

Contents

0 Setup	2
1 Beginner: CLI tools	3
1.1 Configuring environment	3
1.2 Using <code>turtlesim</code> , <code>ros2</code> and <code>rqt</code>	3
1.3 Understanding nodes	8
1.4 Understanding topics	9
1.5 Understanding services	15
1.6 Understanding parameters	18
1.7 Understanding actions	21
1.8 Using <code>rqt_console</code> to view logs	24
1.9 Launching nodes	26
1.10 Recording and playing back data	27
2 Beginner: Client libraries (C++)	29
2.1 Using colcon to build packages	29
2.2 Creating a workspace	31
2.3 Creating a package	33

ROS2 Homework #1

李馥安
CSIE B13902019

August 13, 2025

0 Setup

Since I am using Arch Linux, I modified the `install.sh` script accordingly. On Arch Linux, Python packages are named with the prefix `python-` and, once installed, they are available for the system's Python 3 interpreter.

```
...
# install python modules
sudo pacman -Sy python-argcomplete \
              python-docker \
              python-prettytable \
              python-yaml
...
```

Activate shell completion:

```
activate-global-python-argcomplete
source /etc/bash_completion.d/python-argcomplete
```

Build needed image:

```
nturt_docker image build nturacing/nturt_ros:host-devel
```

Create a new container:

```
nturt_docker container create 0731 nturacing/nturt_ros:host-devel
host
```

Attach shell into container:

```
nturt_docker container shell 0731
```

1 Beginner: CLI tools

1.1 Configuring environment

Most of the environment configuration is done automatically when starting the Docker container. However, it is important to note that the `ROS_DISTRO` is set to `iron`, so when installing packages, I need to make sure to use the package repositories corresponding to the Iron distribution instead of Humble distribution.

1.2 Using `turtlesim`, `ros2` and `rqt`

1. Install `turtlesim`

I found the password for root and user "docker" is `docker` in `Dockerfile/nturt_ros/host/devel/Dockerfile`.

```
sudo apt update
sudo apt install ros-iron-turtlesim
```

And I found that it has been installed already in the container.

```
[docker@host-devel:~$ sudo apt install ros-iron-turtlesim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ros-iron-turtlesim is already the newest version (1.6.1-1jammy.20240213.160912).
ros-iron-turtlesim set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 187 not upgraded.
```

Double check whether the package is installed:

```
ros2 pkg executables turtlesim
```

```
[docker@host-devel:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

2. Start `turtlesim`

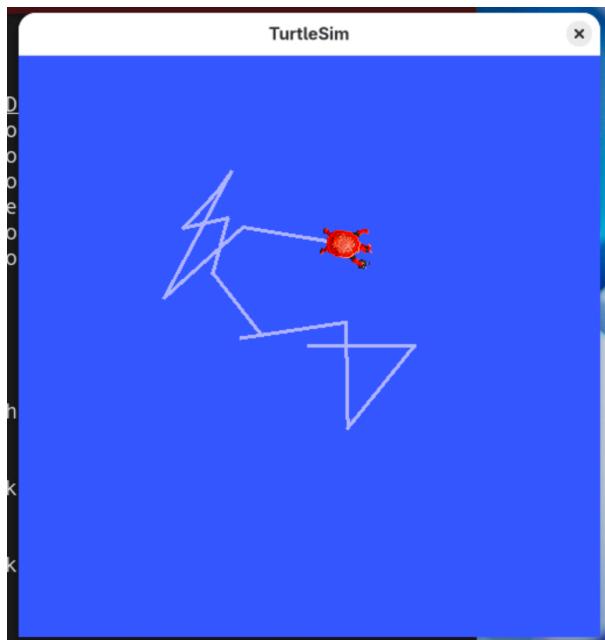
```
ros2 run turtlesim turtlesim_node
```

```
[docker@host-devel:~$ ros2 run turtlesim turtlesim_node
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-docker'
[INFO] [1754039084.932637403] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1754039084.937605542] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

3. Use turtlesim

```
ros2 run turtlesim turtle_teleop_key
```

```
docker@host-devel:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```



By using the `list` subcommands, I can see the nodes, topics, services, and actions.

```
ros2 node list
ros2 topic list
ros2 service list
ros2 action list
```

```
docker@host-devel:~$ ros2 node list
/teleop_turtle
/turtlesim
docker@host-devel:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
docker@host-devel:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/get_type_description
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/get_type_description
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
docker@host-devel:~$ ros2 action list
/turtle1/rotate_absolute
```

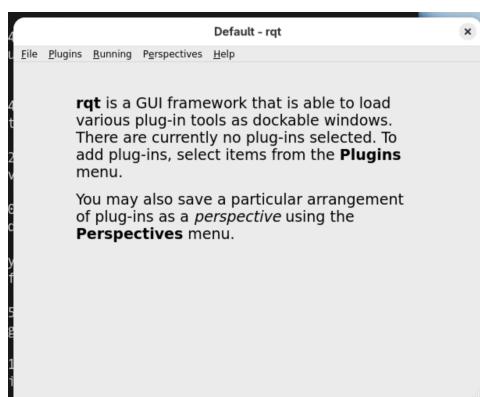
4. Install rqt

Similarly, after checking, the `ros2-iron-rqt` packages has been installed in the container.

Run `rqt`:

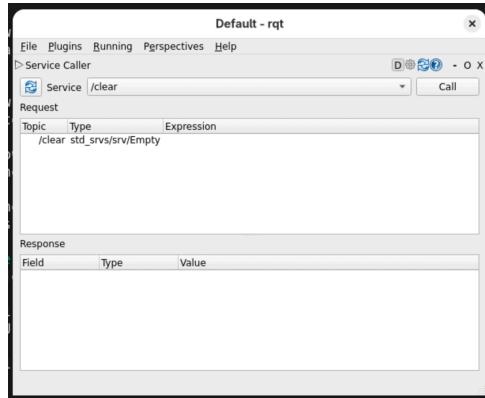
```
rqt
```

```
docker@host-devel:~$ rqt
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-docker'
```

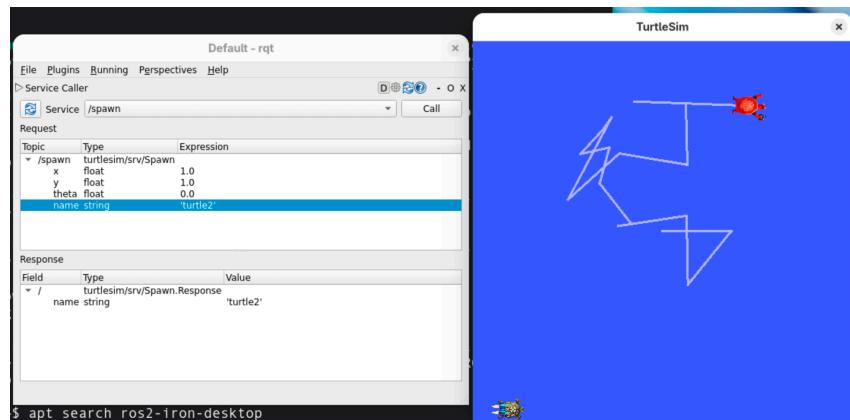


5. Use rqt

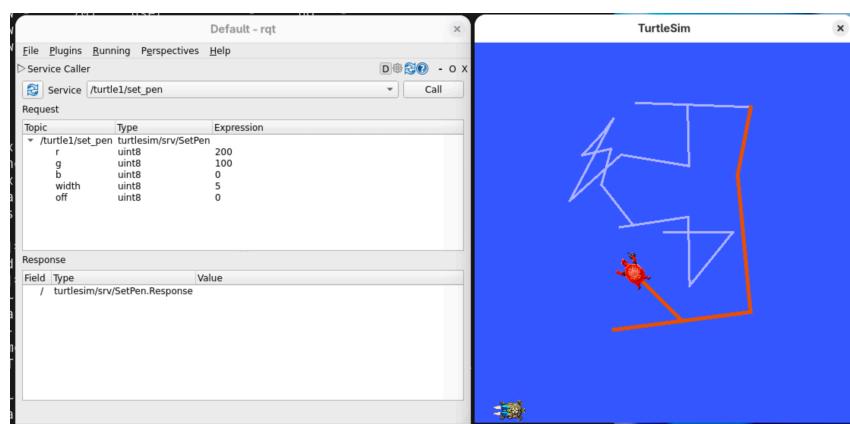
Select Plugins > Services > Service Caller:



(a) Try the spawn service



(b) Try the set_pen service



6. Remapping

In order to control `turtle2`, I can't use the same command before, because it will also control `turtle1` too. Instead, use the new command to remap the topic `cmd_vel`:

```
ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/
  cmd_vel:=turtle2/cmd_vel
```

```
docker@host-devel:~$ ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
Reading from keyboard
-----
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```



7. Close turtlesim

Use `Ctrl+C` and `q` to close the terminals.

1.3 Understanding nodes

1. ros2 run

To launch an executable from a package:

```
ros2 run <package> <executable>
```

For instance:

```
ros2 run turtlesim turtlesim_node
```

And I got same result as the previous section got.

2. ros2 node list

Show the running nodes:

```
ros2 node list
```

- Remapping

Allow me to reassign default node properties, e.g. node name(by using `--remap`):

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node: myturtle
```

There will be a new node:

```
docker@host-devel:~$ ros2 node list
/myturtle
/teleop_turtle
/turtlesim
```

3. ros2 node info

```
ros2 node info /myturtle
```

```
docker@host-devel:~$ ros2 node info /myturtle
/myturtle
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /myturtle/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /myturtle/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /myturtle/get_parameters: rcl_interfaces/srv/GetParameters
    /myturtle/get_type_description: type_description_interfaces/srv/GetTypeDescription
    /myturtle/list_parameters: rcl_interfaces/srv/ListParameters
    /myturtle/set_parameters: rcl_interfaces/srv/SetParameters
    /myturtle/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  Service Clients:
  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

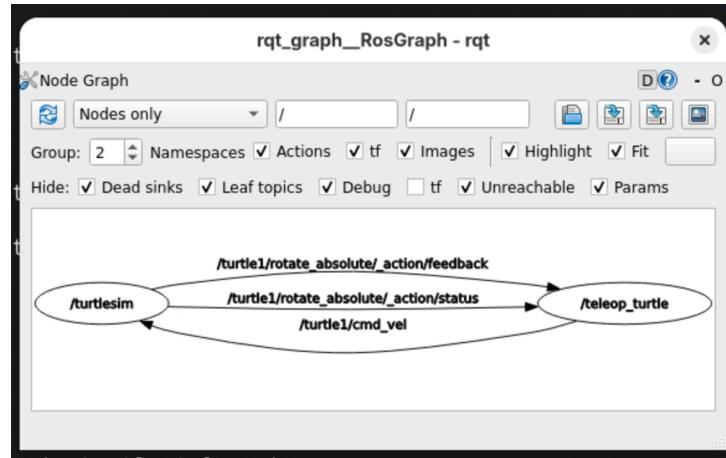
1.4 Understanding topics

1. Setup

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtlesim_teleop_key
```

2. rqt_graph

```
rqt_graph
```



We can discover that `/teleop_turtle` publishes data to the topic `/turtle1/cmd_vel`, and `/turtlesim` is subscribed to it.

3. ros2 topic list

```
ros2 topic list
```

Return currently active topics: With the flag `-t`, I get more information of topic

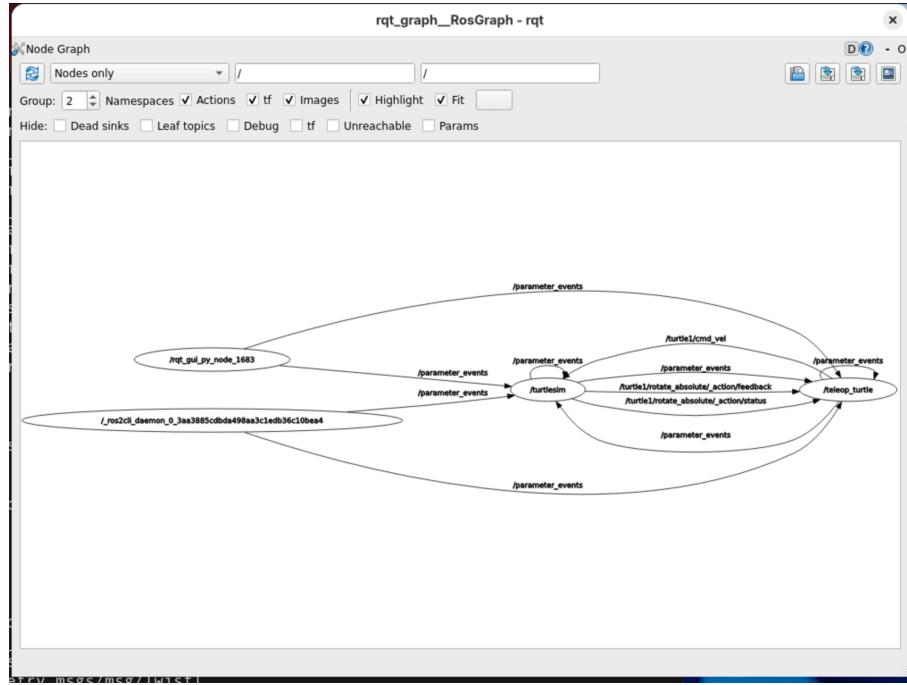
```
docker@host-devel:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

type appended in bracket.

```
ros2 topic list -t
```

```
docker@host-devel:~$ ros2 topic list -t
/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]
```

Uncheck all boxes under Hider can see all topics on `rqt_graph`:



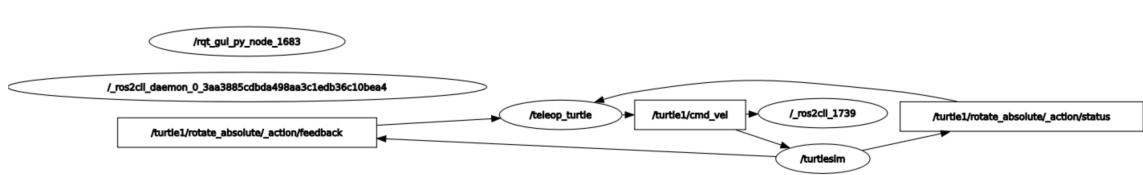
4. ros2 topic echo

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

And we can see the data published to the topic `/turtle1/cmd_vel`:

```
docker@host-devel:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: -2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Uncheck the Debug box on `rqt_graph`, we can see that `/_ros2cli_1739` is created by previous command, and there are two nodes subscribed to `/turtle1/cmd_vel`:



5. ros2 topic info

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

```
docker@host-devel:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
```

It returns the information including the amount of publishers and subscriptions of `/turtle1/cmd_vel`.

6. ros2 interface show

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

It shows the detail of the type "geometry_msgs/msg/Twist".

```
docker@host-devel:~$ ros2 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
```

7. ros2 topic pub

I can directly publish a data to a topic by command:

```
ros2 topic pub /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}}"
```

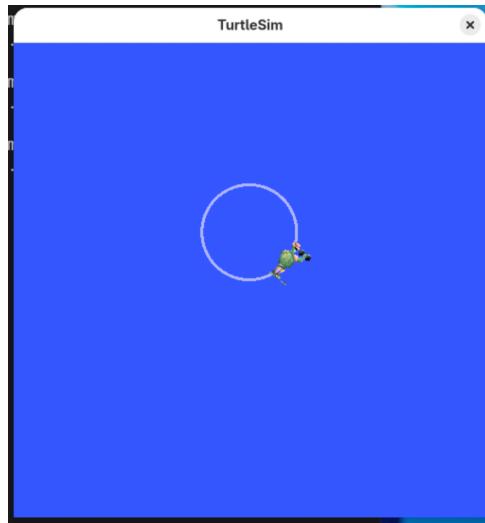
With no other options, it will publish in a steady stream 1Hz.

```
docker@host-devel:~$ ros2 topic pub /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}}"
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))

publishing #2: 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))

publishing #3: 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))

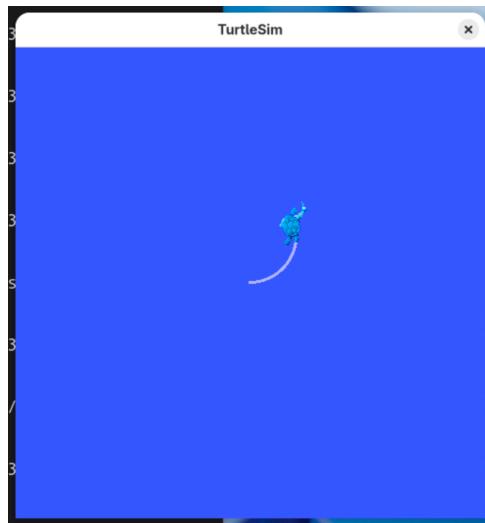
publishing #4: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.
0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8)}
```



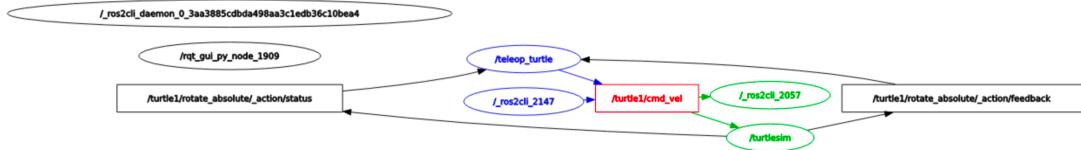
By using `--once`(publish one message then exit) and `-w 2`(wait for 2 matching subscriptions):

```
ros2 topic pub /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 -w 2
```

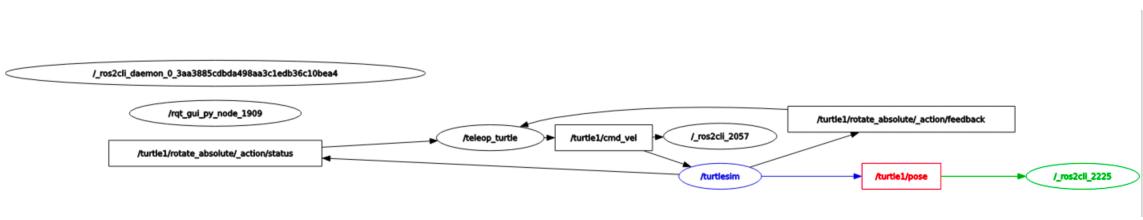
```
docker@host-devel:~$ ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x:  
  2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}" --once -w 2  
publisher: beginning loop  
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.  
0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))  
docker@host-devel:~$
```



Refresh the `rqt_graph`, and we can see that the node `/_ros2cli_2147(ros2 topic pub...)` also published data to the topic `/turtle1/cmd_vel`:



Run `echo` on the `pose` topic and recheck `rqt_graph`: We can see that `/turtlesim`



also publishing to the topic `/turtle1/pose`, which the new `echo` node (`/_ros2cli_2225`) subscribed to.

8. ros2 topic hz

We can see the rate at which data is published:

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

```
docker@host-devel:~$ ros2 topic hz /turtle1/pose
average rate: 62.509
    min: 0.015s max: 0.017s std dev: 0.00055s window: 64
```

9. ros2 topic bw

To view the bandwidth of a topic:

```
ros2 topic bw /turtle1/pose
```

```
^Cdocker@host-devel:~$ ros2 topic bw /turtle1/pose
Subscribed to [/turtle1/pose]
1.51 KB/s from 57 messages
    Message size mean: 0.02 KB min: 0.02 KB max: 0.02 KB
1.52 KB/s from 100 messages
    Message size mean: 0.02 KB min: 0.02 KB max: 0.02 KB
```

10. ros2 topic find Give the topics available for specific type of messages:

```
ros2 topic find geometry_msgs/msg/Twist
```

```
docker@host-devel:~$ ros2 topic find geometry_msgs/msg/Twist
/turtle1/cmd_vel
```

11. Clean up

Use Ctrl+C to clean up all terminals.

1.5 Understanding services

1. Setup

```
ros2 run turtlesim turtlesim_node  
ros2 run turtlesim turtle_teleop_key
```

2. ros2 service list

To see current active services:

```
ros2 service list
```

```
docker@host-devel:~$ ros2 service list  
/clear  
/kill  
/reset  
/spawn  
/teleop_turtle/describe_parameters  
/teleop_turtle/get_parameter_types  
/teleop_turtle/get_parameters  
/teleop_turtle/get_type_description  
/teleop_turtle/list_parameters  
/teleop_turtle/set_parameters  
/teleop_turtle/set_parameters_atomically  
/turtle1/set_pen  
/turtle1/teleport_absolute  
/turtle1/teleport_relative  
/turtlesim/describe_parameters  
/turtlesim/get_parameter_types  
/turtlesim/get_parameters  
/turtlesim/get_type_description  
/turtlesim/list_parameters  
/turtlesim/set_parameters  
/turtlesim/set_parameters_atomically
```

To see the type of a service:

```
ros2 service type <service>
```

e.g. the type of /kill:

```
docker@host-devel:~$ ros2 service type /kill  
turtlesim/srv/Kill
```

- ros2 service list -t

Similar to `topic`, we can add the option `-t`, to see the types of all services.

```
ros2 service list -t
```

```
docker@host-devel:~$ ros2 service list -t
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/reset [std_srvs/srv/Empty]
/spawn [turtlesim/srv/Spawn]
/teleop_turtle/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/teleop_turtle/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/teleop_turtle/get_parameters [rcl_interfaces/srv/GetParameters]
/teleop_turtle/get_type_description [type_description_interfaces/srv/GetTypeDescription]
/teleop_turtle/list_parameters [rcl_interfaces/srv/ListParameters]
/teleop_turtle/set_parameters [rcl_interfaces/srv/SetParameters]
/teleop_turtle/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/turtle1/set_pen [turtlesim/srv/SetPen]
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]
/turtle1/teleport_relative [turtlesim/srv/TeleportRelative]
/turtlesim/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/turtlesim/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/turtlesim/get_parameters [rcl_interfaces/srv/GetParameters]
/turtlesim/get_type_description [type_description_interfaces/srv/GetTypeDescription]
/turtlesim/list_parameters [rcl_interfaces/srv/ListParameters]
/turtlesim/set_parameters [rcl_interfaces/srv/SetParameters]
/turtlesim/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
```

3. ros2 service find

Find all services of a specific type:

```
ros2 service find std_srvs/srv/Empty
```

```
docker@host-devel:~$ ros2 service find std_srvs/srv/Empty
/clear
/reset
```

4. ros2 interface show

See the detail of a type:

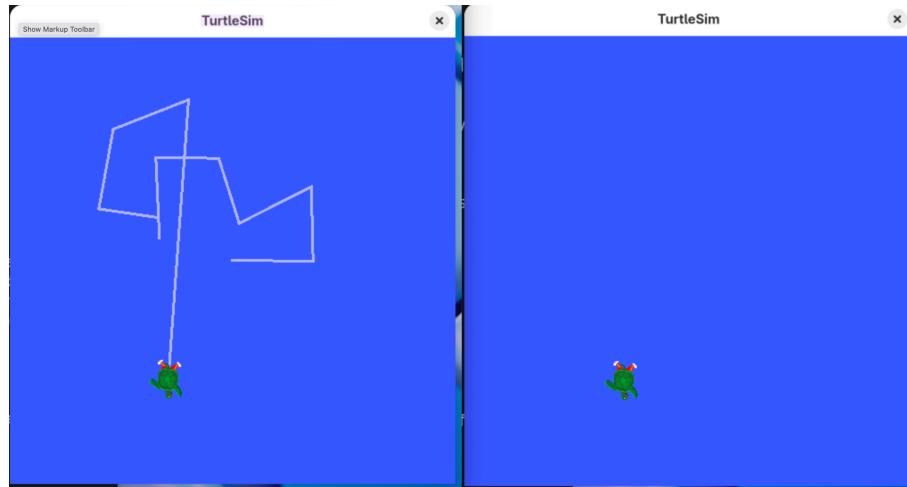
```
ros2 interface show turtlesim/srv/Spawn
```

```
docker@host-devel:~$ ros2 interface show std_srvs/srv/Empty
---
docker@host-devel:~$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
---
string name
```

5. ros2 service call

Call a service:

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



```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 3, y: 4, theta: 0.5, name: 'hehe'}
```

```
docker@host-devel:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 3, y: 4, theta: 0.5, name: 'hehe'}"
requester: making request: turtlesim.srv.Spawn_Request(x=3.0, y=4.0, theta=0.5, name='hehe')

response:
turtlesim.srv.Spawn_Response(name='hehe')
```



1.6 Understanding parameters

1. Setup

```
ros2 run turtlesim turtlesim_node  
ros2 run turtlesim turtle_teleop_key
```

2. ros2 param list

To see each node's parameters:

```
ros2 param list
```

```
docker@host-devel:~$ ros2 param list  
/teleop_turtle:  
    qos_overrides./parameter_events.publisher.depth  
    qos_overrides./parameter_events.publisher.durability  
    qos_overrides./parameter_events.publisher.history  
    qos_overrides./parameter_events.publisher.reliability  
    scale_angular  
    scale_linear  
    start_type_description_service  
    use_sim_time  
/turtlesim:  
    background_b  
    background_g  
    background_r  
    holonomic  
    qos_overrides./parameter_events.publisher.depth  
    qos_overrides./parameter_events.publisher.durability  
    qos_overrides./parameter_events.publisher.history  
    qos_overrides./parameter_events.publisher.reliability  
    start_type_description_service  
    use_sim_time
```

3. ros2 get param

To see the type and value of a node's parameter:

```
ros2 get param /turtlesim background_g
```

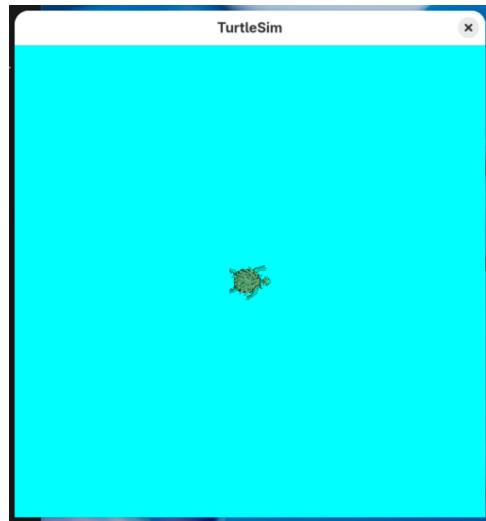
```
docker@host-devel:~$ ros2 param get /turtlesim background_g  
Integer value is: 86
```

4. ros2 set param

To set a parameter of a node:

```
ros2 set param /turtlesim background_g 255
```

```
docker@host-devel:~$ ros2 param set /turtlesim background_g 255
Set parameter successful
```



5. ros2 param dump

To see all parameters of a node:

```
ros2 param dump /turtlesim
```

and it can also output to a file by appending > <filename>.

```
docker@host-devel:~$ ros2 param dump /turtlesim
/turtlesim:
  ros_parameters:
    background_b: 255
    background_g: 255
    background_r: 69
    holonomic: false
    qos_overrides:
      /parameter_events:
        publisher:
          depth: 1000
          durability: volatile
          history: keep_last
          reliability: reliable
    start_type_description_service: true
    use_sim_time: false
```

6. ros2 param load

To load a file into a node:

```
ros2 param load /turtlesim turtlesim.yaml
```

```
docker@host-devel:~$ ros2 param load /turtlesim turtlesim.yaml
Set parameter background_b successful
Set parameter background_g successful
Set parameter background_r successful
Set parameter holonomic successful
Set parameter qos_overrides./parameter_events.publisher.depth failed: parameter
'qos_overrides./parameter_events.publisher.depth' cannot be set because it is re
ad-only
Set parameter qos_overrides./parameter_events.publisher.duration failed: param
eter 'qos_overrides./parameter_events.publisher.duration' cannot be set becaus
e it is read-only
Set parameter qos_overrides./parameter_events.publisher.history failed: paramete
r 'qos_overrides./parameter_events.publisher.history' cannot be set because it i
s read-only
Set parameter qos_overrides./parameter_events.publisher.reliability failed: para
meter 'qos_overrides./parameter_events.publisher.reliability' cannot be set beca
use it is read-only
Set parameter start_type_description_service failed: parameter 'start_type_descri
ption_service' cannot be set because it is read-only
Set parameter use_sim_time successful
```

7. Load parameter file on node startup:

To load a parameter file to a new node:

```
ros2 run turtlesim turtlesim_node --ros-args --params-file
turtlesim.yaml
```

```
docker@host-devel:~$ ros2 run turtlesim turtlesim_node --ros-args --params-file
turtlesim.yaml
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-docker'
[INFO] [1754099661.786248715] [turtlesim]: Starting turtlesim with node name /tu
rtlesim
[INFO] [1754099661.808473231] [turtlesim]: Spawning turtle [turtle1] at x=[5.544
445], y=[5.544445], theta=[0.000000]
```

1.7 Understanding actions

1. Setup

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

2. Use actions

Use the operation in the `turtle_teleop_key`'s terminal, and observe the terminal of the node `turtlesim`:

```
[WARN] [1754125973.926366485] [turtlesim]: Rotation goal received before a previous goal finished. Aborting previous goal
[INFO] [1754125975.638636732] [turtlesim]: Rotation goal completed successfully
[INFO] [1754125979.254624339] [turtlesim]: Rotation goal canceled
```

3. ros2 node info

To see the detail of a node:

```
ros2 node info /turtlesim
```

```
[docker@host-devel:~$ ros2 node info /turtlesim
/turtlesim
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/color_sensor: turtlesim/msg/Color
  /turtle1/pose: turtlesim/msg/Pose
Service Servers:
  /clear: std_srvs/srv/Empty
  /kill: turtlesim/srv/Kill
  /reset: std_srvs/srv/Empty
  /spawn: turtlesim/srv/Spawn
  /turtle1/set_pen: turtlesim/srv/SetPen
  /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
  /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
  /turtlesim/get_type_description: type_description_interfaces/srv/GetTypeDescription
  /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
  /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
  /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
Service Clients:
Action Servers:
  /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
Action Clients:
```

4. ros2 action list

To see all active actions:

```
ros2 action list
```

Similarly, `-t` for type:

```
ros2 action list -t
```

```
[docker@host-devel:~$ ros2 action list
/turtle1/rotate_absolute
[docker@host-devel:~$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]]
```

5. ros2 action info: To see more detail about an action:

```
ros2 action info /turtle1/rotate_absolute
```

```
[docker@host-devel:~$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
  /teleop_turtle
Action servers: 1
  /turtlesim
```

6. ros2 interface show: To see the interface of a type:

```
ros2 interface show turtlesim/action/RotateAbsolute
```

```
[docker@host-devel:~$ ros2 interface show turtlesim/action/RotateAbsolute
# The desired heading in radians
float32 theta
---
# The angular displacement in radians to the starting position
float32 delta
---
# The remaining rotation in radians
float32 remaining
```

7. ros2 action send_goal To send an action goal from the command line directly:

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

```
[docker@host-devel:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute]
  "{theta: 1.57}"
Waiting for an action server to become available...
Sending goal:
  theta: 1.57

Goal accepted with ID: e9a2d7ad62b4464284d999cd6e5f6419

Result:
  delta: -2.7839999198913574

Goal finished with status: SUCCEEDED
```

Add --feedback to see feedback:

```
docker@host-devel:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute
  "{theta: 1.58}" --feedback
Waiting for an action server to become available...
Sending goal:
  theta: 1.58

Feedback:
  remaining: 0.02799999713897705

Goal accepted with ID: ef1e9c3d49b043f5bb0573c881280d47

Feedback:
  remaining: 0.012000083923339844

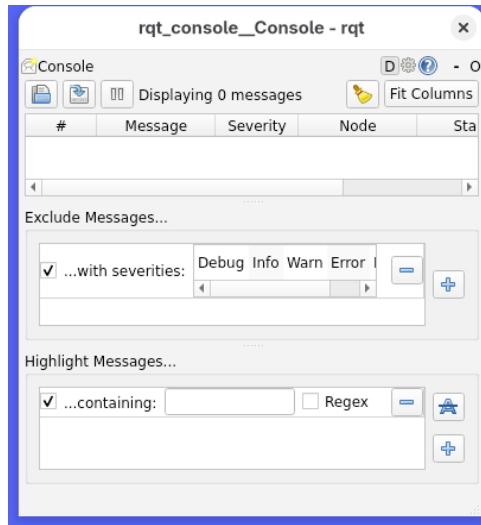
Result:
  delta: -0.01600000075995922

Goal finished with status: SUCCEEDED
```

1.8 Using rqt_console to view logs

1. Setup

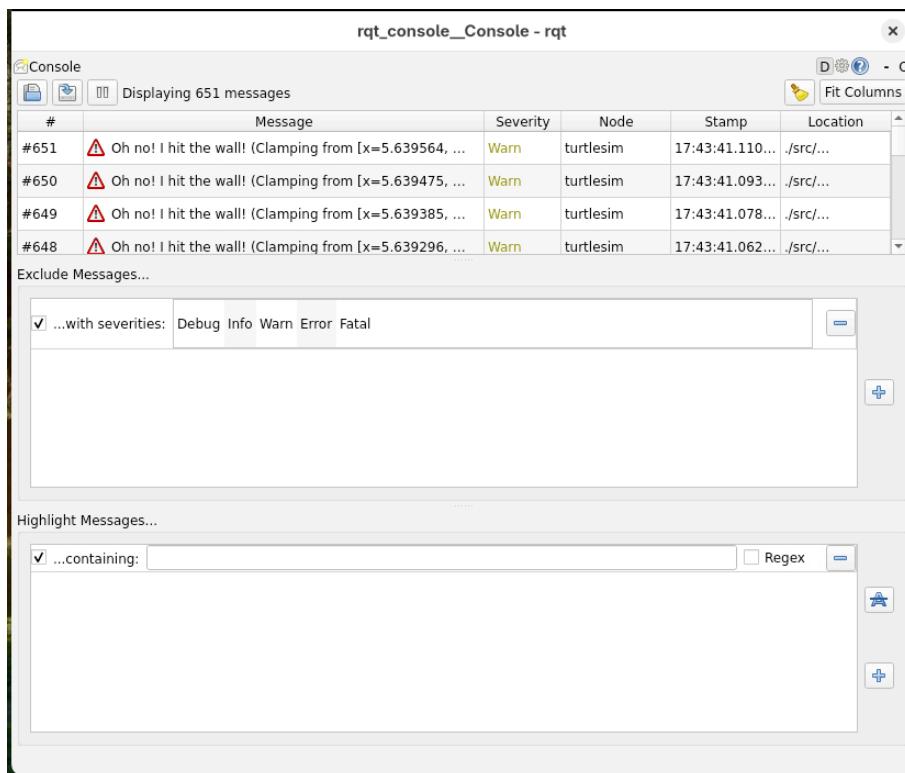
```
ros2 run rqt_console rqt_console
```



2. Messages on rqt_console

```
ros2 topic pub -r 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: 0.0}"
```

And we can see warning on the console:



We can use `--log-level` to set the logger level:

```
ros2 run turtlesim turtlesim_node --ros-args --log-level WARN
```

And messages with level lower than [WARN] ([INFO]) cannot be seen.

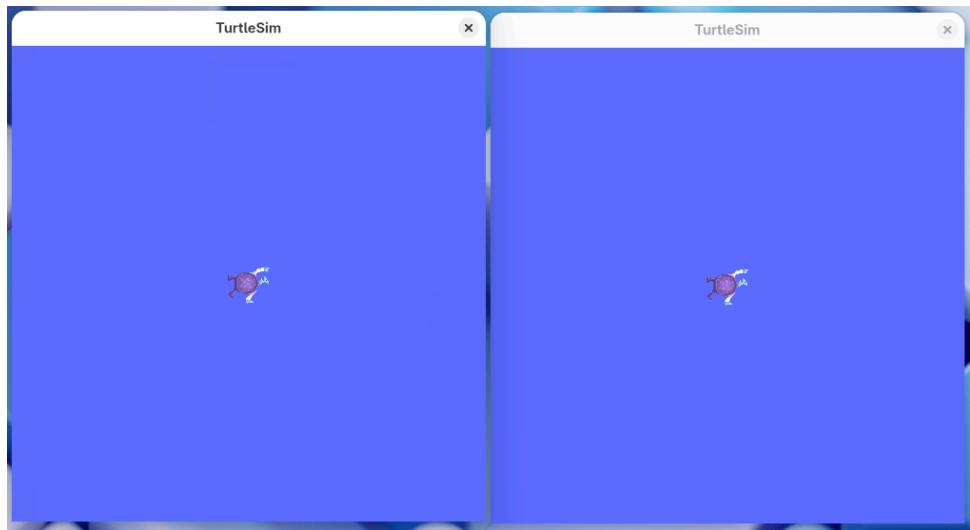
```
docker@host-devel:~$ ros2 run turtlesim turtlesim_node --ros-args --log-level WARN
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-docker'
[WARN] [1754128151.515655151] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.646716, y=-0.018107])
[WARN] [1754128151.531215837] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.641706, y=-0.031605])
[WARN] [1754128151.547063430] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.636696, y=-0.031605])
[WARN] [1754128151.563732201] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.631686, y=-0.031605])
[WARN] [1754128151.579698771] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.626676, y=-0.031605])
[WARN] [1754128151.595036265] [turtlesim]: Oh no! I hit the wall! (Clamping from [x=2.621667, y=-0.031605])
```

1.9 Launching nodes

- Running a Luanch File

```
ros2 launch turtlesim multisim.launch.py
```

We will see two turtlesim nodes:



1.10 Recording and playing back data

1. Setup

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

And also create a directory to store recordings.

```
mkdir bag_files
```

2. Choose a topic Use:

```
ros2 topic list
```

We can choose a topic we want to record. Additionally, we can use the `topic echo` command to show the messages published to the topic.

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

3. ros2 bag record

(a) Record a single topic

```
ros2 bag record /turtle1/cmd_vel
```

```
docker@host-devel:~/bag_files$ ros2 bag record /turtle1/cmd_vel
[INFO] [1754138003.891950254] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1754138003.906152615] [rosbag2_recorder]: Listening for topics...
[INFO] [1754138003.906205812] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1754138003.931850936] [rosbag2_recorder]: Subscribed to topic '/turtle1/cmd_vel'
[INFO] [1754138003.931967370] [rosbag2_recorder]: Recording...
[INFO] [1754138003.932207173] [rosbag2_recorder]: All requested topics are subscribed. Stopping discovery...
[INFO] [1754138053.755767642] [rosbag2_cpp]: Writing remaining messages from cache to the bag. It may take a while
[INFO] [1754138053.758234218] [rosbag2_recorder]: Event publisher thread: Exiting
[INFO] [1754138053.758836258] [rosbag2_recorder]: Recording stopped
docker@host-devel:~/bag_files$ ls
rosbag2_2025_08_02-20_33_23
```

(b) Record multiple topics

```
ros2 bag record <topic_1> <topic_2> ...
```

Option `-o` can name the file, and `-a` will record all topics.

4. ros2 bag info

```
ros2 bag info <file_name>
```

```
docker@host-devel:~/bag_files$ ros2 bag info rosbag2_2025_08_02-20_33_23/ ]  
Files:          rosbag2_2025_08_02-20_33_23_0.mcap  
Bag size:      16.1 KiB  
Storage id:    mcap  
Duration:      33.891s  
Start:         Aug  2 2025 20:33:30.374 (1754138010.374)  
End:          Aug  2 2025 20:34:04.266 (1754138044.266)  
Messages:       130  
Topic information: Topic: /turtle1/cmd_vel | Type: geometry_msgs/msg/Twist | Count: 130 | Serialization Format: cdr
```

5. ros2 bag play To play the recorded file (I reset the condition first):

```
ros2 bag play <file_name>
```

```
docker@host-devel:~/bag_files$ ros2 service call /reset std_srvs/srv/Empty ]  
requester: making request: std_srvs.srv.Empty_Request()  
  
response:  
std_srvs.srv.Empty_Response()  
  
docker@host-devel:~/bag_files$ ros2 bag play rosbag2_2025_08_02-20_33_23/ ]  
[INFO] [1754138182.736604377] [rosbag2_player]: Set rate to 1  
[WARN] [1754138182.736770545] [rclcpp]: A zero depth with KEEP_LAST doesn't make sense; no data could be stored. This will be interpreted as SYSTEM_DEFAULT  
[INFO] [1754138182.741805947] [rosbag2_player]: Adding keyboard callbacks.  
[INFO] [1754138182.741832010] [rosbag2_player]: Press SPACE for Pause/Resume  
[INFO] [1754138182.741841453] [rosbag2_player]: Press CURSOR_RIGHT for Play Next Message  
[INFO] [1754138182.741852070] [rosbag2_player]: Press CURSOR_UP for Increase Rate 10%  
[INFO] [1754138182.741860113] [rosbag2_player]: Press CURSOR_DOWN for Decrease Rate 10%  
[INFO] [1754138182.741939352] [rosbag2_player]: Playback until timestamp: -1
```

2 Beginner: Client libraries (C++)

2.1 Using colcon to build packages

1. Prerequisites

Install colcon and ROS2

2. Create a workspace

Create a directory:

```
mkdir -p ~/ros2_ws/src  
cd ~/ros2_ws$
```

3. Add some sources

```
git clone https://github.com/ros2/examples src/examples -b iron
```

4. Build the workspace

```
colcon build --symlink-install
```

Run tests

```
colcon test
```

```
Summary: 22 packages finished [45.0s]  
 10 packages had stderr output: examples_rclpy_executors examples_rclpy_guard_conditions examples_rclpy_minimal_action_client examples_rclpy_minimal_action_server examples_rclpy_minimal_client examples_rclpy_minimal_publisher examples_rclpy_minimal_service examples_rclpy_minimal_subscriber examples_rclpy_pointcloud_publisher launch_testing_examples
```

5. source the environment

```
source install/setup.bash
```

6. Try a demo

```
ros2 run examples_rclcpp_minimal_subscriber
    subscriber_member_function
ros2 run examples_rclcpp_minimal_publisher
    publisher_member_function
```

```
docker@host-devel:~$ ros2 run examples_rclcpp_minimal_publisher publisher_member_function
[INFO] [1754300286.568197686] [minimal_publisher]: Publishing: 'Hello, world! 0'
[INFO] [1754300287.068041561] [minimal_publisher]: Publishing: 'Hello, world! 1'
[INFO] [1754300287.568028513] [minimal_publisher]: Publishing: 'Hello, world! 2'
[INFO] [1754300288.068202192] [minimal_publisher]: Publishing: 'Hello, world! 3'
[INFO] [1754300288.568217054] [minimal_publisher]: Publishing: 'Hello, world! 4'
[INFO] [1754300289.068174325] [minimal_publisher]: Publishing: 'Hello, world! 5'
[INFO] [1754300289.568162086] [minimal_publisher]: Publishing: 'Hello, world! 6'
^C[INFO] [1754300289.633651500] [rclcpp]: signal_handler(signum=2)
docker@host-devel:~/ros2_ws$ ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
[INFO] [1754300286.568873460] [minimal_subscriber]: I heard: 'Hello, world! 0'
[INFO] [1754300287.068308340] [minimal_subscriber]: I heard: 'Hello, world! 1'
[INFO] [1754300287.568520098] [minimal_subscriber]: I heard: 'Hello, world! 2'
[INFO] [1754300288.068676554] [minimal_subscriber]: I heard: 'Hello, world! 3'
[INFO] [1754300288.568671336] [minimal_subscriber]: I heard: 'Hello, world! 4'
[INFO] [1754300289.068648245] [minimal_subscriber]: I heard: 'Hello, world! 5'
[INFO] [1754300289.568585636] [minimal_subscriber]: I heard: 'Hello, world! 6'
```

7. Setup colcon_cd

```
echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc
echo "export _colcon_cd_root=/opt/ros/iron/" >> ~/.bashrc
```

2.2 Creating a workspace

1. Source ROS 2 environment

Done in previous section.

2. Create a new directory

Done in previous section.

3. Clone a sample repo

In the `ros2_ws/src` directory:

```
git clone https://github.com/ros/ros_tutorials.git -b iron
```

4. Resolve dependencies

In `ros_ws` directory:

```
rosdep install -i --from-path src --rosdistro iron -y
```

```
[docker@host-devel:~/ros2_ws$ rosdep install -i --from-path src --rosdistro iron -y  
#All required rosdeps installed successfully]
```

5. Build the workspace with colcon

```
colcon build --parallel-workers 1
```

Using option `--parallel-workers 1` to prevent my memory run out of space.

6. source the overlay

```
source install/local_setup.bash
```

7. Modify the overlay

Find the file `turtle_fram.cpp` in

`/ros2_ws/src/ros_tutorials/turtlesim/src`. Find the function `setWindowTitle()` and change the value.

```
ros2 run turtlesim turtlesim_node
```

To see underlay, open a new terminal:

```
ros2 run turtlesim turtlesim_node
```



2.3 Creating a package

Before proceeding to this step, I went through the official CMake tutorial to make sure I understand the whole concept.

1. Create a package

In the directory `/ros2_ws/src`:

```
ros2 pkg create --build-type ament_cmake --license Apache-2.0 --
    node-name my_node my_package
```

```
docker@host-devel:~/ros2_ws/src$ ros2 pkg create --build-type ament_cmake --license Apache-2.0 --node-name my_node my_package
going to create a new package
package name: my_package
destination directory: /home/docker/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['docker <docker@todo.todo>']
licenses: ['Apache-2.0']
build type: ament_cmake
dependencies: []
node_name: my_node
creating folder ./my_package
creating ./my_package/package.xml
creating source and include folder
creating folder ./my_package/src
creating folder ./my_package/include/my_package
creating ./my_package/CMakeLists.txt
creating ./my_package/src/my_node.cpp
docker@host-devel:~/ros2_ws/src$
```

2. Build a package

```
colcon build --packages-select my_package
```

```
docker@host-devel:~/ros2_ws/src$ colcon build --packages-select my_package
Starting >>> my_package
Finished <<< my_package [0.14s]

Summary: 1 package finished [0.53s]
```

3. Source the setup file

```
source install/local_setup.bash
```

4. Use the package

```
ros2 run my_package my_node
```

```
docker@host-devel:~/ros2_ws/src$ ros2 run my_package my_node
hello world my_package package
```

5. Examine package contents

```
my_package/
├── CMakeLists.txt
├── LICENSE
└── include
    └── my_package
├── package.xml
└── src
    └── my_node.cpp

3 directories, 4 files
```

6. Customize package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_package</name>
  <version>0.0.0</version>
  <description>Beginner client libraries tutorials practice package</description>
  <maintainer email="b13902019@csie.ntu.edu.tw"></maintainer>
  <license>Apache License 2.0</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```