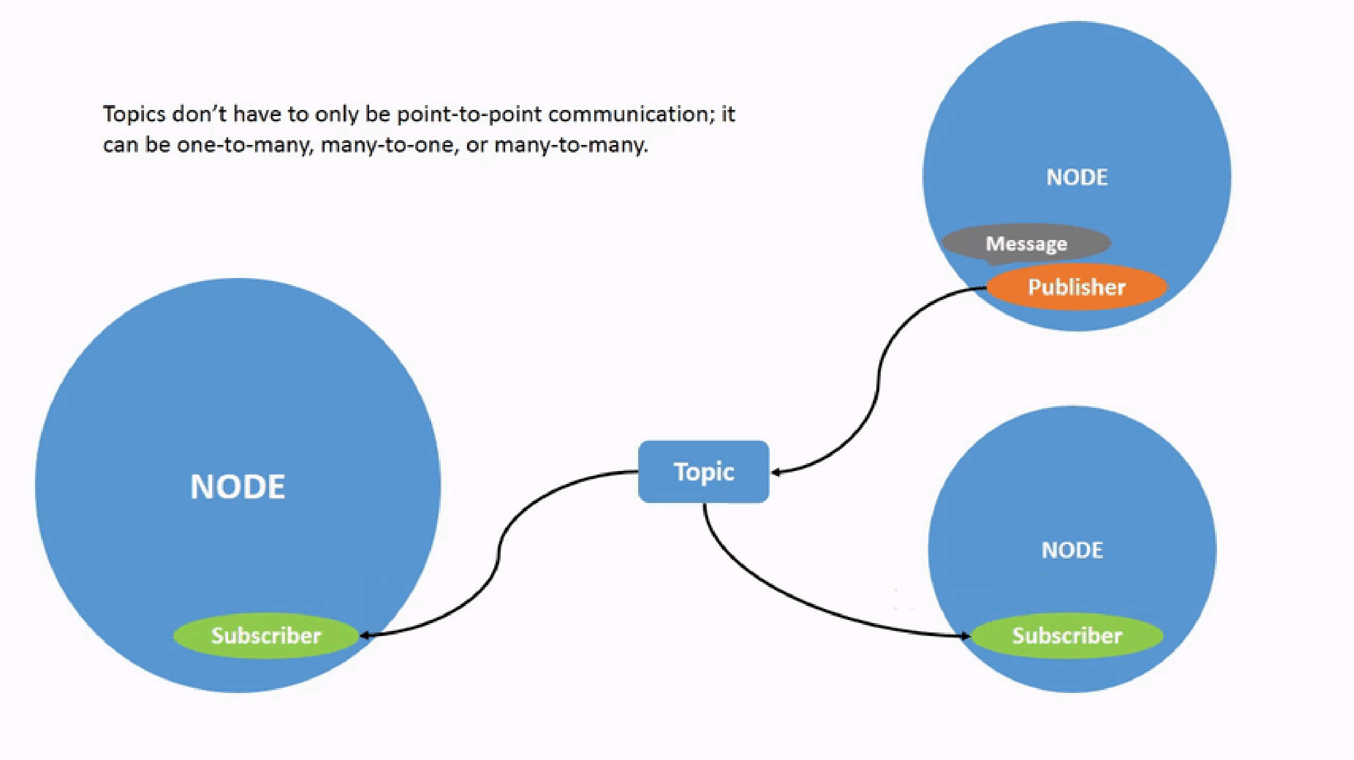
Contents

Background

ROS 2 breaks complex systems down into many modular nodes. Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.



A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.


Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
NODE
Message
Publisher
Topic
NODE
Subscriber
NODE
Subscriber

Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.

Prerequisites

The previous tutorial provides some useful background information on nodes that is built upon here.

As always, don’t forget to source ROS 2 in every new terminal you open.

Tasks

1 Setup

By now you should be comfortable starting up turtlesim.

Open a new terminal and run:

```
ros2 run turtlesim turtlesim_node
```

Open another terminal and run:

```
ros2 run turtlesim turtle_teleop_key
```

Recall from the previous tutorial that the names of these nodes are `/turtlesim` and `/teleop_turtle` by default.

2 rqt_graph

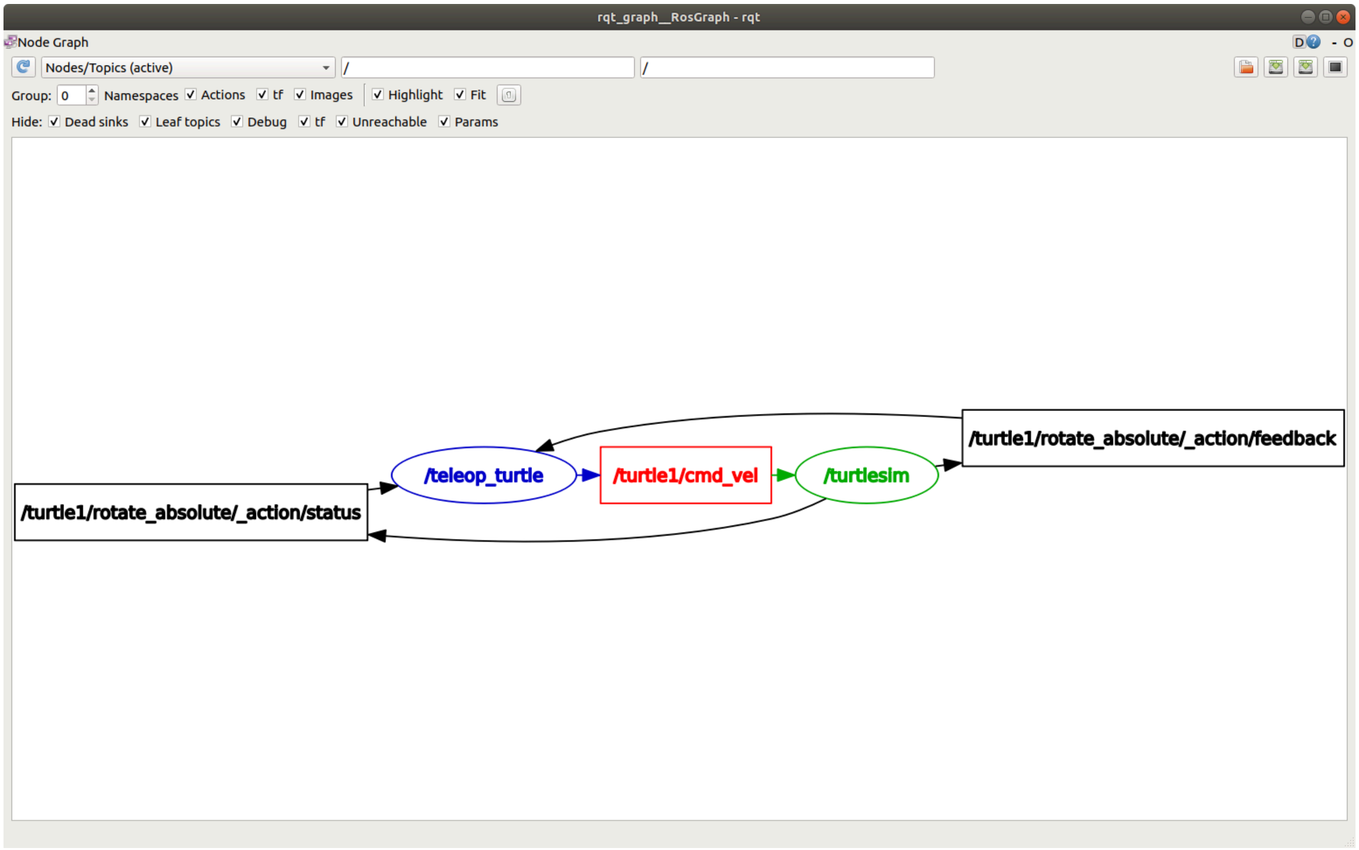
Throughout this tutorial, we will use `rqt_graph` to visualize the changing nodes and topics, as well as the connections between them.

The turtlesim tutorial tells you how to install rqt and all its plugins, including `rqt_graph`.

To run rqt_graph, open a new terminal and enter the command:

```
rqt_graph
```

You can also open rqt_graph by opening `rqt` and selecting **Plugins** > **Introspection** > **Node Graph**.



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (active) / /

Group: 0 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ tf ☑ Unreachable ☑ Params

/turtle1/rotate_absolute/_action/feedback
/teleop_turtle
/turtle1/cmd_vel
/turtlesim
/turtle1/rotate_absolute/_action/status

You should see the above nodes and topic, as well as two actions around the periphery of the graph (let's ignore those for now). If you hover your mouse over the topic in the center, you'll see the color highlighting like in the image above.

The graph is depicting how the `/turtlesim` node and the `/teleop_turtle` node are communicating with each other over a topic. The `/teleop_turtle` node is publishing data (the keystrokes you enter to move the turtle around) to the `/turtle1/cmd_vel` topic, and the `/turtlesim` node is subscribed to that topic to receive the data.

The highlighting feature of rqt_graph is very helpful when examining more complex systems with many nodes and topics connected in many different ways.

rqt_graph is a graphical introspection tool. Now we'll look at some command line tools for introspecting topics.

3 ros2 topic list

Running the `ros2 topic list` command in a new terminal will return a list of all the topics currently active in the system:

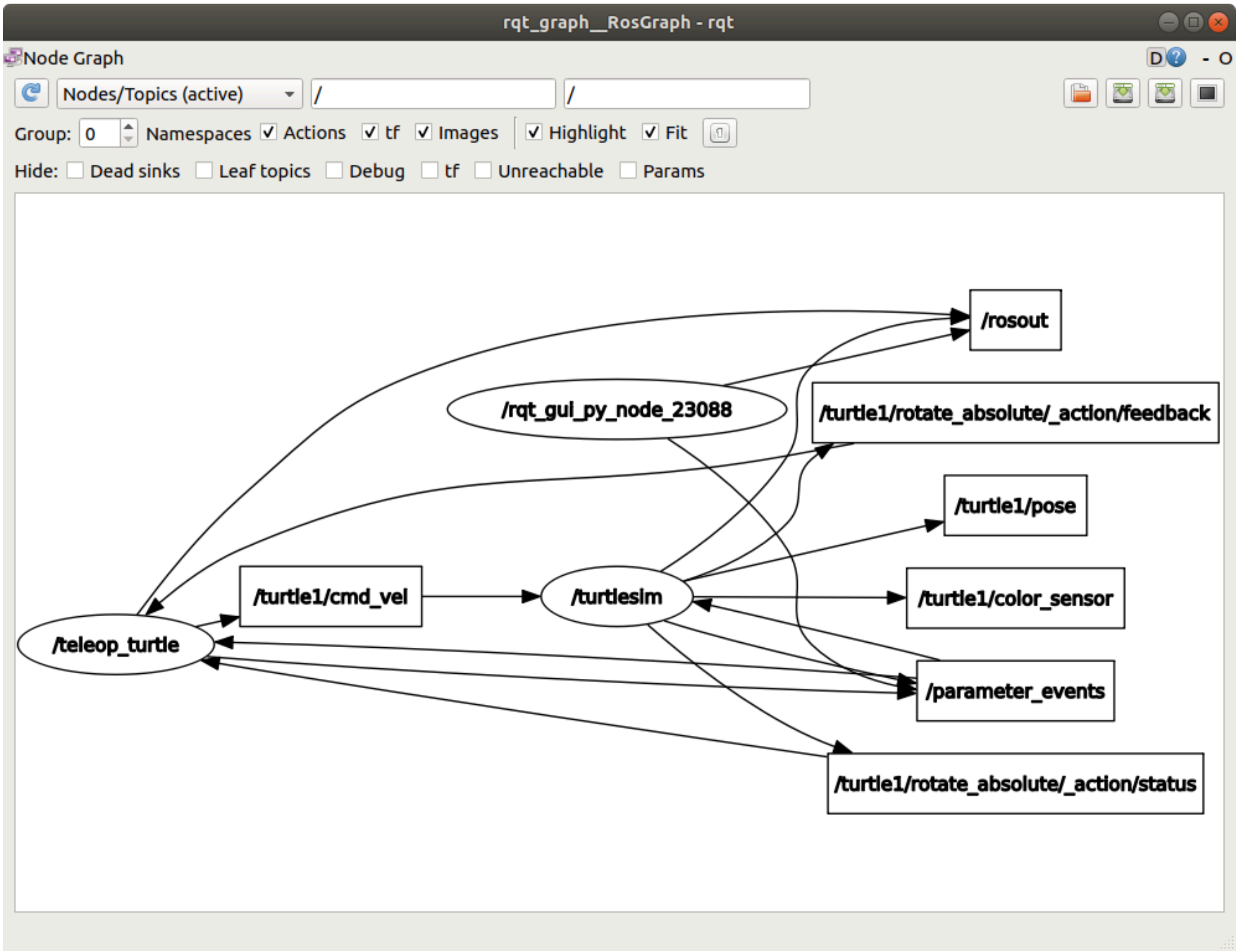
```
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

`ros2 topic list -t` will return the same list of topics, this time with the topic type appended in brackets:

```
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
```

These attributes, particularly the type, are how nodes know they're talking about the same information as it moves over topics.

If you're wondering where all these topics are in rqt_graph, you can uncheck all the boxes under **Hide:**



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (active) / /

Group: 0 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: □ Dead sinks □ Leaf topics □ Debug □ tf □ Unreachable □ Params

For now, though, leave those options checked to avoid confusion.

4 ros2 topic echo

To see the data being published on a topic, use:

```
ros2 topic echo <topic_name>
```

Since we know that `/teleop_turtle` publishes data to `/turtlesim` over the `/turtle1/cmd_vel` topic, let’s use `echo` to introspect that topic:

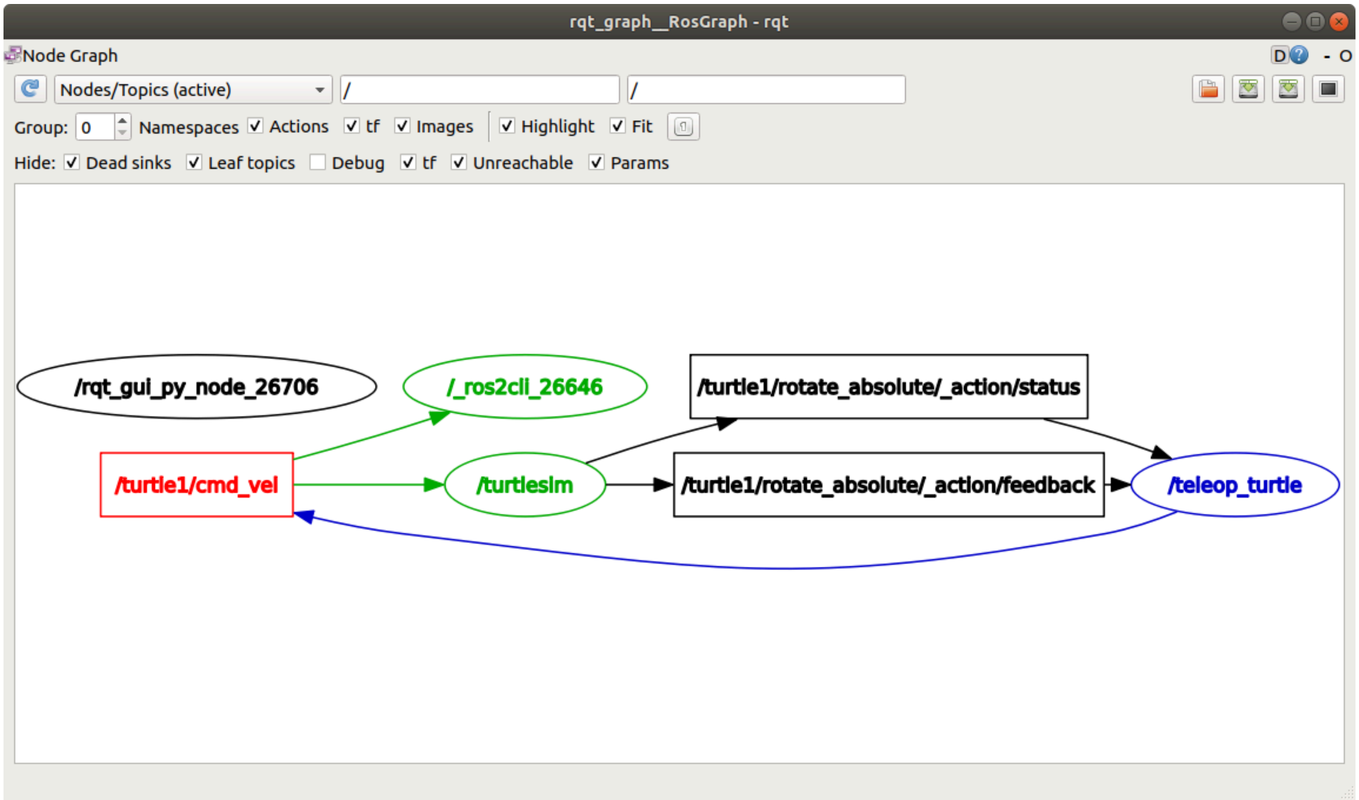
```
ros2 topic echo /turtle1/cmd_vel
```

At first, this command won’t return any data. That’s because it’s waiting for `/teleop_turtle` to publish something.

Return to the terminal where `turtle_teleop_key` is running and use the arrows to move the turtle around. Watch the terminal where your `echo` is running at the same time, and you'll see position data being published for every movement you make:

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
  ---
```

Now return to rqt_graph and uncheck the **Debug** box.


rqt_graph__RosGraph - rqt
Node Graph
D
/rqt_gui_py_node_26706
/_ros2cli_26646
/turtle1/rotate_absolute/_action/status
/turtle1/cmd_vel
/turtlesim
/turtle1/rotate_absolute/_action/feedback
/teleop_turtle

Nodes/Topics (active) / /

Group: 0 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics □ Debug ☑ tf ☑ Unreachable ☑ Params

`/_ros2cli_26646` is the node created by the `echo` command we just ran (the number might be different). Now you can see that the publisher is publishing data over the `cmd_vel` topic, and two subscribers are subscribed to it.

5 ros2 topic info

Topics don't have to only be one-to-one communication; they can be one-to-many, many-to-one, or many-to-many.

Another way to look at this is running:

```
ros2 topic info /turtle1/cmd_vel
```

Which will return:

```
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
```

6 ros2 interface show

Nodes send data over topics using messages. Publishers and subscribers must send and receive the same type of message to communicate.

The topic types we saw earlier after running `ros2 topic list -t` let us know what message type is used on each topic. Recall that the `cmd_vel` topic has the type:

```
geometry_msgs/msg/Twist
```

This means that in the package `geometry_msgs` there is a `msg` called `Twist`.

Now we can run `ros2 interface show <msg type>` on this type to learn its details. Specifically, what structure of data the message expects.

```
ros2 interface show geometry_msgs/msg/Twist
```

For the message type from above it yields:

```
# This expresses velocity in free space broken into its linear and angular parts.

    Vector3  linear
            float64 x
            float64 y
            float64 z
    Vector3  angular
            float64 x
            float64 y
            float64 z
```

This tells you that the `/turtlesim` node is expecting a message with two vectors, `linear` and `angular`, of three elements each. If you recall the data we saw `/teleop_turtle` passing to `/turtlesim` with the `echo` command, it's in the same structure:

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
  ---
```

7 ros2 topic pub

Now that you have the message structure, you can publish data to a topic directly from the command line using:

```
ros2 topic pub <topic_name> <msg_type> '<args>'
```

The `'<args>'` argument is the actual data you'll pass to the topic, in the structure you just discovered in the previous section.

It's important to note that this argument needs to be input in YAML syntax. Input the full command like so:

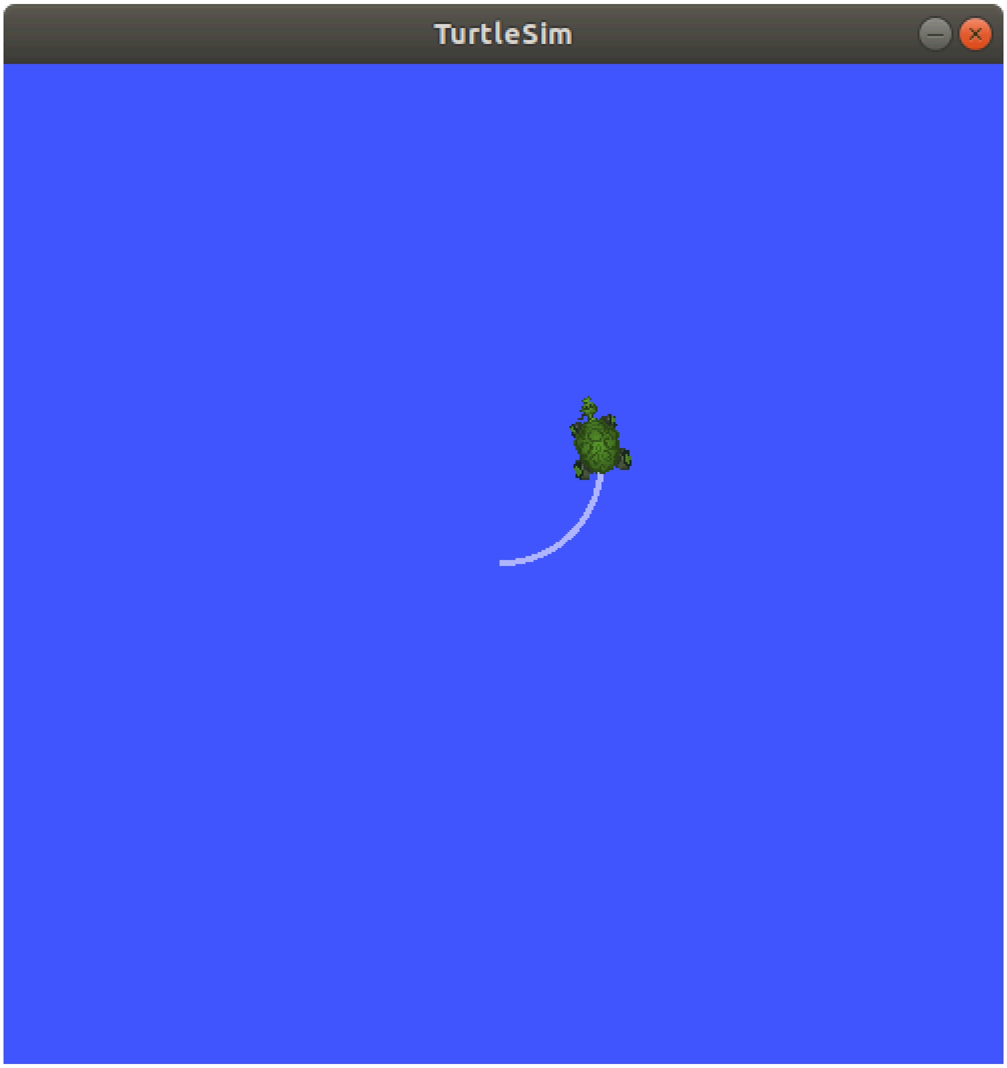
```
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0},
angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

`--once` is an optional argument meaning "publish one message then exit".

You will see the following output in the terminal:

```
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0),
angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))
```

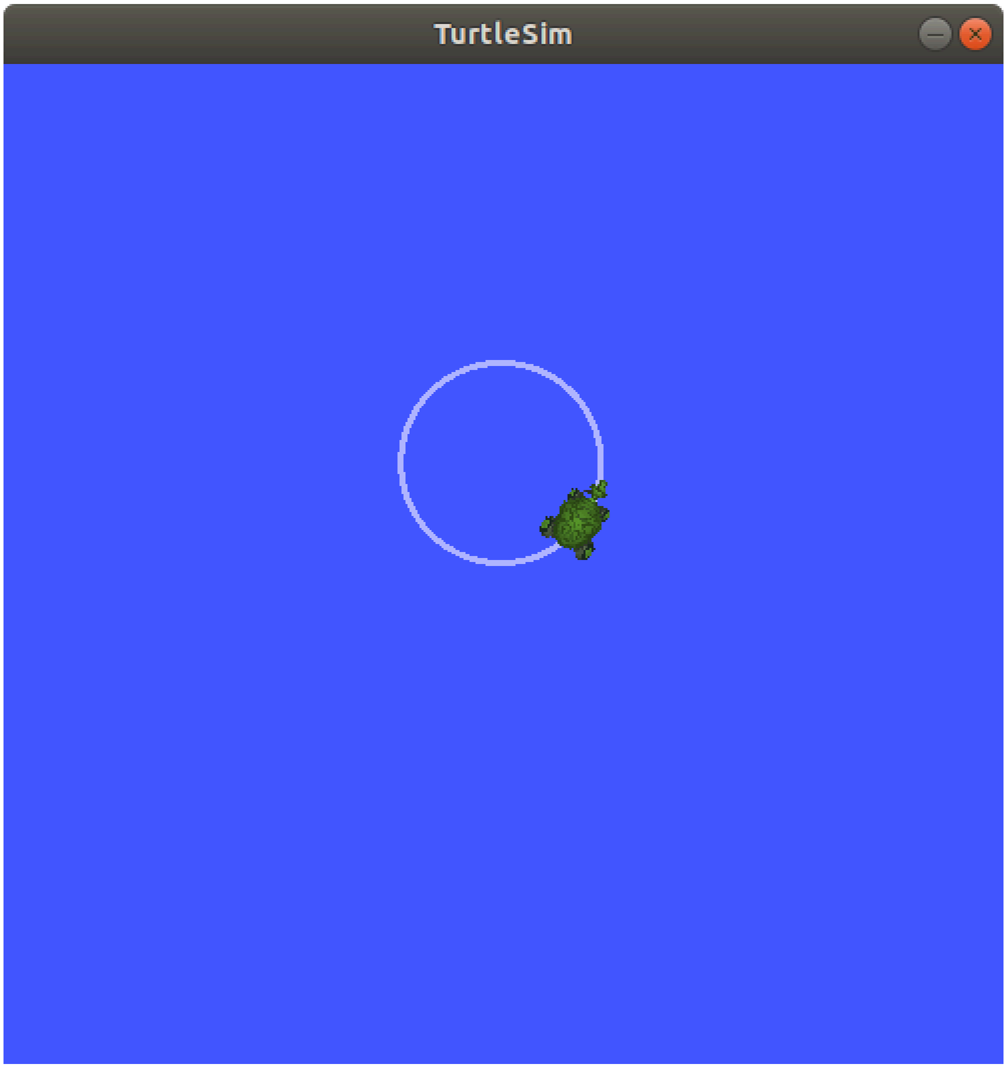
And you will see your turtle move like so:



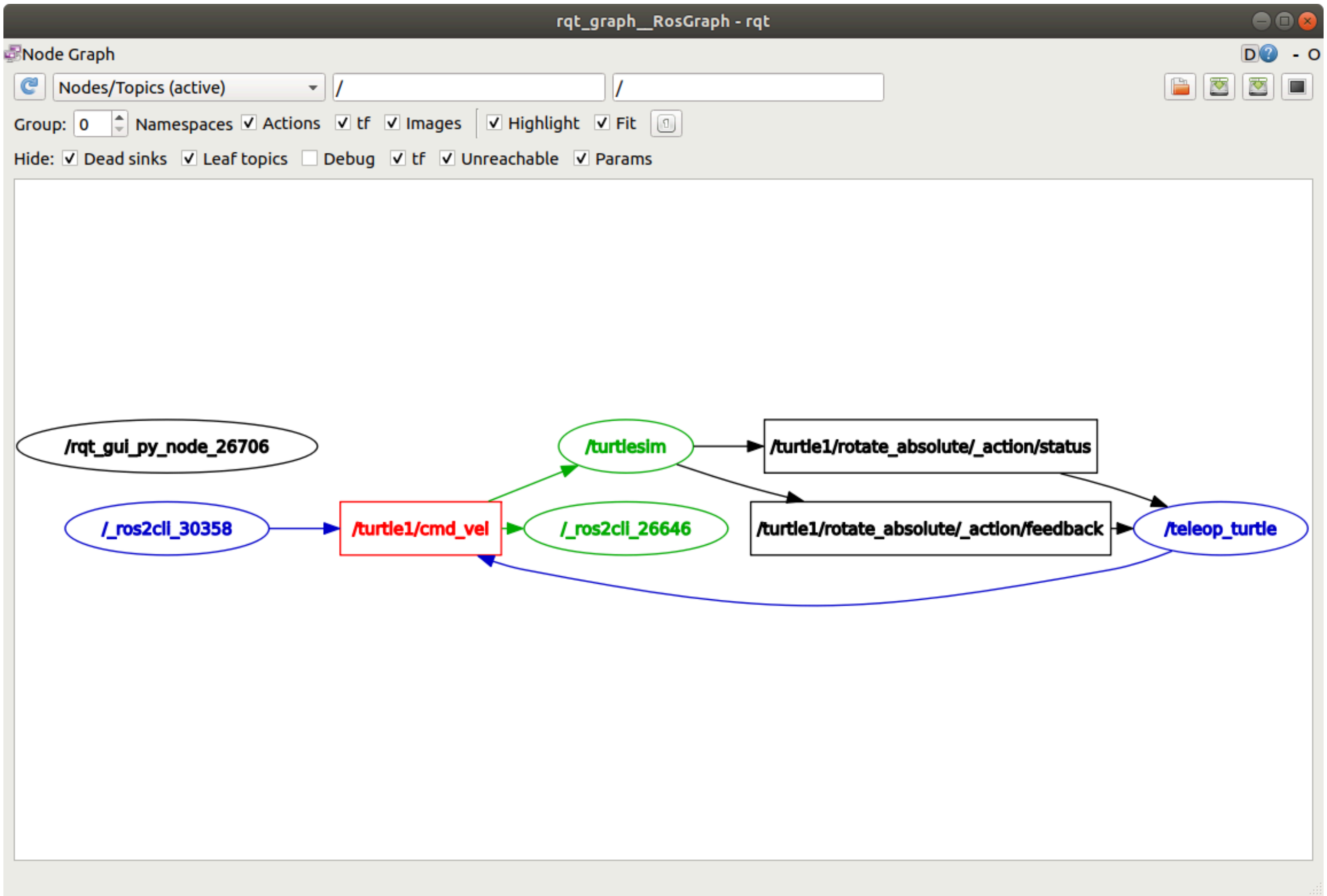
The turtle (and commonly the real robots which it is meant to emulate) require a steady stream of commands to operate continuously. So, to get the turtle to keep moving, you can run:

```
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z:
0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

The difference here is the removal of the `--once` option and the addition of the `--rate 1` option, which tells `ros2 topic pub` to publish the command in a steady stream at 1 Hz.



You can refresh rqt_graph to see what's happening graphically. You will see that the `ros2 topic pub ...` node (`/_ros2cli_30358`) is publishing over the `/turtle1/cmd_vel` topic, which is being received by both the `ros2 topic echo ...` node (`/_ros2cli_26646`) and the `/turtlesim` node now.



rqt_graph__RosGraph - rqt

Node Graph

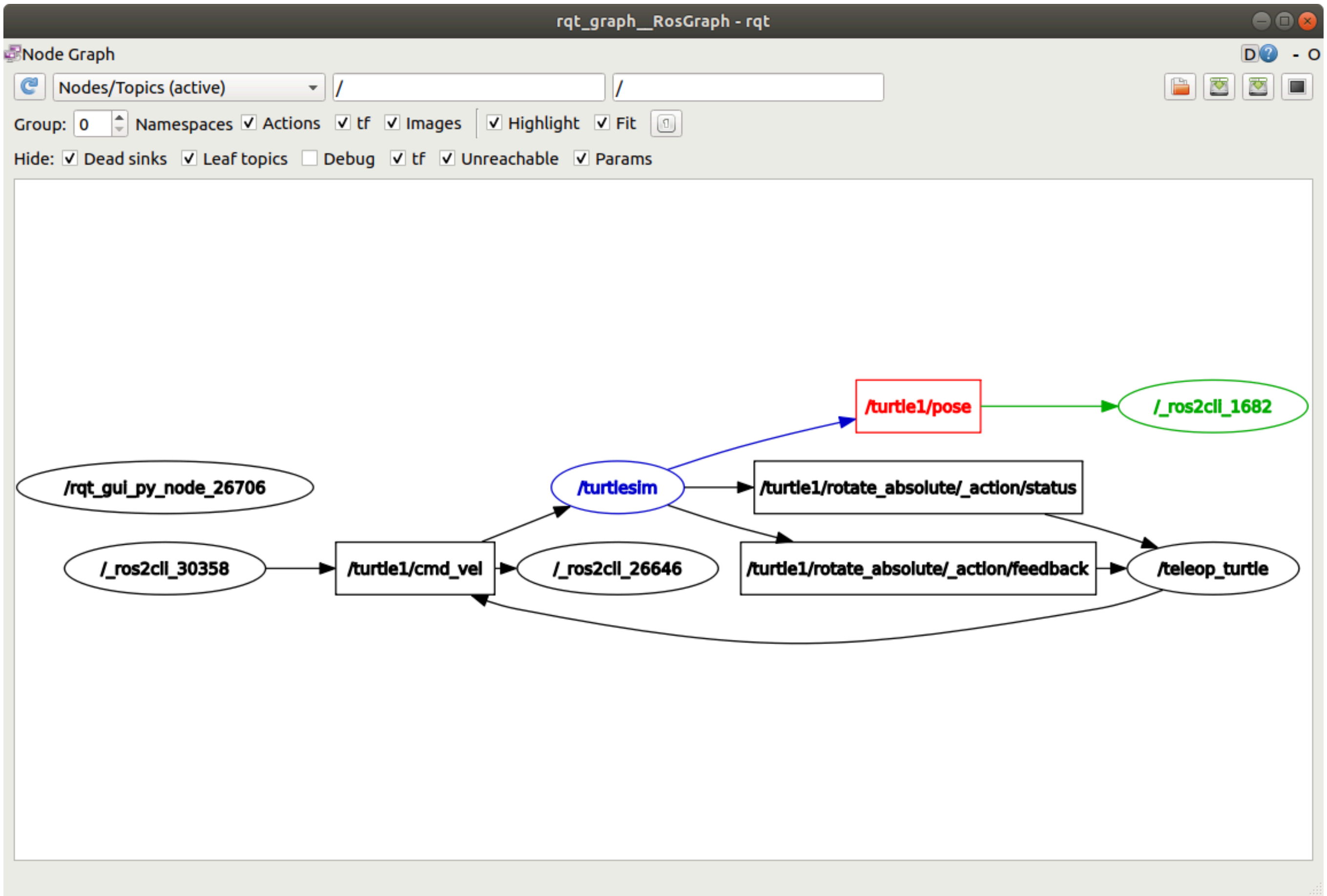
Nodes/Topics (active) / /

Group: 0 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics □ Debug ☑ tf ☑ Unreachable ☑ Params

Finally, you can run `echo` on the `pose` topic and recheck rqt_graph:

```
ros2 topic echo /turtle1/pose
```



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (active) / /

Group: 0 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☐ Debug ☑ tf ☑ Unreachable ☑ Params

You can see that the `/turtlesim` node is also publishing to the `pose` topic, which the new `echo` node has subscribed to.

When publishing messages with timestamps, `pub` has two methods to automatically fill them out with the current time. For messages with a `std_msgs/msg/Header`, the header field can be set to `auto` to fill out the `stamp` field.

```
ros2 topic pub /pose geometry_msgs/msg/PoseStamped '{header: "auto", pose: {position: {x: 1.0, y:
2.0, z: 3.0}}}'
```

If the message does not use a full header, but just has a field with type `builtin_interfaces/msg/Time`, that can be set to the value `now`.

```
ros2 topic pub /reference sensor_msgs/msg/TimeReference '{header: "auto", time_ref: "now", source:
"dumy"}'
```

8 ros2 topic hz

For one last introspection on this process, you can view the rate at which data is published using:

```
ros2 topic hz /turtle1/pose
```

It will return data on the rate at which the `/turtlesim` node is publishing data to the `pose` topic.

```
average rate: 59.354
  min: 0.005s max: 0.027s std dev: 0.00284s window: 58
```

Recall that you set the rate of `turtle1/cmd_vel` to publish at a steady 1 Hz using `ros2 topic pub --rate 1`. If you run the above command with `turtle1/cmd_vel` instead of `turtle1/pose`, you will see an average reflecting that rate.

9 Clean up

At this point you'll have a lot of nodes running. Don't forget to stop them by entering `Ctrl+C` in each terminal.

Summary

Nodes publish information over topics, which allows any number of other nodes to subscribe to and access that information. In this tutorial you examined the connections between several nodes over topics using rqt_graph and command line tools. You should now have a good idea of how data moves around a ROS 2 system.

Next steps

Next you'll learn about another communication type in the ROS graph with the tutorial Understanding services.