ROS2 tutorial

The ROS2 tutorial provides a comprehensive guide on using ROS2, including running nodes, checking nodes and topics, publishing and subscribing to topics, and using services. It explains how to create a Python package, compile and build packages, and execute nodes. The tutorial also covers topics such as parameters, publication nodes, services, actions, logs, and Python packages. It includes command-line instructions for various operations and provides links to additional resources.

https://docs.ros.org/en/humble/Tutorials.html

https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy

🐢 turtlesim

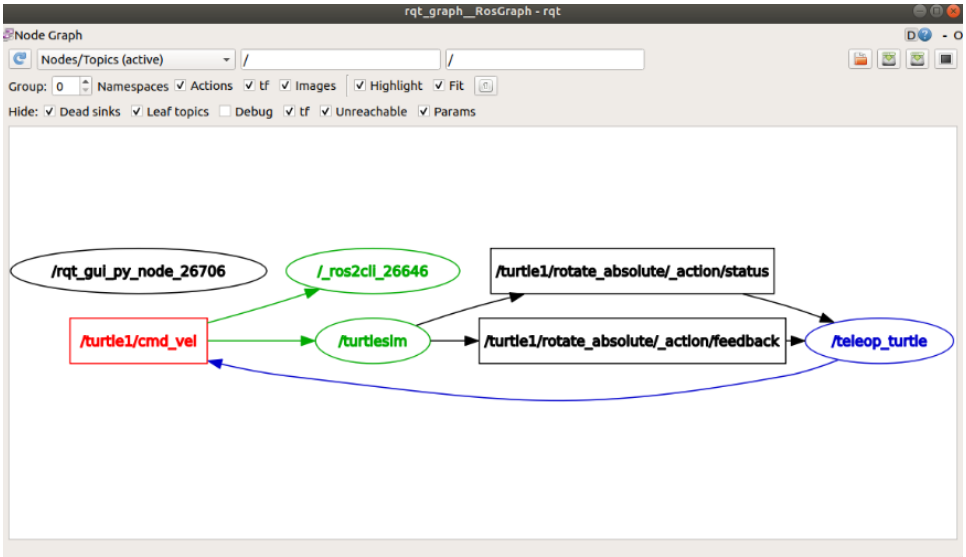
Table of contents

Nodes :
    Parameters
Topics :
Publication node :
Services :
Actions :
Logs :

Python package :

`ros2 run <package_name> <executable_name>`

`rqt_graph` :



We can see that the node /teleop_turtle publishes in the topics /turtle/cmd_vel which have subscriptions from /turtlesim (turtle GUI) and /_ros2cli… (topic echo).

Nodes :

to check nodes : `ros2 node list`

- `/rqt_gui_py_node_3886` → echo command (shows turtle speed)
- `/teleop_turtle` → control turtle
- `/turtlesim` → turtle GUI

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

Parameters

Node settings.

`ros2 param list` `ros2 param get <node_name> <parameter_name>`

`ros2 param dump /turtlesim` + `> turtlesim.yaml`

set parameter's value: `ros2 param set <node_name> <parameter_name> <value>`

Load parameter file : `ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>`

Topics :

a variable, a communication channel

To see all the active topics:

`ros2 topic list` + `-t`

To count the number of publishes and subscriptions in a topic:

`ros2 topic info /turtle1/cmd_vel`

> geometry_msgs/msg/Twist → means that in the package `geometry_msgs` there is a `msg` called `Twist`

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

`ros2 topic echo /turtle1/cmd_vel`

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

```
x: 0.0
y: 0.0
z: 2.0
```

To see the type of data of the topic :

`ros2 interface show geometry_msgs/msg/Twistros2 interface show geometry_msgs/msg/Twist`

> 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
> ```

to see the frequency of publication in the topic:

`ros2 topic hz /turtle/pose`

Publication node :

Publish data onto a topic directly from the command line:

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

to see the turtle make a small turn :

`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}}”`

to not stop turning :

`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}}”`

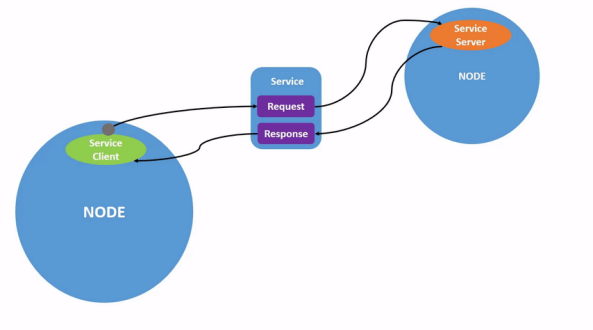
Rate that data its being published :

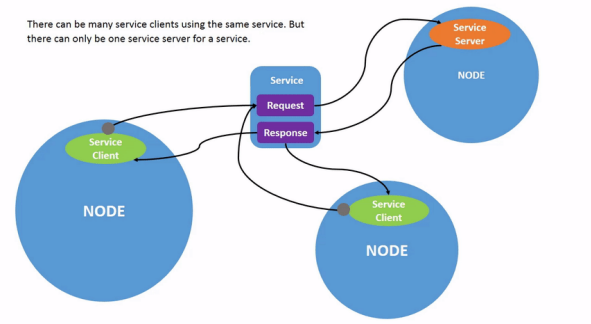
```
ros2 topic hz /turtle1/pose
```

Services :

While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client. (server-client interactions)

obs.: you can’t see services in the rqt_graph





list of services : `ros2 service list` + `-t` and `ros2 service type <service_name>`

to find services with a specific type : `ros2 service find std_srvs/srv/Empty`

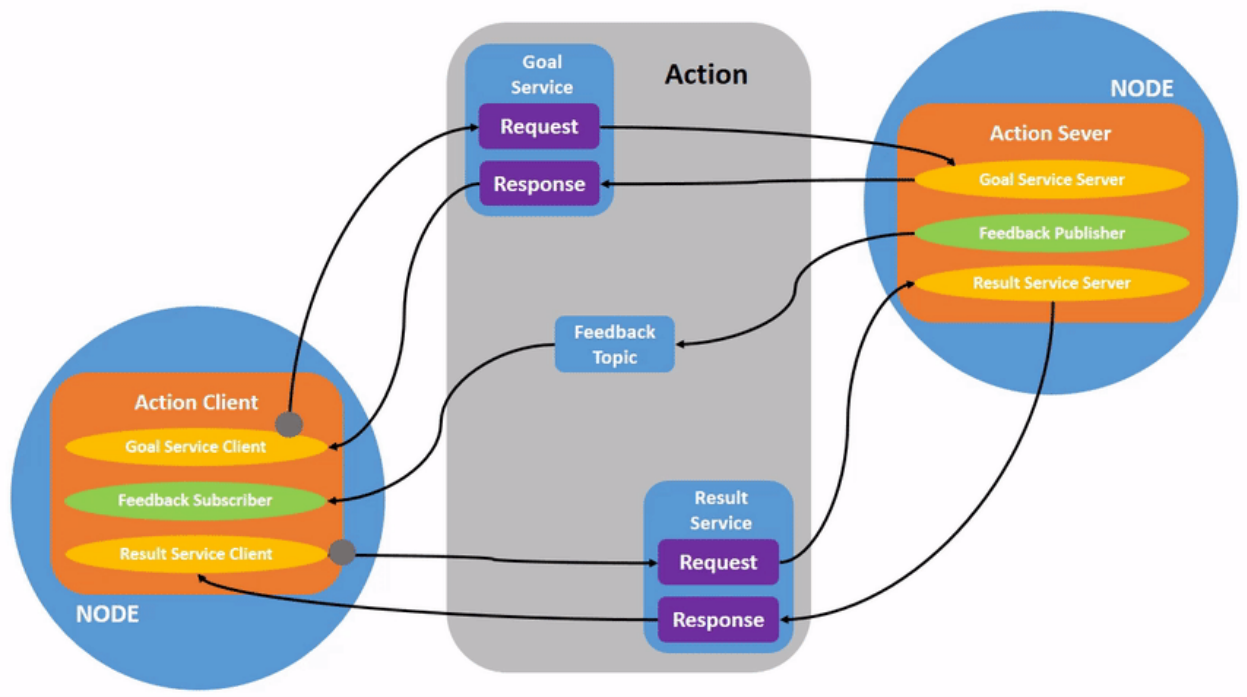
To see the request and response arguments of the `/spawn` service: `ros2 interface show turtlesim/srv/Spawn`

calling a service: `ros2 service call <service_name> <service_type> <arguments>` so :

```
ros2 service call /clear std_srvs/srv/Empty
```

```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}”
```

Actions :



type: `ros2 action list` + `-t`

further introspect the `/turtle1/rotate_absolute` action : `ros2 action info /turtle1/rotate_absolute`

structure of the action type: `ros2 interface show turtlesim/action/RotateAbsolute`

send goal: `ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}”`

feedback: `ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback`

Logs :

To see the logs: `ros2 run rqt_console rqt_console`

levels of severity : Fatal ← Error ← Warn ← Info ← Debug

Python package :

`ros2 pkg create <name_of_the_package>` + `--build-type ament_pyhton` in the folder that we want. Usually, we use as name the (name of the robot)_(functionality).

`ros2 pkg create agilexLimo_controller --build-type ament_python --dependencies rclpy`

…ament + tab (to see the options)

To

compile and build the packages, go to the ros2_ws directory and type (each time you edit the code, type the three commands) :

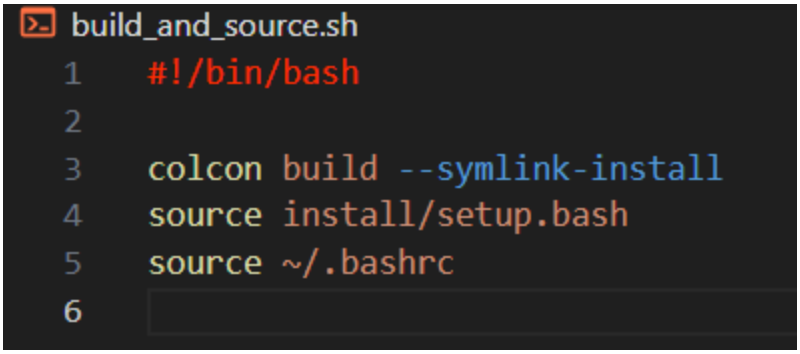
`colcon build --symlink-install` (to be able to execute ros2 commands and update the node)

Also in the source folder (ros2_ws), type :

`source install/setup.bash`

`source ~/.bashrc`

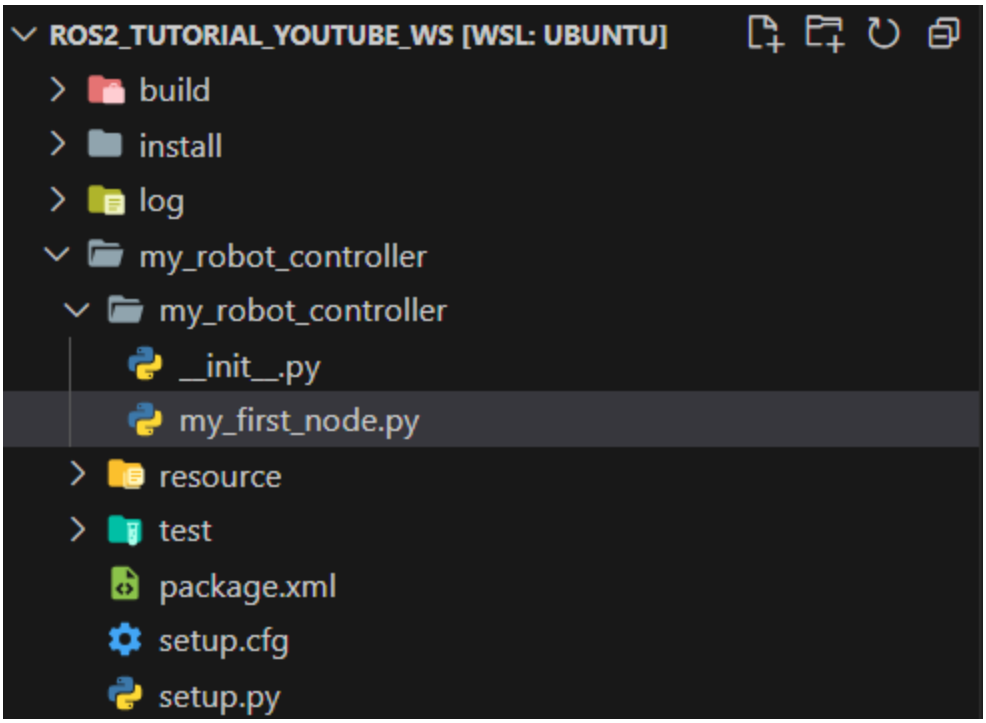
to facilitate the work, we can create a source file to execute these commands (make-it in the ws folder):



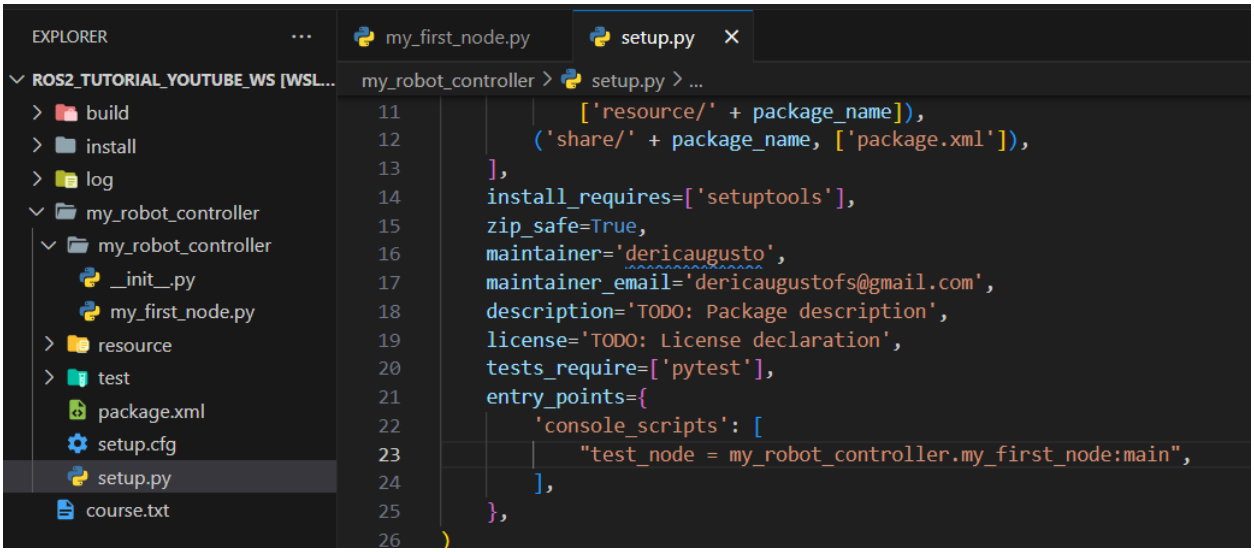
then run `chmod +x my_script.sh` in the bash to make the file executable, and then run it by:

`./build_and_source.sh` or `source build_and_source.sh`

The python files for the nodes will stay in the folder with the name of the package created:



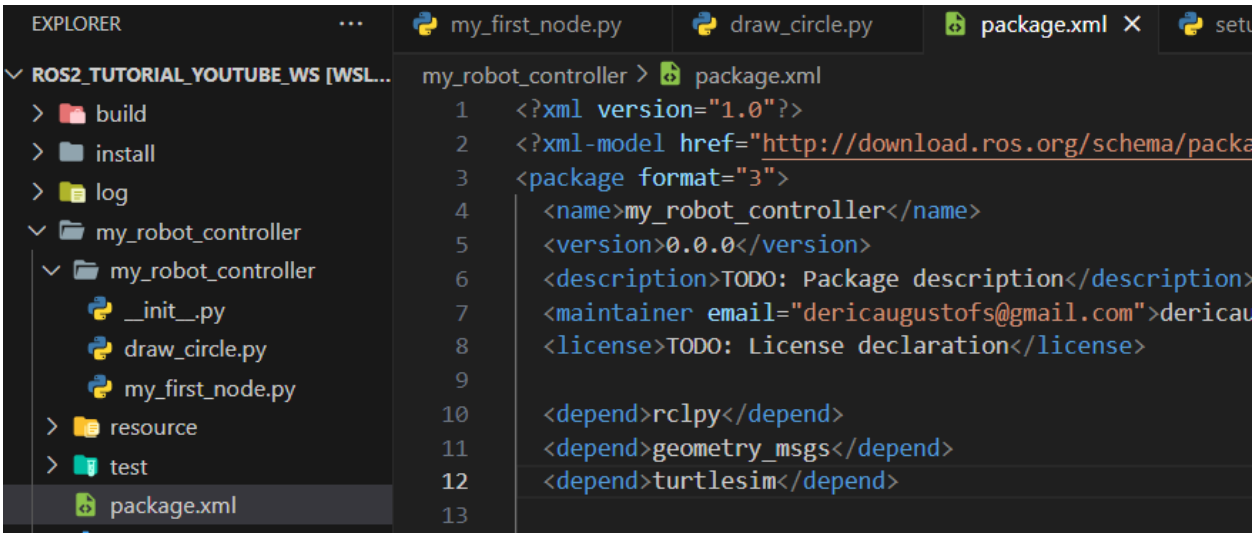
Then add this “test_node=” line in the setup file to access the node everywhere, this will be the executable name to run the file.



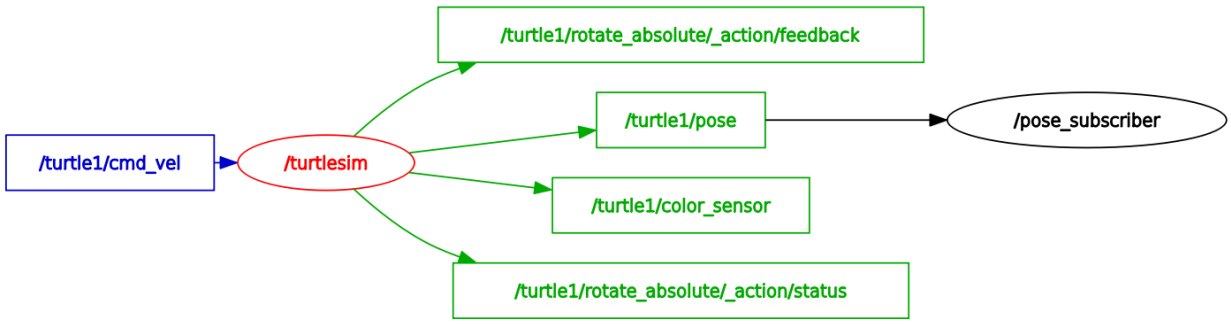
After adding the file do the `source install/setup.bash` command and then you can run with `ros2 run my_robot_controller test_node`

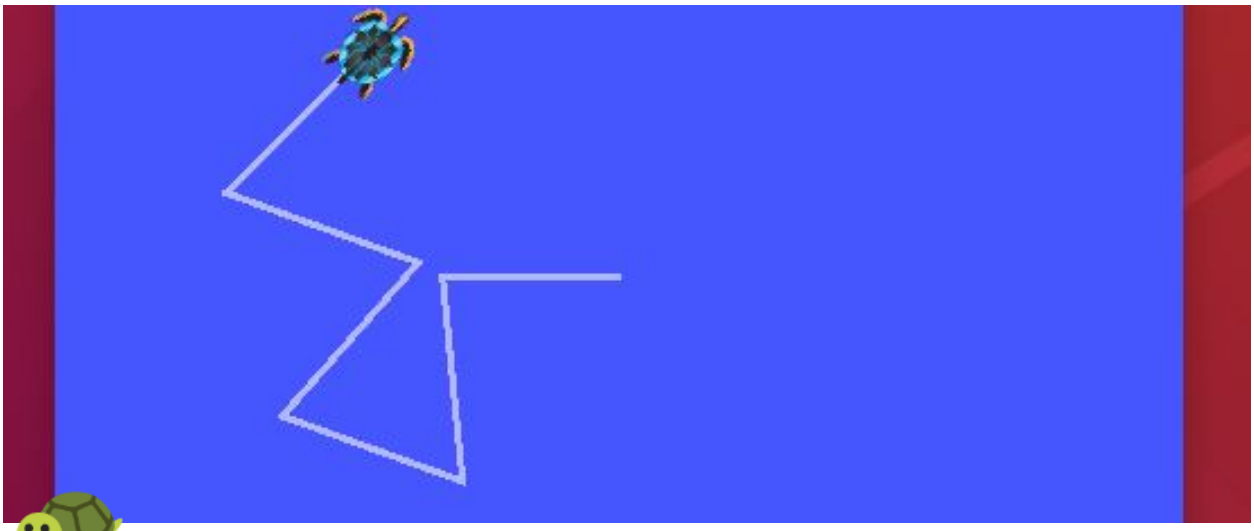
to execute : `ros2 run my_robot_controller test_node`

also add here all the packages that are being used in the python scripts that generates the nodes:



rqt graph of the result:



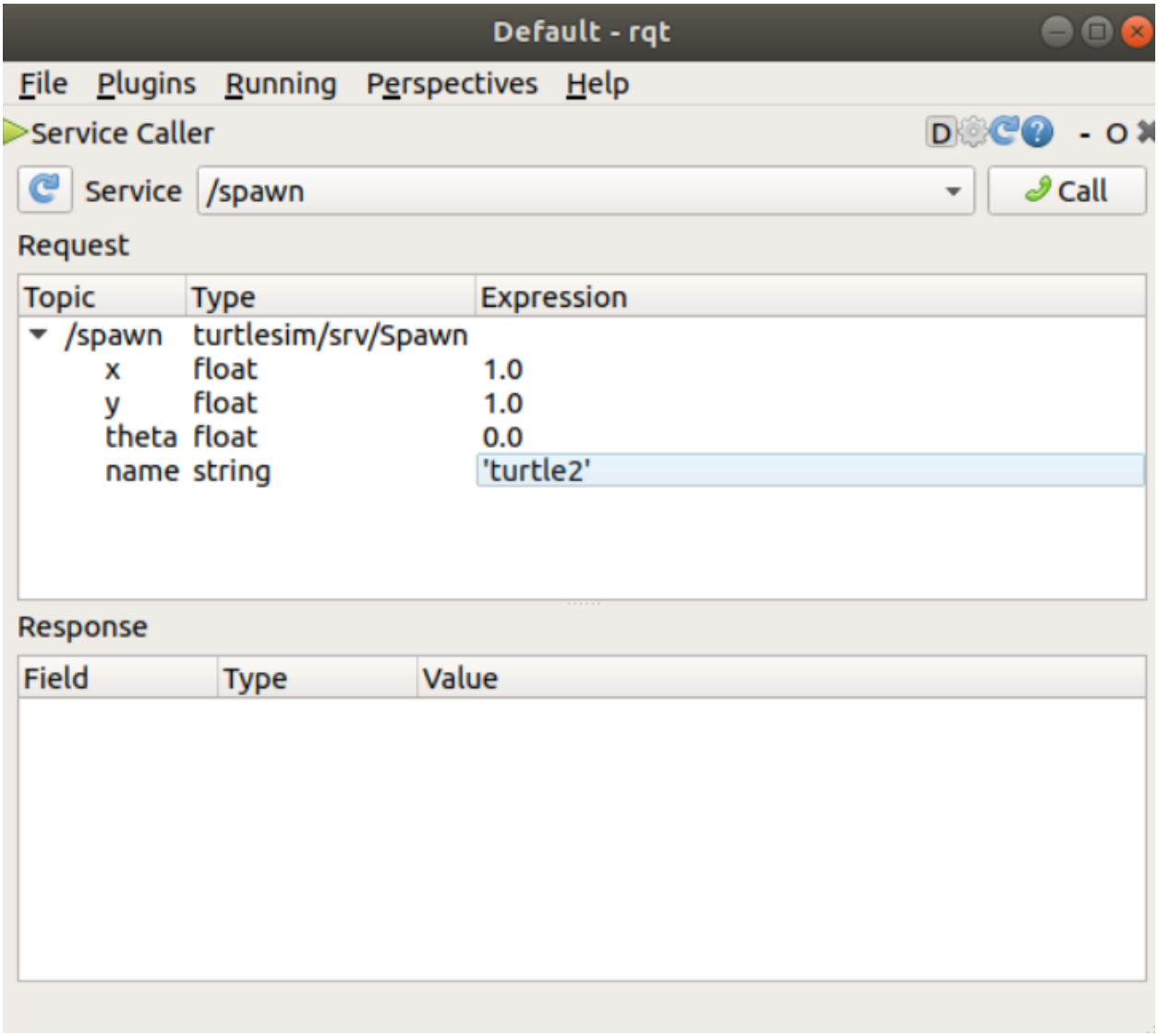


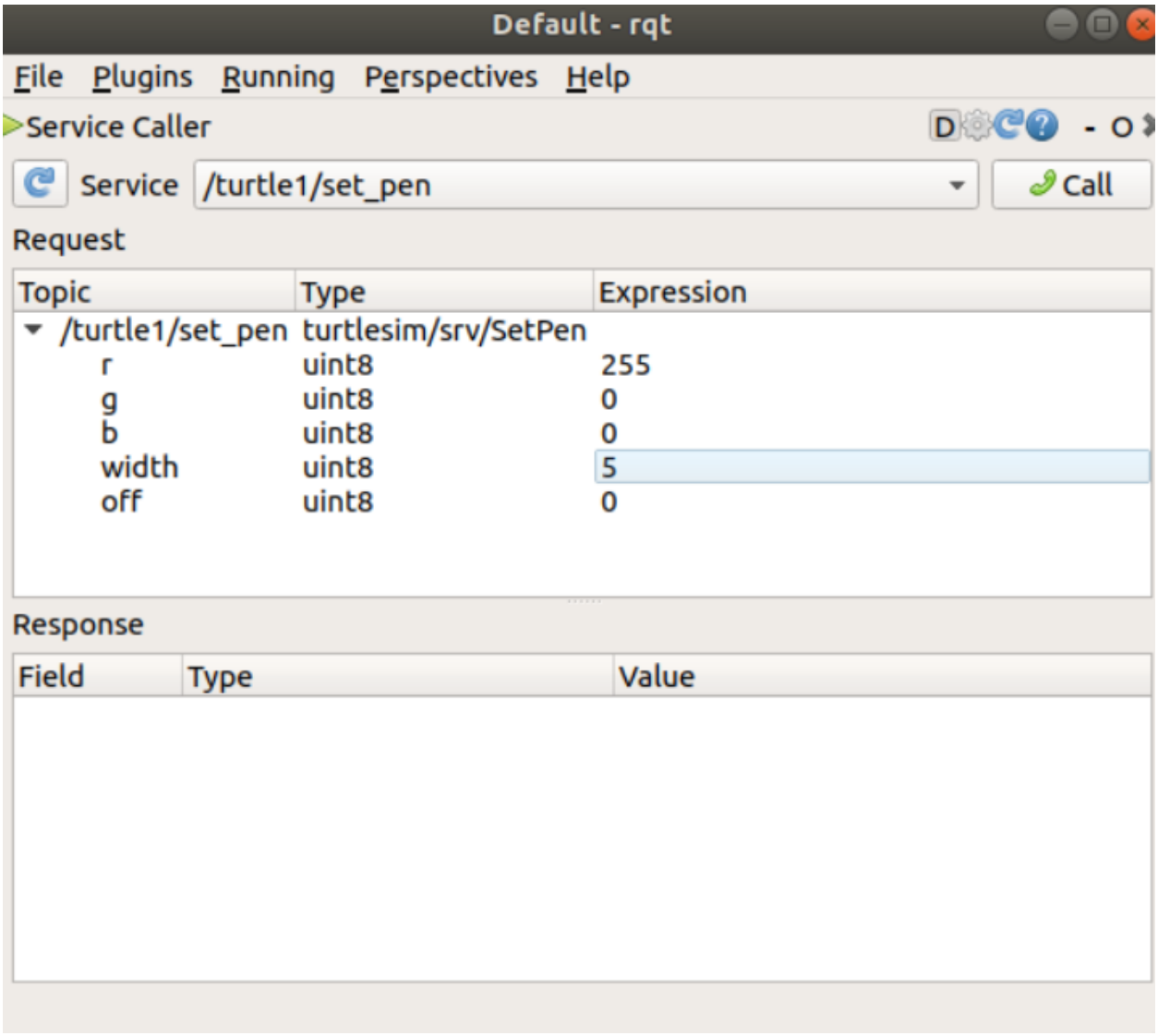
turtlesim

turtlesim works with nodes, so we need to start a node (in each terminal) for :

- turtle window → `ros2 run turtlesim turtlesim_node`
- control the turtle → `ros2 run turtlesim turtle_teleop_key`
- to control a second turtle → `ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel`
- run with parameter file → `ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml`

RQT to spawn a second turtle


Default - rqt
File Plugins Running Perspectives Help
Service Caller
Service /spawn
Call
Request
Topic Type Expression
/spawn turtlesim/srv/Spawn
x float 1.0
y float 1.0
theta float 0.0
name string 'turtle2'
Response
Field Type Value


Default - rqt
File Plugins Running Perspectives Help
Service Caller
Service /turtle1/set_pen
Call
Request
Topic Type Expression
/turtle1/set_pen turtlesim/srv/SetPen
r uint8 255
g uint8 0
b uint8 0
width uint8 5
off uint8 0
Response
Field Type Value