CSCI 598B: Robotic Mapping & Localization

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lab06: ROS 2

*Courtesy of Francisco Martin Rico and others

Outline

- **ROS 2**
  - What is ROS, and why?
  - Installation
  - ROS structure
  - Writing a node
  - ROS in terminal
  - Package example
  - Simulation
  - Transforms (tf2_ros)
  - ROS bag
  - References

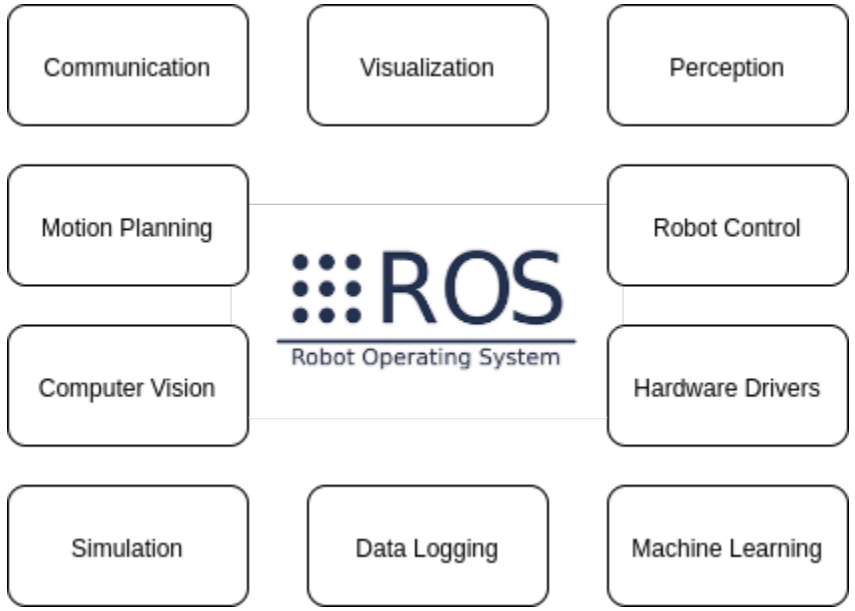
Crazyflie

Clearpath Husky UGV

Pepper

Why ROS?

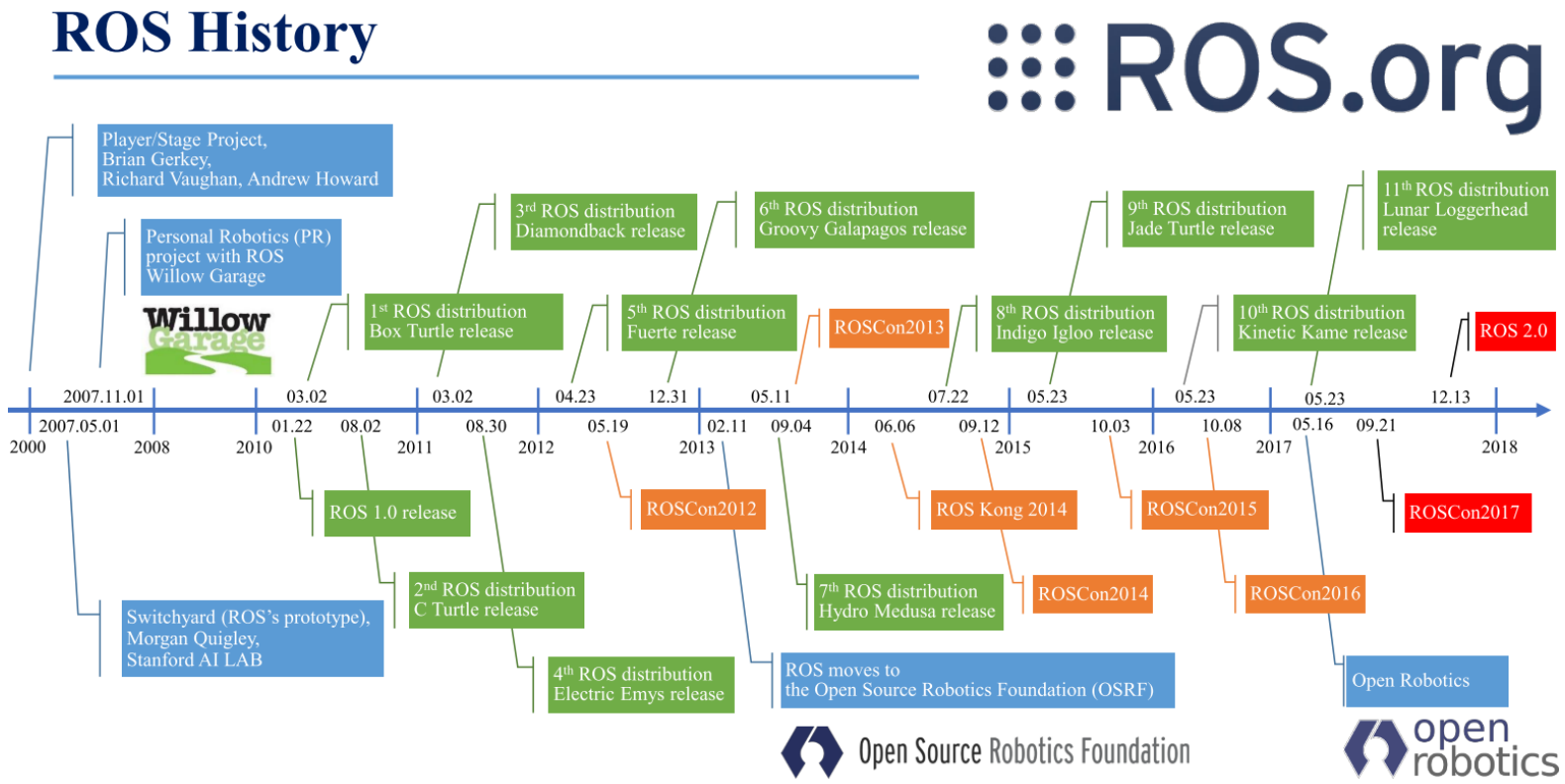
- ROS provides a framework to conveniently connect different robot modules (sensor data, algorithms/code, motor control commands, other robots, …)
- Widely used in robotics industry & academic labs
- Lots of tools/utilities and pre-built packages/libraries for robotics



https://maker.pro/ros/tutorial/robot-operating-system-2-ros-2-introduction-and-getting-started

ROS History

- Robot Operating System, known as ROS, was released in 2007
- In 2017, the new version ROS 2 was released
- Support for last release of ROS1 ends in 2025



ROS 1 vs. ROS 2

- ROS 1 is based on TCP/IP, while ROS 2 uses Data Distribution Service (DDS) - better for real-time communication, reliability, scalability
- ROS 1 mainly supports C++/Python, while ROS 2 is designed to be language-agnostic and in addition supports Java, Matlab, etc.
- ROS 1 primarily supports Linux, while ROS 2 in addition supports Windows, macOS
- ROS 2 has improved security, such as authentication/encryption for secure communication

ROS 1 vs. ROS 2

Features	ROS 1	ROS 2
Platforms	Linux, macOS	Linux, macOS, Windows
Real-time	external frameworks like OROCOS	real-time nodes when using a proper RTOS with carefully written user code
Security	SROS	SROS 2, DDS-Security, Robotic Systems Threat Model
Communication	XMLRPC + TCPROS	DDS (RTPS)
Middleware interface	-	rmw
Node manager (discovery)	ROS Master	No, use DDS's dynamic discovery
Languages	C++03, Python 2.7	C++14 (C++17), Python 3.5+
Client library	roscpp, rospy, rosjava, rosnodejs, and more	rclcpp, rclpy, rcljava, rcljs, and more
Build system	rosbuild → catkin (CMake)	ament (CMake), Python setuptools (Full support)
Build tools	catkin_make, catkin_tools	colcon
Build options	-	Multiple workspace, No non-isolated build, No devel space
Version control system	rosws → wstool, rosinstall (*.rosinstall)	vcstool (*.repos)
Life cycle	-	node life cycle
Multiple nodes	one node in a process	multiple nodes in a process
Threading model	single-threaded execution or multi-threaded execution	custom executors
Messages (topic, service, action)	*.msg, *.srv, *.action	*.msg, *.srv, *.action, *.idl
Command Line Interface	rosrun, roslaunch, ros topic ...	ros2 run, ros2 launch, ros2 topic ...
roslaunch	XML	Python, XML, YAML
Graph API	remapping at startup time only	remapping at runtime
Embedded Systems	rosserial, mROS	microROS, XEL Network, ros2arduino, Renesas DDS-XRCE(Micro-XRCE-DDS), AWS ARCLM

https://cafe.naver.com/openrt/23965

Some key features differences between ROS and ROS2

Features	ROS	ROS2
Platforms	Tested on Ubuntu Maintained on other Linux flavors as well as OS X	ROS2 is currently being CI tested and supported on Ubuntu Xenial, OS X El Capitan as well as Windows 10
C++	C++03 // don't useC++11 features in its API	Mainly uses C++11 Start and plan to use C++14 & C++17
Python	Target Python 2	>= Python 3.5
Middleware	Custom serialization format (transport protocol + central discovery mechanism)	Currently all implementations of this interface are based on the DDS standard.
Unify duration and time types	The duration and time types are defined in the client libraries, they are in C++ and Python	In ROS2 these types are defined as messages and therefore are consistent across languages.
Components with life cycle	In ROS every node usually has its own main function.	The life cycle can be used by tools like roslaunch to start a system composed of many components in a deterministic way.
Threading model	In ROS the developer can only choose between single-threaded execution or multi-threaded execution.	In ROS2 more granular execution models are available and custom executors can be implemented easily.
Multiple nodes	In ROS it is not possible to create more than one node in a process.	In ROS2 it is possible to create multiple nodes in a process.
roslaunch	In ROS roslaunch files are defined in XML with very limited capabilities.	In ROS2 launch files are written in Python which enables to use more complex logic like conditionals etc.

https://www.generationrobots.com/blog/en/ros-vs-ros2/

ROS 2 Distros

- List of distros: https://docs.ros.org/en/humble/Releases.html
- We use ROS 2 Humble
- ROS 2 distros are *not* tied exclusively to Ubuntu distros

Below is a list of current and historic ROS 2 distributions. Rows in the table marked in green are the currently supported distributions.

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021

ROS 2 - Installation

- Debian packages for ROS 2 Humble for Ubuntu 22.04 (recommended): https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html

- Build from source: https://docs.ros.org/en/humble/Installation/Alternatives/Ubuntu-Development-Setup.html

Activation

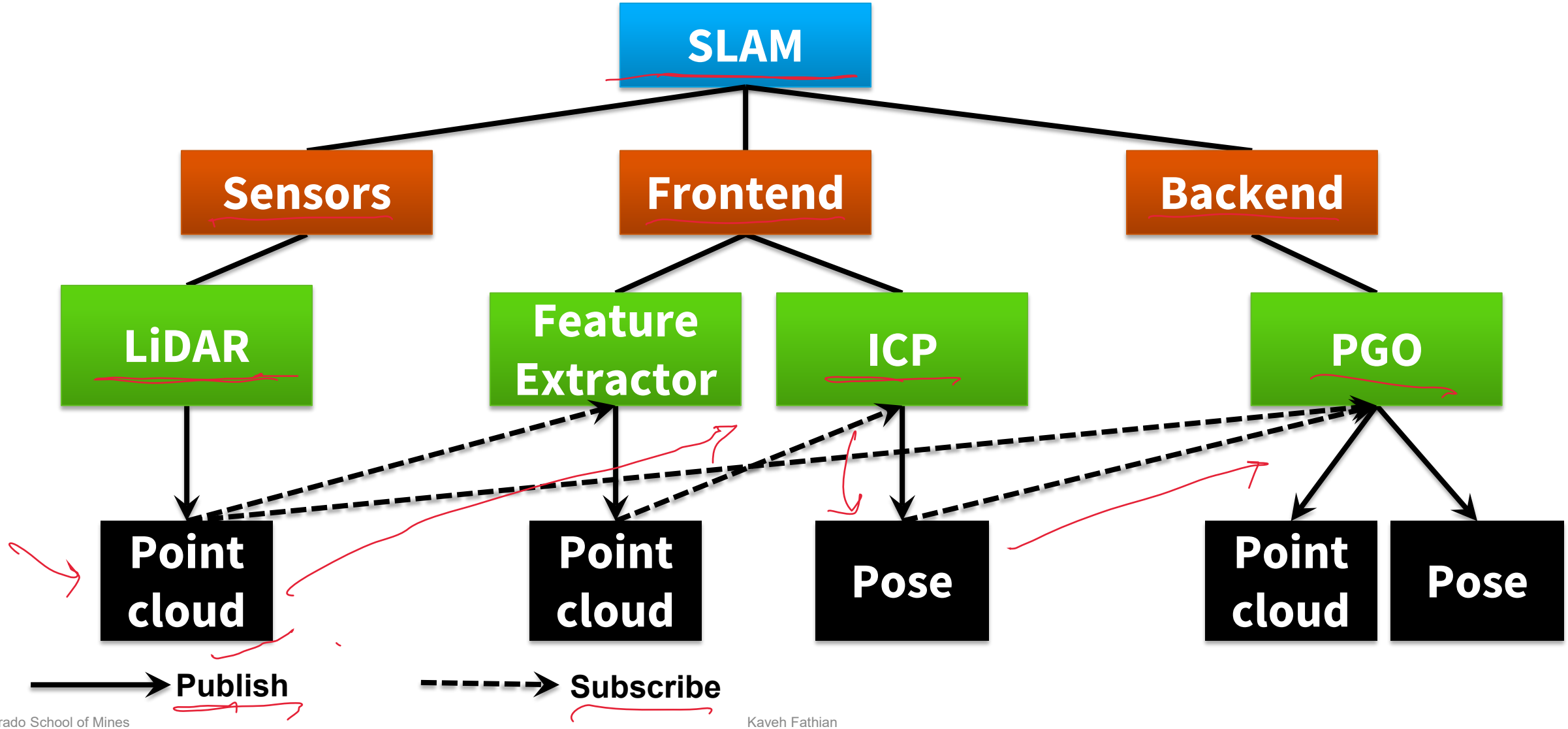
- Activate ROS2:

```
$ source /opt/ros/humble/setup.bash
```

```
$ echo “source /opt/ros/humble/setup.bash” >> /.bashrc
```

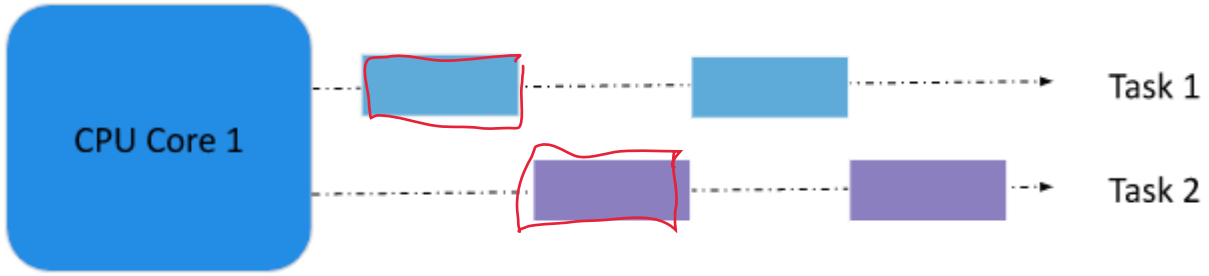
ROS Workspace Structure – SLAM Example

- Workspace → Packages → Nodes → Messages



Callbacks & Executers

- **Callback** is part of code that can be executed by an executor, or by the spinning functions (e.g., `spin`, `spin once`).
- You are in charge of defining what a callback does, but you cannot strictly enforce the moment in which it will be executed
- An **executor** controls the threading model used to process callbacks
- There are three types of executors: `SingleThreadedExecutor`, `StaticSingleThreadedExecutor`, and `MultiThreadedExecutor`
- The `spin` function will delegate to the executor the task of executing any callback associated with a node, whenever it thinks it's best to.



https://medium.com/@nullbyte.in/ros2-from-the-ground-up-part-5-concurrency-executors-and-callback-groups-c45900973fd2

Writing a ROS Node

*Slides by Francisco Martin Rico (https://github.com/fmrico/book_ros2)

Set up the user workspace

slam_ws

- Build the workspace

```
$ cd
$ mkdir -p bookros2_ws/src
$ cd bookros2_ws/src
```

- Activate the user workspace

```
$ source ~/bookros2_ws/install/setup.bash
```

```
$ echo "source ~/bookros2_ws/install/setup.bash" >>  /.bashrc
```

Package creation

- Build the workspace

```
$ cd

$ mkdir -p bookros2_ws/src
```

- Create the package template

```
$ cd ~/bookros2_ws/src

$ ros2 pkg create my_package --dependencies rclcpp std_msgs
```

Package my_package

```
my_package/
├── CMakeLists.txt
├── include
│   └── my_package
├── package.xml
└── src
    └── simple.cpp
```

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>TODO: Package description</description>
  <maintainer email="john.doe@evilrobot.com">johndoe</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>

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

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

First program

- The simplest node

```
#include "rclcpp/rclcpp.hpp"

int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("simple_node");

  rclcpp::spin(node);

  rclcpp::shutdown();

  return 0;
}
```

First program

- How to make a node

```cpp
#include "rclcpp/rclcpp.hpp"

int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("simple_node");

  rclcpp::spin(node);

  rclcpp::shutdown();

  return 0;
}
```

```cpp
1. std::shared_ptr<rclcpp::Node> node = std::shared_ptr<rclcpp::Node>(
   new rclcpp::Node("simple_node"));

2. std::shared_ptr<rclcpp::Node> node = std::make_shared<rclcpp::Node>(
   "simple_node");

3. rclcpp::Node::SharedPtr node = std::make_shared<rclcpp::Node>(
   "simple_node");

4. auto node = std::make_shared<rclcpp::Node>("simple_node");

5. auto node = rclcpp::Node::make_shared("simple_node");
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(basics)

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)

set(dependencies
  rclcpp
)

add_executable(simple src/simple.cpp)
ament_target_dependencies(simple ${dependencies})

install(TARGETS
  simple
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION lib/${PROJECT_NAME}
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

ament_export_dependencies(${dependencies})
ament_package()
```

Build and execute

- Build the package

```
cd ~/bookros2_ws

colcon build --symlink-install
```

- Activate the user workspace

```
$ source ~/bookros2_ws/install/setup.bash
```

```
$ echo "source ~/bookros2_ws/install/setup.bash" >>  /.bashrc
```

- Run the package

```
$ ros2 run my_package simple
```

```
$ ros2 node list

/simple_node
```

ROS in Terminal

*Slides by Francisco Martin Rico (https://github.com/fmrico/book_ros2)

ROS2Cli

```
$ ros2

usage: ros2 [-h] Call ‘ros2 <command> -h‘ for more detailed usage. ...
ros2 is an extensible command-line tool for ROS 2.

...
```

```
ros2 <command> <verb> [<params>|<option>]*
```

```
action      extension_points  node       test
bag         extensions        param      topic
component   interface         pkg        wtf
launch      run               daemon     lifecycle
security    doctor            multicast  service
```

Further readings:

- https://github.com/ros2/ros2cli
- https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/cli/cli_cheats_sheet.pdf

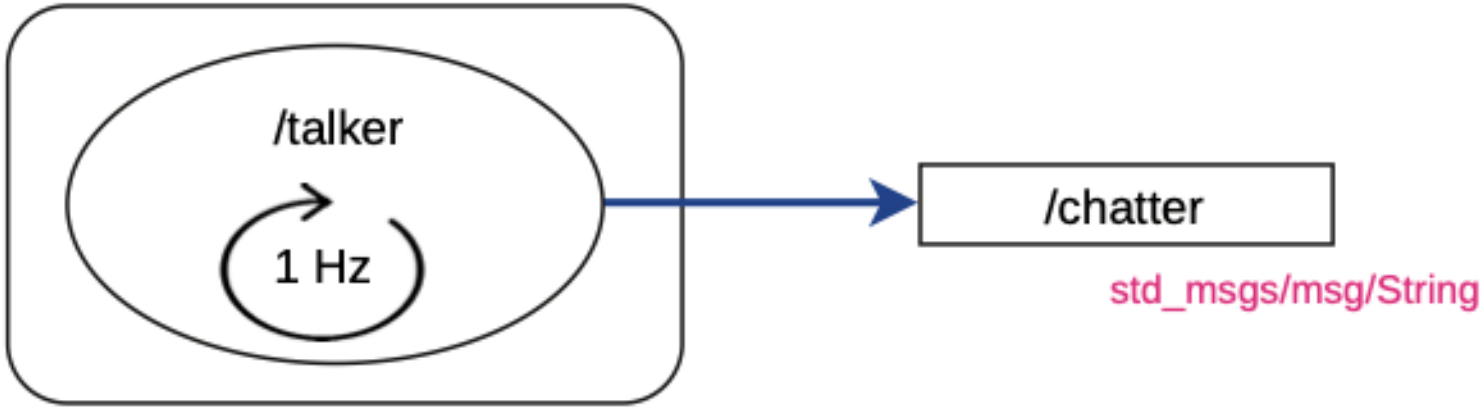
Packages

```
$ ros2 pkg list

ackermann_msgs
action_msgs
action_tutorials_cpp
...
```

```
$ ros2 pkg executables demo_nodes_cpp

demo_nodes_cpp add_two_ints_client
demo_nodes_cpp add_two_ints_client_async
demo_nodes_cpp add_two_ints_server
demo_nodes_cpp allocator_tutorial
...
demo_nodes_cpp talker
...
```

Running a ROS2 program

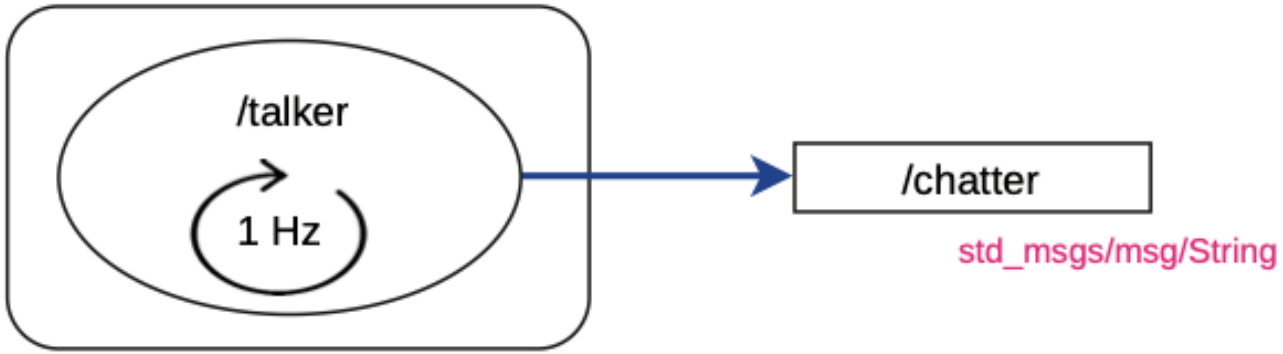
```
$ ros2 run demo_nodes_cpp talker

[INFO] [1643218362.316869744] [talker]: Publishing: 'Hello World: 1'
[INFO] [1643218363.316915225] [talker]: Publishing: 'Hello World: 2'
[INFO] [1643218364.316907053] [talker]: Publishing: 'Hello World: 3'
...
```



Running a ROS2 program

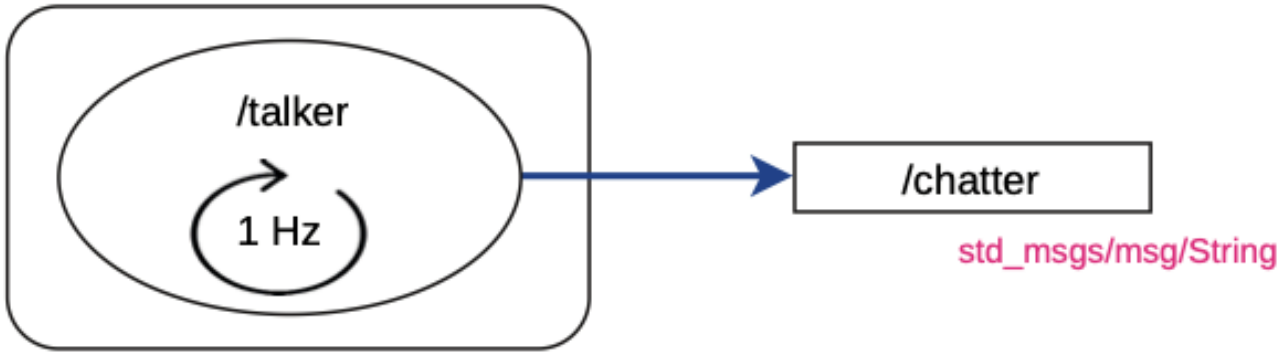
```
$ ros2 node list

/talker
```

```
$ ros2 topic list

/chatter
/parameter_events
/rosout
```



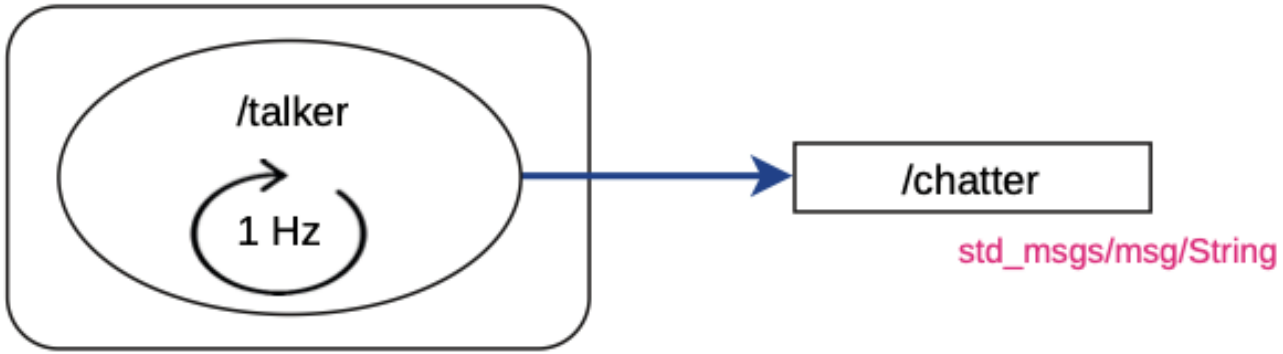
Running a ROS2 program

```
$ ros2 node info /talker

/talker
   Subscribers:
       /parameter_events: rcl_interfaces/msg/ParameterEvent
   Publishers:
       /chatter: std_msgs/msg/String
       /parameter_events: rcl_interfaces/msg/ParameterEvent
       /rosout: rcl_interfaces/msg/Log
   Service Servers:
...
```



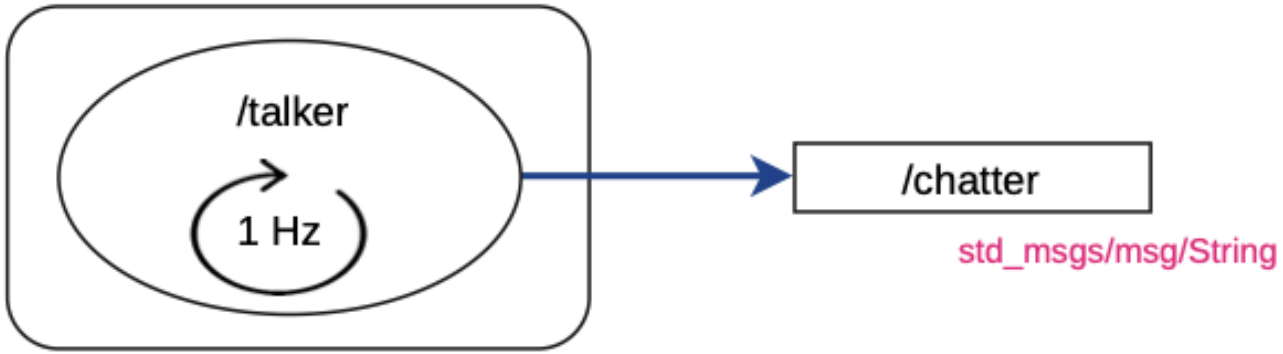
Running a ROS2 program

```
$ ros2 topic info /chatter

Type: std_msgs/msg/String
Publisher count: 1

Subscription count: 0
```



Running a ROS2 program

```
$ ros2 topic echo /chatter

data: 'Hello World: 1578'
---
data: 'Hello World: 1579'
...
```



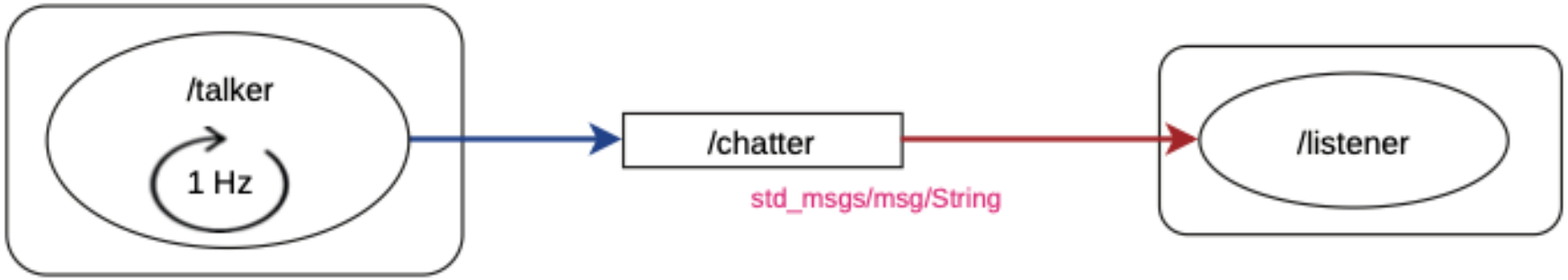
Interfaces

```
$ ros2 interface list

Messages:
    ackermann_msgs/msg/AckermannDrive
    ackermann_msgs/msg/AckermannDriveStamped
    ...
    visualization_msgs/msg/MenuEntry
Services:
    action_msgs/srv/CancelGoal
    ...
    visualization_msgs/srv/GetInteractiveMarkers
Actions:
    action_tutorials_interfaces/action/Fibonacci
    ...
```

```
$ ros2 interface show std_msgs/msg/String

... comments

string data
```

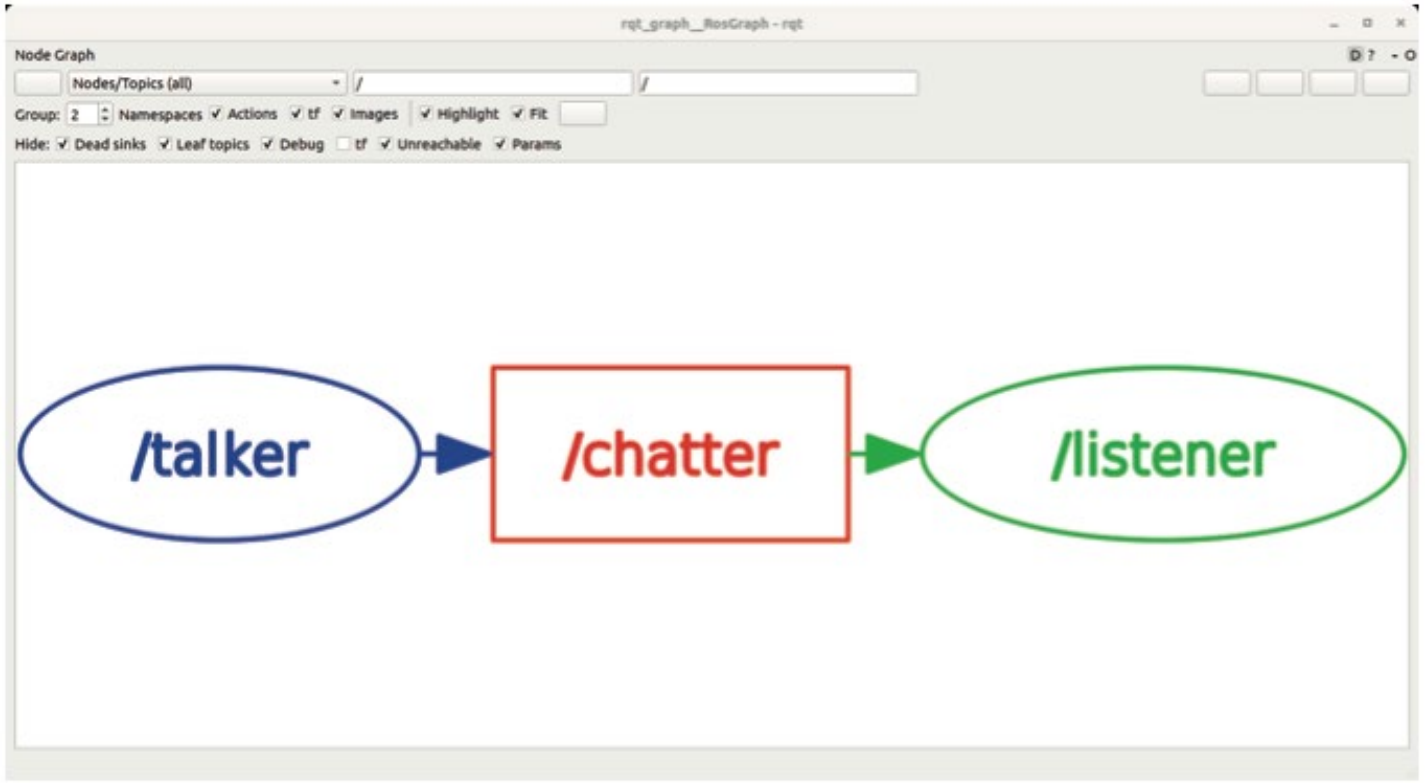
Running a listener

```
$ ros2 run demo_nodes_py listener

[INFO] [1643220136.232617223] [listener]: I heard: [Hello World: 1670]
[INFO] [1643220137.197551366] [listener]: I heard: [Hello World: 1671]
[INFO] [1643220138.198640098] [listener]: I heard: [Hello World: 1672]

...
```



RQT Tools

```
$ ros2 run rqt_graph rqt_graph
```



Package Example

*Slides by Francisco Martin Rico (https://github.com/fmrico/book_ros2)

Set up the user workspace

- Create and fill

```
$ cd
$ mkdir -p bookros2_ws/src
$ cd bookros2_ws/src
$ git clone https://github.com/fmrico/book_ros2.git
```

```
$ cd ~/bookros2_ws/src
$ vcs import . < book_ros2/third_parties.repos
```

Set up the user workspace

- Install dependencies

```
$ sudo rosdep init

$ rosdep update
```

```
$ cd ~/bookros2_ws

$ rosdep install --from-paths src --ignore-src -r -y
```

Set up the user workspace

- Build the workspace

```
$ cd ~/bookros2_ws

$ colcon build --symlink-install
```

- `Build`, `install`, and `log`
- Activate the user workspace

```
$ source ~/bookros2_ws/install/setup.bash
```

```
$ echo "source ~/bookros2_ws/install/setup.bash" >>  /.bashrc
```

Package content

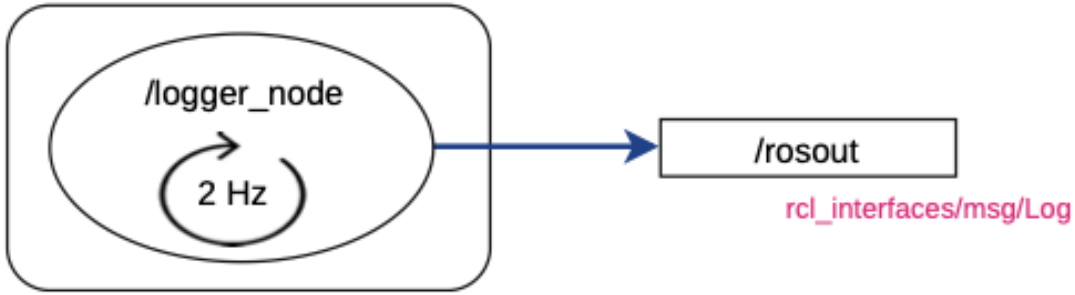
https://github.com/fmrico/book_ros2/tree/main/br2_basics

```
br2_basics
├── CMakeLists.txt
├── config
│   └── params.yaml
├── launch
│   ├── includer_launch.py
│   ├── param_node_v1_launch.py
│   ├── param_node_v2_launch.py
│   ├── pub_sub_v1_launch.py
│   └── pub_sub_v2_launch.py
├── package.xml
└── src
    ├── executors.cpp
    ├── logger_class.cpp
    ├── logger.cpp
    ├── param_reader.cpp
    ├── publisher_class.cpp
    ├── publisher.cpp
    └── subscriber_class.cpp
```

logger.cpp

- Use `RCLCPP_*` to show messages
- Control execution frequency with `rclcpp::Rate`
- `spin()` and `spin_some()`

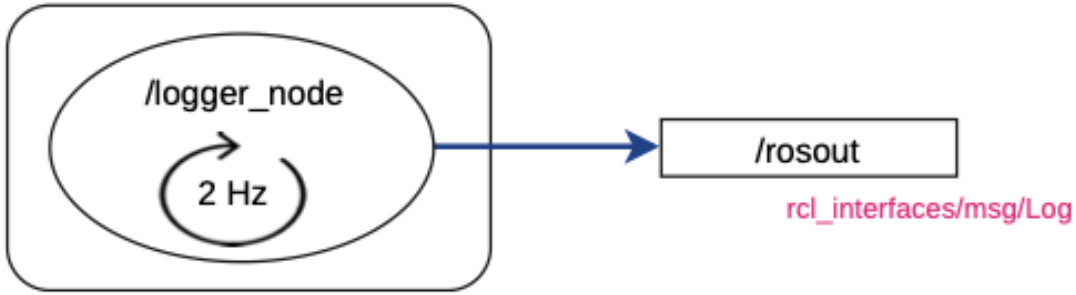
```
auto node = rclcpp::Node::make_shared("logger_node");

rclcpp::Rate loop_rate(500ms);
int counter = 0;

while (rclcpp::ok()) {
  RCLCPP_INFO(node->get_logger(), "Hello %d", counter++);

  rclcpp::spin_some(node);
  loop_rate.sleep();
}
```



logger.cpp

```
$ cd ~/bookros2_ws

$ colcon build --symlink-install --packages-select br2_basics
```

```
$ ros2 run br2_basics logger

[INFO] [1643264508.056814169] [logger_node]: Hello 0
[INFO] [1643264508.556910295] [logger_node]: Hello 1
...
```



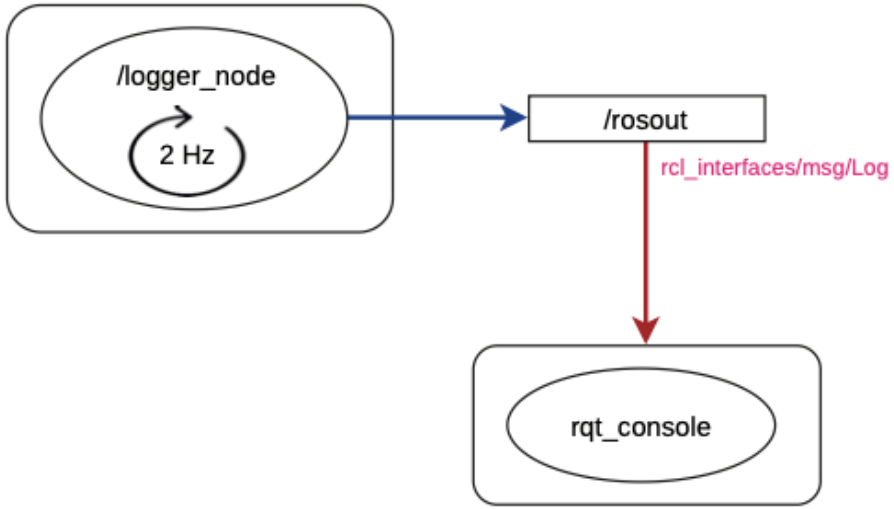
logger.cpp

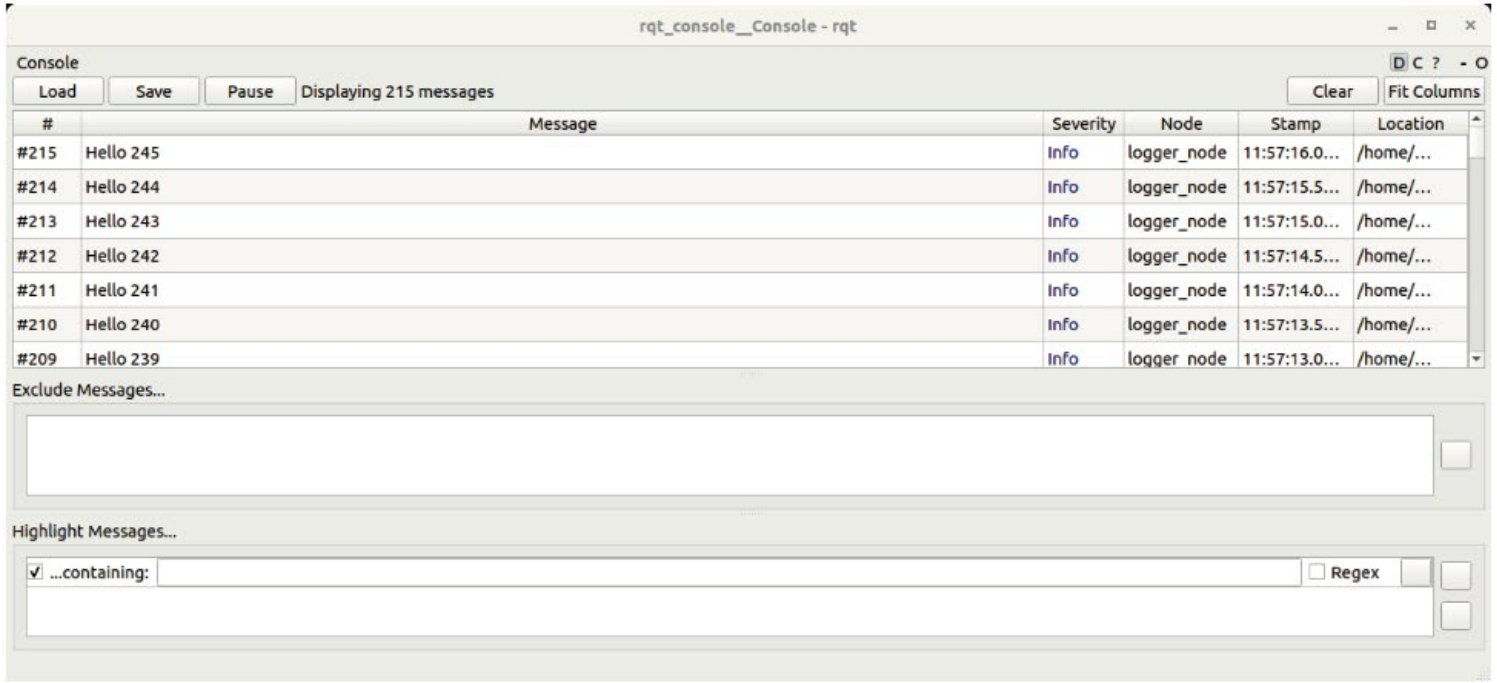
```
$ ros2 topic echo /rosout

stamp:
    sec: 1643264511
    nanosec: 556908791
level: 20
name: logger_node
msg: Hello 7
file: /home/fmrico/ros/ros2/bookros2_ws/src/book_ros2/br2_basics/src/logger.cpp
function: main
line: 27
---
stamp:
    sec: 1643264512
    nanosec: 57037520
level: 20
...
```

```
$ ros2 interface show rcl_interfaces/msg/Log
```

RQT Console

```
$ ros2 run rqt_console rqt_console
```






```
$ ros2 run br2_basics logger --ros-args --log-level debug
```

logger_class.cpp

- Inherit from `rclcpp::Node` helps to organize better your code
- Control execution cycle **internally** with timers

```cpp
class LoggerNode : public rclcpp::Node
{
public:
  LoggerNode() : Node("logger_node")
  {
    counter_ = 0;
    timer_ = create_wall_timer(
      500ms, std::bind(&LoggerNode::timer_callback, this));
  }

  void timer_callback()
  {
    RCLCPP_INFO(get_logger(), "Hello %d", counter_++);
  }

private:
  rclcpp::TimerBase::SharedPtr timer_;
  int counter_;
};

int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node = std::make_shared<LoggerNode>();

  rclcpp::spin(node);

  rclcpp::shutdown();
  return 0;
}
```

logger_class.cpp

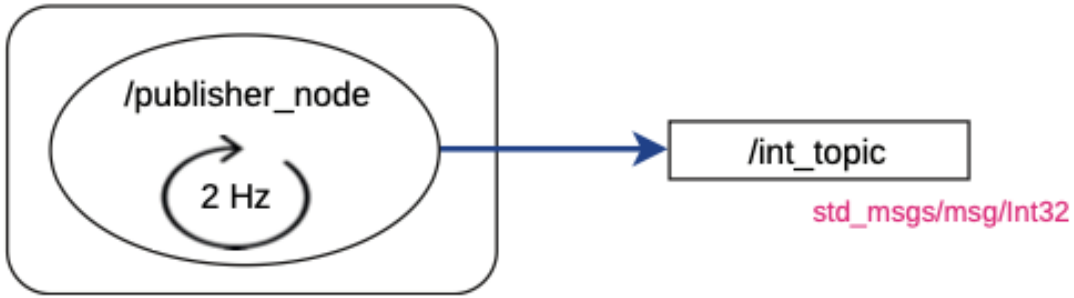
```
add_executable(logger_class src/logger.cpp)
ament_target_dependencies(logger ${dependencies})

add_executable(logger_class src/logger_class.cpp)
ament_target_dependencies(logger_class ${dependencies})

install(TARGETS
  logger
  logger_class
  ...
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION lib/${PROJECT_NAME}
)
```

```
$ ros2 run br2_basics logger_class
```

Publishing

```cpp
class PublisherNode : public rclcpp::Node
{
public:
  PublisherNode() : Node("publisher_node")
  {
    publisher_ = create_publisher<std_msgs::msg::Int32>("int_topic", 10);
    timer_ = create_wall_timer(
      500ms, std::bind(&PublisherNode::timer_callback, this));
  }

  void timer_callback()
  {
    message_.data += 1;
    publisher_->publish(message_);
  }

private:
  rclcpp::Publisher<std_msgs::msg::Int32>::SharedPtr publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
  std_msgs::msg::Int32 message_;
};
```

```cpp
// For std_msgs/msg/Int32
#include "std_msgs/msg/int32.hpp"

std_msgs::msg::Int32 msg_int32;


// For sensor_msgs/msg/LaserScan
#include "sensor_msgs/msg/laser_scan.hpp"

sensor_msgs::msg::LaserScan msg_laserscan;
```

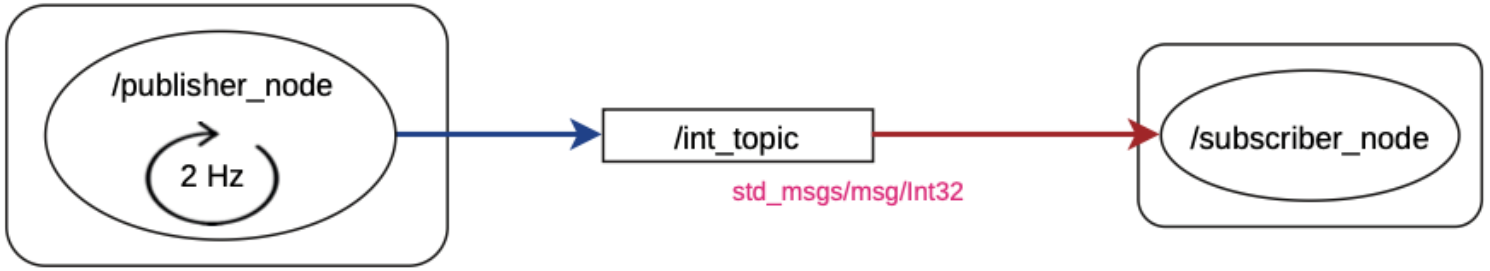



```
$ ros2 run br2_basics publisher_class
```

Subscribing

```cpp
class SubscriberNode : public rclcpp::Node
{
public:
  SubscriberNode() : Node("subscriber_node")
  {
    subscriber_ = create_subscription<std_msgs::msg::Int32>("int_topic", 10,
      std::bind(&SubscriberNode::callback, this, _1));
  }

  void callback(const std_msgs::msg::Int32::SharedPtr msg)
  {
    RCLCPP_INFO(get_logger(), "Hello %d", msg->data);
  }

private:
  rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr subscriber_;
};
```

```
$ ros2 run br2_basics subscriber_class
```



About QoS

Default	Reliable	Volatile	Keep Last
Services	Reliable	Volatile	Normal Queue
Sensor	Best Effort	Volatile	Small Queue
DParameters	Reliable	Volatile	Large Queue

```
publisher = node->create_publisher<std_msgs::msg::String>(
  "chatter", rclcpp::QoS(100).transient_local().best_effort());
```

```
publisher_ = create_publisher<sensor_msgs::msg::LaserScan>(
  "scan", rclcpp::SensorDataQoS().reliable());
```

Compatibility of QoS **durability** profiles		**Subscriber**	
		Volatile	**Transient Local**
Publisher	**Volatile**	Volatile	**No Connection**
	Transient Local	Volatile	Transient Local

Compatibility of QoS **reliability** profiles		**Subscriber**	
		Best Effort	**Reliable**
Publisher	**Best Effort**	Best Effort	**No Connection**
	Reliable	Best Effort	Reliable

Read more at: https://medium.com/@nullbyte.in/ros2-from-the-ground-up-part-7-achieving-reliable-communication-in-ros-2-with-qos-configurations-83c534c3aff5

Launchers

- Declaratives
- Alternatives: xml and yaml

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
  pub_cmd = Node(
    package='basics',
    executable='publisher',
    output='screen'
  )

  sub_cmd = Node(
    package='basics',
    executable='subscriber_class',
    output='screen'
  )

  ld = LaunchDescription()
  ld.add_action(pub_cmd)
  ld.add_action(sub_cmd)

  return ld
```

```
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
```

```
$ ros2 launch br2_basics pub_sub_v2_launch.py
```

Parameters

- Use parameters for configure node's behavior
- Declare parameters and get their values

```
class LocalizationNode : public rclcpp::Node
{
public:
  LocalizationNode() : Node("localization_node")
  {
    declare_parameter<int>("number_particles", 200);
    declare_parameter<std::vector<std::string>>("topics", {});
    declare_parameter<std::vector<std::string>>("topic_types", {});

    get_parameter("number_particles", num_particles_);
    RCLCPP_INFO_STREAM(get_logger(), "Number of particles: " << num_particles_);

    get_parameter("topics", topics_);
    get_parameter("topic_types", topic_types_);

    if (topics_.size() != topic_types_.size()) {
      RCLCPP_ERROR(get_logger(), "Number of topics (%zu) != number of types (%zu)",
        topics_.size(), topic_types_.size());
    } else {
      RCLCPP_INFO_STREAM(get_logger(), "Number of topics: " << topics_.size());
      for (size_t i = 0; i < topics_.size(); i++) {
        RCLCPP_INFO_STREAM(
          get_logger(),
          "\t" << topics_[i] << "\t - " << topic_types_[i]);
      }
    }
}

private:
  int num_particles_;
  std::vector<std::string> topics_;
  std::vector<std::string> topic_types_;
};
```

Parameters

- Use parameters for configure node's behavior
- Declare parameters and get their values

```
$ ros2 run br2_basics param_reader
```

```
$ ros2 run br2_basics param_reader --ros-args -p number_particles:=300
```

```
$ ros2 run br2_basics param_reader --ros-args -p number_particles:=300
-p topics:= '[scan, image]' -p topic_types:='[sensor_msgs/msg/LaserScan,
sensor_msgs/msg/Image]'
```

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    param_reader_cmd = Node(
        package='basics',
        executable='param_reader',
        parameters=[{
            'particles': 300,
            'topics': ['scan', 'image'],
            'topic_types': ['sensor_msgs/msg/LaserScan', 'sensor_msgs/msg/Image']
        }],
        output='screen'
    )

    ld = LaunchDescription()
    ld.add_action(param_reader_cmd)

    return ld
```

Parameters

- Use parameters for configure node's behavior
- Declare parameters and get their values

config/params.yaml

```
localization_node:
  ros__parameters:
    number_particles: 300
    topics: [scan, image]
    topic_types: [sensor_msgs/msg/LaserScan, sensor_msgs/msg/Image]
```

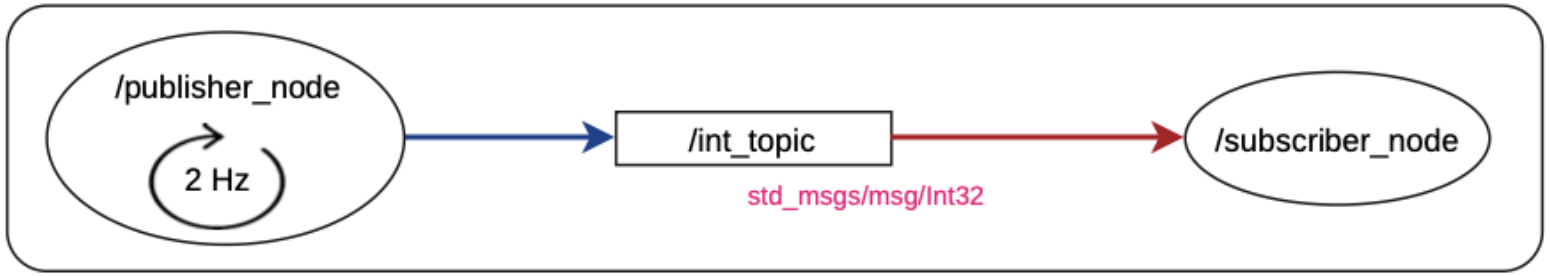
```
$ ros2 run br2_basics param_reader --ros-args --params-file
install/basics/share/basics/config/params.yaml
```

```
def generate_launch_description():
    ...
    param_reader_cmd = Node(
        package='basics',
        executable='param_reader',
        parameters=[param_file],
        output='screen'
    )
```

Executors

```
int main(int argc, char * argv[]) {
  rclcpp::init(argc, argv);

  auto node_pub = std::make_shared<PublisherNode>();
  auto node_sub = std::make_shared<SubscriberNode>();

  rclcpp::executors::SingleThreadedExecutor executor;

  executor.add_node(node_pub);
  executor.add_node(node_sub);

  executor.spin();

  rclcpp::shutdown();
  return 0;
}
```

```
auto node_pub = std::make_shared<PublisherNode>();
auto node_sub = std::make_shared<SubscriberNode>();

rclcpp::executors::MultiThreadedExecutor executor(
      rclcpp::executor::ExecutorArgs(), 8);

executor.add_node(node_pub);
executor.add_node(node_sub);

executor.spin();
}
```



ROS 2 Cheats Sheet

Command Line Interface

All ROS 2 CLI tools start with the prefix 'ros2' followed by a command, a verb and (possibly) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ ros2 command --help
```

and similarly for verb documentation,

```
$ ros2 command verb -h
```

Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,

```
$ ros2 command [tab][tab]
```

Some of the examples below rely on:
ROS 2 demos package.

---

action Allows to manually send a goal and displays debugging information about actions.

Verbs:

info	Output information about an action.
list	Output a list of action names.
send_goal	Send an action goal.
show	Output the action definition.

Examples:

```
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \
 action_tutorials/action/Fibonacci "order: 5"
$ ros2 action show action_tutorials/action/Fibonacci
```

---

bag Allows to record/play topics to/from a rosbag.

Verbs:

info	Output information of a bag.
play	Play a bag.
record	Record a bag.

Examples:

```
$ ros2 info <bag-name>
$ ros2 play <bag-name>
$ ros2 record -a
```

---

component Various component related verbs.

Verbs:

list	Output a list of running containers and components.
load	Load a component into a container node.
standalone	Run a component into its own standalone container node.
types	Output a list of components registered in the ament index.
unload	Unload a component from a container node.

Examples:

```
$ ros2 component list
$ ros2 component load /ComponentManager \
 composition composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager 1
```

---

daemon Various daemon related verbs.

Verbs:

start	Start the daemon if it isn't running.
status	Output the status of the daemon.
stop	Stop the daemon if it is running

---

doctor A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.

Alias: **wtf** (where's the fire).

Arguments:

--report/-r	Output report of all checks.
--report-fail/-rf	Output report of failed checks only.
--include-warning/-iw	Include warnings as failed checks.

Examples:

```
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
$ ros2 doctor --include-warning --report-fail
```

or similarly,

```
$ ros2 wtf
```

---

extension_points List extension points.

---

extensions List extensions.

---

interface Various ROS interfaces (actions/topics/services)-related verbs. Interface type can be filtered with either of the following option, '--only-actions', '--only-msgs', '--only-srvs'.

Verbs:

list	List all interface types available.
package	Output a list of available interface types within one package.
packages	Output a list of packages that provide interfaces.
proto	Print the prototype (body) of an interfaces.
show	Output the interface definition.

Examples:

```
$ ros2 interface list
$ ros2 interface package std_msgs
$ ros2 interface packages --only-msgs
$ ros2 interface proto example_interfaces/srv/AddTwoInts
$ ros2 interface show geometry_msgs/msg/Pose
```

---

launch Allows to run a launch file in an arbitrary package without to 'cd' there first.

Usage:

```
$ ros2 launch <package> <launch-file>
```

Example:

```
$ ros2 launch demo_nodes_cpp add_two_ints.launch.py
```

---

lifecycle Various lifecycle related verbs.

Verbs:

get	Get lifecycle state for one or more nodes.
list	Output a list of available transitions.
nodes	Output a list of nodes with lifecycle.
set	Trigger lifecycle state transition.

---

msg (deprecated) Displays debugging information about messages.

Verbs:

list	Output a list of message types.
package	Output a list of message types within a given package.
packages	Output a list of packages which contain messages.
show	Output the message definition.

Examples:

```
$ ros2 msg list
$ ros2 msg package std_msgs
$ ros2 msg packages
$ ros2 msg show geometry_msgs/msg/Pose
```

---

multicast Various multicast related verbs.

Verbs:

receive	Receive a single UDP multicast packet.
send	Send a single UDP multicast packet.

---

node Displays debugging information about nodes.

Verbs:

info	Output information about a node.
list	Output a list of available nodes.

Examples:

```
$ ros2 node info /talker
$ ros2 node list
```

---

param Allows to manipulate parameters.

Verbs:

delete	Delete parameter.
describe	Show descriptive information about declared parameters.
dump	Dump the parameters of a given node in yaml format, either in terminal or in a file.
get	Get parameter.
list	Output a list of available parameters.
set	Set parameter

Examples:

```
$ ros2 param delete /talker /use_sim_time
$ ros2 param get /talker /use_sim_time
$ ros2 param list
$ ros2 param set /talker /use_sim_time false
```

---

pkg Create a ros2 package or output package(s)-related information.

Verbs:

create	Create a new ROS2 package.
executables	Output a list of package specific executables.
list	Output a list of available packages.
prefix	Output the prefix path of a package.
xml	Output the information contained in the package xml manifest.

Examples:

```
$ ros2 pkg executables demo_nodes_cpp
$ ros2 pkg list
$ ros2 pkg prefix std_msgs
$ ros2 pkg xml -t version
```

---

run Allows to run an executable in an arbitrary package without having to 'cd' there first.

Usage:

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

Example:

```
$ ros2 run demo_node_cpp talker
```

---

security Various security related verbs.

Verbs:

create_key	Create key.
create_permission	Create keystore.
generate_artifacts	Create permission.
list_keys	Distribute key.
create_keystore	Generate keys and permission files from a list of identities and policy files.
distribute_key	Generate XML policy file from ROS graph data.
generate_policy	List keys.

Examples (see sros2 package):

```
$ ros2 security create_key demo_keys /talker
$ ros2 security create_permission demo_keys /talker  \
 policies/sample_policy.xml
$ ros2 security generate_artifacts
$ ros2 security create_keystore demo_keys
```

---

service Allows to manually call a service and displays debugging information about services.

Verbs:

call	Call a service.
find	Output a list of services of a given type.
list	Output a list of service names.
type	Output service's type.

Examples:

```
$ ros2 service call /add_two_ints  \
 example_interfaces/AddTwoInts "a: 1, b: 2"
$ ros2 service find rcl_interfaces/srv/ListParameters
$ ros2 service list
$ ros2 service type /talker/describe_parameters
```

---

srv (deprecated) Various srv related verbs.

Verbs:

list	Output a list of available service types.
package	Output a list of available service types within one package.
packages	Output a list of packages which contain services.
show	Output the service definition.

---

test Run a ROS2 launch test.

---

topic A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Verbs:

bw	Display bandwidth used by topic.
delay	Display delay of topic from timestamp in header.
echo	Output messages of a given topic to screen.
find	Find topics of a given type type.
hz	Display publishing rate of topic.
info	Output information about a given topic.
list	Output list of active topics.
pub	Publish data to a topic.
type	Output topic's type.

Examples:

```
$ ros2 topic bw /chatter
$ ros2 topic echo /chatter
$ ros2 topic find rcl_interfaces/msg/Log
$ ros2 topic hz /chatter
$ ros2 topic info /chatter
$ ros2 topic list
$ ros2 topic pub /chatter std_msgs/msg/String  \
 'data: Hello ROS 2 world'
$ ros2 topic type /rosout
```

ROS 2 Cheats Sheet

colcon - collective construction

colcon is a command line tool to improve the workflow of building, testing and using multiple software packages. It automates the process, handles the ordering and sets up the environment to use the packages.

All colcon tools start with the prefix 'colcon' followed by a command and (likely) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ colcon command --help
```

Moreover, colcon offers auto-completion for all verbs and most positional/optional arguments. E.g.,

```
$ colcon command [tab][tab]
```

Find out how to enable auto-completion at colcon's online documentation.

Environment variables:

- CMAKE_COMMAND The full path to the CMake executable.
- COLCON_ALL_SHELLS Flag to enable all shell extensions.
- COLCON_COMPLETION_LOGFILE Set the logfile for completion time.
- COLCON_DEFAULTS_FILE Set path to the yaml file containing the default values for the command line arguments (default:$COLCON_HOME/defaults.yaml).
- COLCON_DEFAULT_EXECUTOR Select the default executor extension.
- COLCON_EXTENSION_BLACKLIST Blacklist extensions which should not be used.
- COLCON_HOME Set the configuration directory (default: /.colcon.)
- COLCON_LOG_LEVEL Set the log level (debug—10, info—20, warn—30, error—40, critical—50, or any other positive numeric value).
- COLCON_LOG_PATH Set the log directory (default: $COLCON_HOME/log).
- CTEST_COMMAND The full path to the CTest executable.
- POWERSHELL_COMMAND The full path to the PowerShell executable.

Global options:

- --log-base <path> The base path for all log directories (default: log).
- --log-level <level> Set log level for the console output, either by numeric or string value (default: warn)

---

build Build a set of packages.

Examples:

Build the whole workspace:

```
$ colcon build
```

Build a single package excluding dependencies:

```
$ colcon build --packages-selected demo_nodes_cpp
```

Build two packages including dependencies, use symlinks instead of copying files where possible and print immediately on terminal:

```
$ colcon build --packages-up-to demo_nodes_cpp \
 action_tutorials --symlink-install \
 --event-handlers console_direct+
```

---

extension-points List extension points.

---

extensions Package information.

---

info List extension points.

---

list List packages, optionally in topological ordering.

Example:

List all packages in the workspace:

```
$ colcon list
```

List all packages names in topological order up-to a given package:

```
$ colcon list --names-only --topological-order \
 --packages-up-to demo_nodes_cpp
```

---

metadata Manage metadata of packages.

---

test Test a set of packages.

Example:

Test the whole workspace:

```
$ colcon test
```

Test a single package excluding dependencies:

```
$ colcon test --packages-select demo_nodes_cpp
```

Test a package including packages that depend on it:

```
$ colcon test --packages-above demo_nodes_py
```

Test two packages including dependencies, and print on terminal:

```
$ colcon test --packages-up-to demo_nodes_cpp \
 demo_nodes_py --event-handlers console_direct+
```

Pass arguments to pytest (e.g. to print a coverage report):

```
$ colcon test --packages-select demo_nodes_cpp \
 --event-handlers console_direct+ \
 --pytest-args --cov=sros2
```

---

test-result Show the test results generated when testing a set of packages.

Example:

Show all test results generated, including successful tests:

```
$ colcon test-result --all
```

---

version-check Compare local package versions with PyPI.

Examples:

```
$ todo
```

Must know colcon flags.

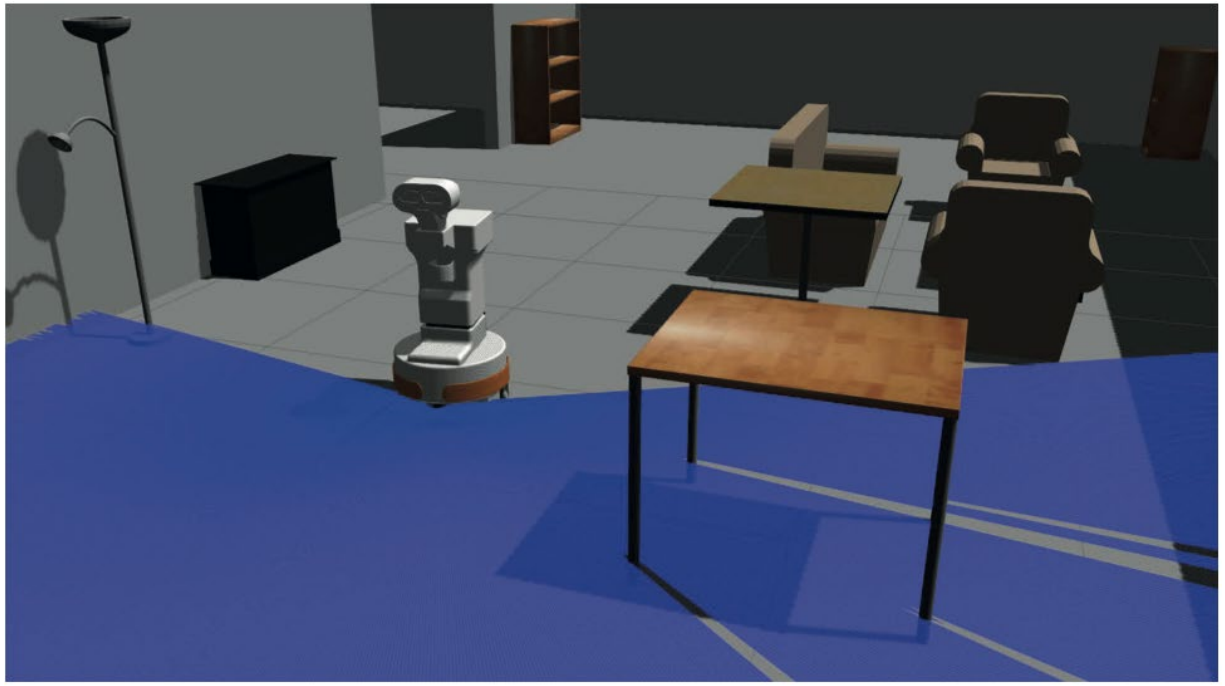
- --symlink-install Use 'symlinks' instead of installing (copying) files where possible.
- --continue-on-error Continue other packages when a package fails to build. Packages recursively depending on the failed package are skipped.
- --event-handlers console_direct+ Show output on console.
- --event-handlers console_cohesion+ Show output on console after a package has finished.
- --packages-select Build only specific package(s).
- --packages-up-to Build specific package(s) and its/their recursive dependencies.
- --packages-above Build specific package(s) and other packages that recursively depending on it.
- --packages-skip Skip package(s).
- --packages-skip-build-finished Skip a set of packages which have finished to build previously.
- --cmake-args Pass arguments to CMake projects.
- --cmake-clean-cache Remove CMake cache before the build (implicitly forcing CMake configure step).
- --cmake-clean-first Build target 'clean' first, then build (to only clean use '–cmake-target clean').
- --cmake-force-configure Force CMake configure step.

Simulation

*Slides by Francisco Martin Rico (https://github.com/fmrico/book_ros2)

Simulated Robot Setup

https://github.com/fmrico/book_ros2/tree/main/br2_tiago



```
$ ros2 launch br2_tiago sim.launch.py world:=factory
$ ros2 launch br2_tiago sim.launch.py world:=featured
$ ros2 launch br2_tiago sim.launch.py world:=pal_office
$ ros2 launch br2_tiago sim.launch.py world:=small_factory
$ ros2 launch br2_tiago sim.launch.py world:=small_office
$ ros2 launch br2_tiago sim.launch.py world:=willow_garage
```

Topics and Remaps

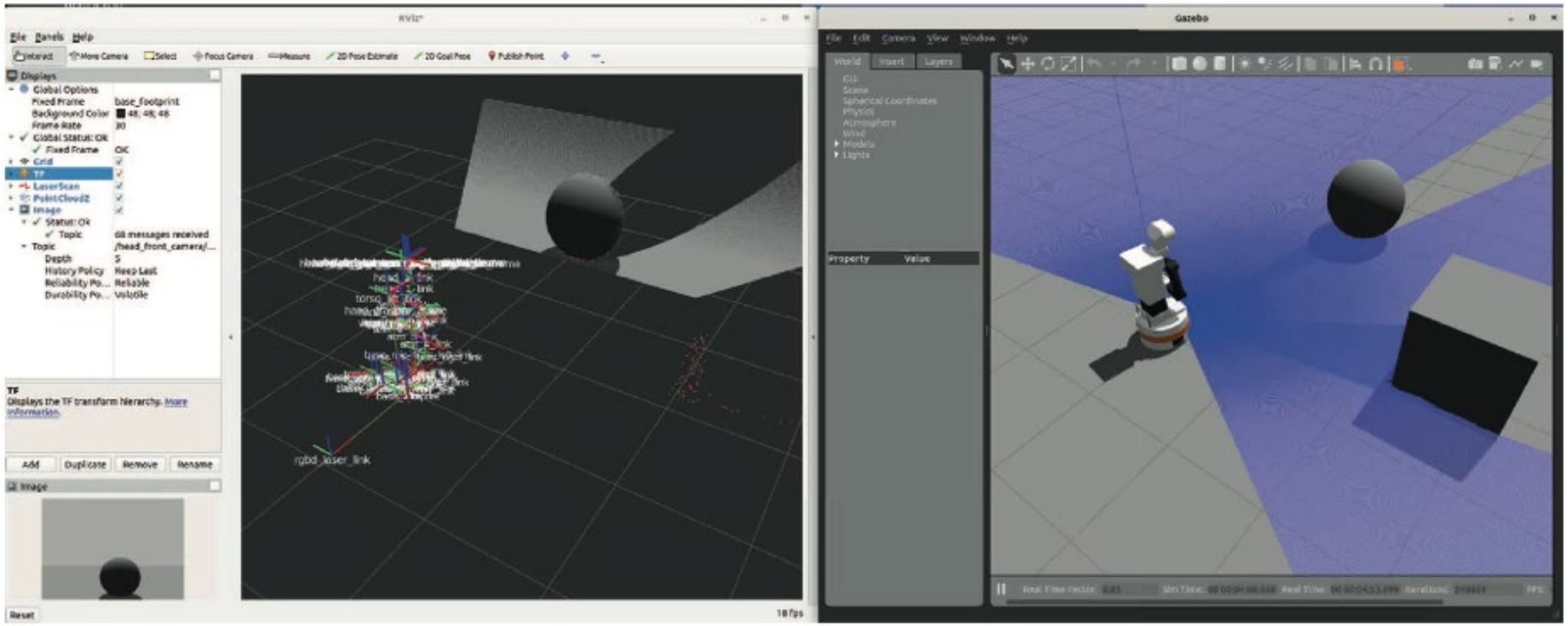
```
$ ros2 topic list
```

```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r
cmd_vel:=key_vel
```

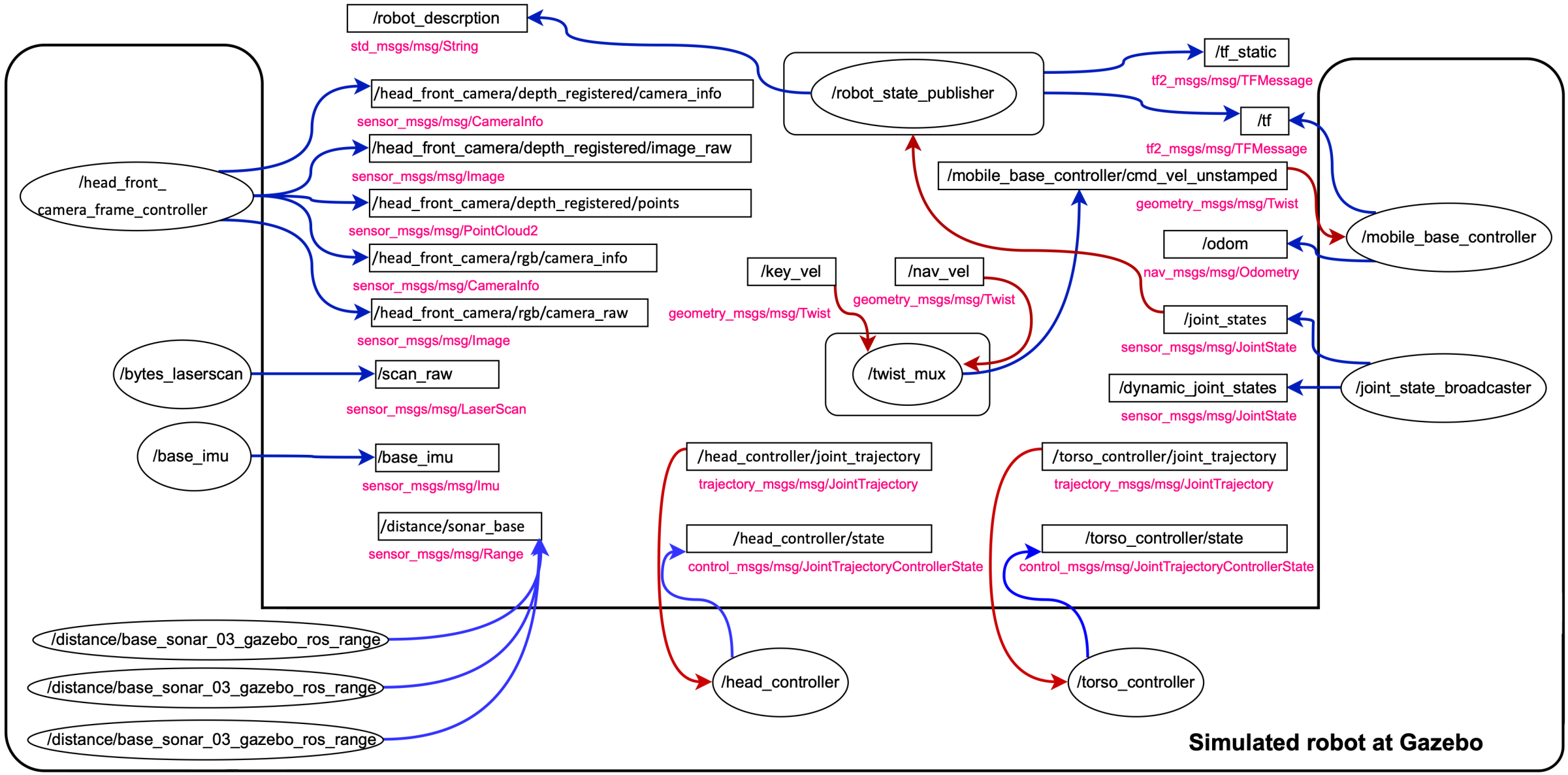

```
$ ros2 topic echo --no-arr /scan_raw

$ ros2 topic echo --no-arr /head_front_camera/rgb/image_raw
```

Rviz2

```
$ ros2 run rviz2 rviz2
```



Computation Graph



Transforms

*Slides by Francisco Martin Rico (https://github.com/fmrico/book_ros2)

The TF Subsystem

- One of the greatest treasures in ROS
- It allows to use and transform coordinates between different reference axes (**frames**)
- The robot perceives through sensors placed somewhere in the robot, even in moving parts, and performs actions specifying spatial positions

$$P_B = RT_{A \to B} * P_A$$

$$\begin{pmatrix} x_B \\ y_B \\ z_B \\ 1 \end{pmatrix} = \begin{pmatrix} R^{xx}_{A \to B} & R^{xy}_{A \to B} & R^{xz}_{A \to B} & T^{x}_{A \to B} \\ R^{yx}_{A \to B} & R^{yy}_{A \to B} & R^{yz}_{A \to B} & T^{y}_{A \to B} \\ R^{zx}_{A \to B} & R^{zy}_{A \to B} & R^{zz}_{A \to B} & T^{z}_{A \to B} \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}$$

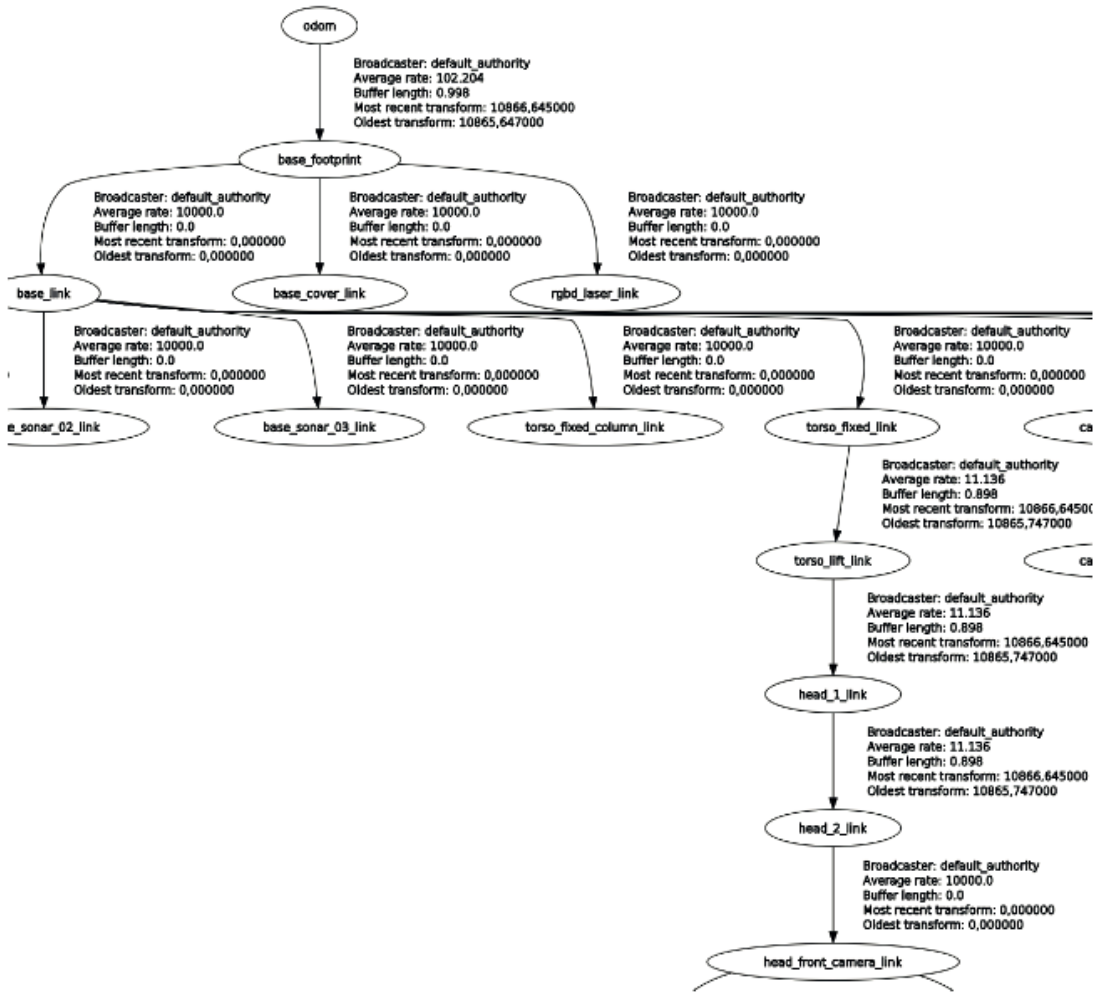
The TF Subsystem

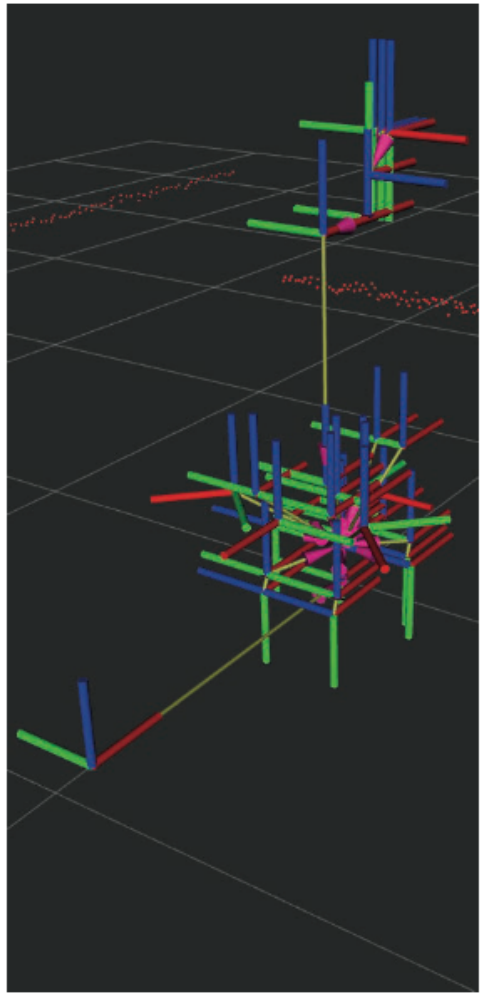
- Topics `/tf` and `/tf_static` (`tf2_msgs/msg/TFMessage`)

```
$ ros2 interface show tf2_msgs/msg/TFMessage

geometry_msgs/TransformStamped[] transforms
   std_msgs/Header header
   string child_frame_id
   Transform transform
      Vector3 translation
         float64 x
         float64 y
         float64 z
      Quaternion rotation
         float64 x 0
         float64 y 0
         float64 z 0
         float64 w 1
```

The TF Subsystem

```
$ ros2 run rqt_tf_tree rqt_tf_tree
```

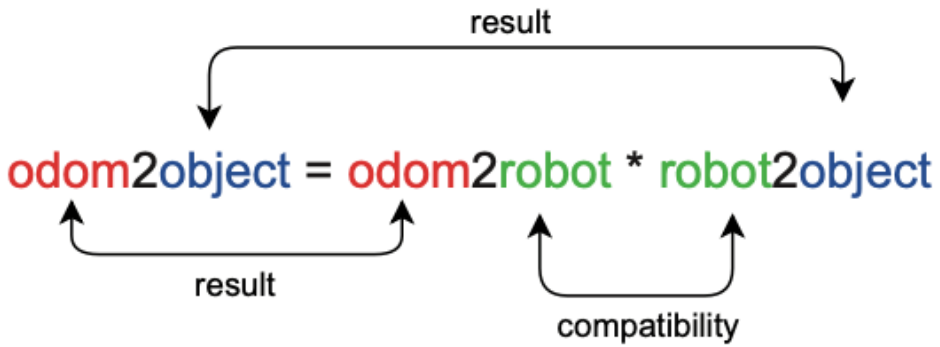




TF Listeners and Publishers

```
geometry_msgs::msg::TransformStamped detection_tf;

detection_tf.header.frame_id = "base_footprint";
detection_tf.header.stamp = now();
detection_tf.child_frame_id = "detected_obstacle";
detection_tf.transform.translation.x = 1.0;

tf_broadcaster_->sendTransform(detection_tf);
```

```
tf2_ros::Buffer tfBuffer;
tf2_ros::TransformListener tfListener(tfBuffer);

...

geometry_msgs::msg::TransformStamped odom2obstacle;
odom2obstacle = tfBuffer_.lookupTransform("odom", "detected_obstacle", tf2::TimePointZero);
```



ROS Bag

Recording and playing back data

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.

Goal: Record data published on a topic so you can replay and examine it any time.

Tutorial level: Beginner

Time: 10 minutes

Contents

- Background
- Prerequisites
- Tasks
  - 1 Setup
  - 2 Choose a topic
  - 3 ros2 bag record
  - 4 ros2 bag info
  - 5 ros2 bag play
- Summary
- Next steps
- Related content

Background

`ros2 bag` is a command line tool for recording data published on topics in your system. It accumulates the data passed on any number of topics and saves it in a database. You can then replay the data to reproduce the results of your tests and experiments. Recording topics is also a great way to share your work and allow others to recreate it.

Prerequisites

You should have `ros2 bag` installed as a part of your regular ROS 2 setup.

If you need to install ROS 2, see the Installation instructions.

This tutorial talks about concepts covered in previous tutorials, like nodes and topics. It also uses the turtlesim package.

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

Tasks

1 Setup

You'll be recording your keyboard input in the `turtlesim` system to save and replay later on, so begin by starting up the `/turtlesim` and `/teleop_turtle` nodes.

Open a new terminal and run:

```
ros2 run turtlesim turtlesim_node
```

Open another terminal and run:

```
ros2 run turtlesim turtle_teleop_key
```

Let's also make a new directory to store our saved recordings, just as good practice:

```
mkdir bag_files
cd bag_files
```

2 Choose a topic

`ros2 bag` can only record data from published messages in topics. To see the list of your system's topics, open a new terminal and run the command:

```
ros2 topic list
```

Which will return:

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

In the topics tutorial, you learned that the `/turtle_teleop` node publishes commands on the `/turtle1/cmd_vel` topic to make the turtle move in turtlesim.

To see the data that `/turtle1/cmd_vel` is publishing, run the command:

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

Nothing will show up at first because no data is being published by the teleop. Return to the terminal where you ran the teleop and select it so it's active. Use the arrow keys to move the turtle around, and you will see data being published on the terminal running `ros2 topic echo`.

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

3 ros2 bag record

3.1 Record a single topic

To record the data published to a topic use the command syntax:

```
ros2 bag record <topic_name>
```

Before running this command on your chosen topic, open a new terminal and move into the `bag_files` directory you created earlier, because the rosbag file will save in the directory where you run it.

Run the command:

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

You will see the following messages in the terminal (the date and time will be different):

```
[INFO] [rosbag2_storage]: Opened database 'rosbag2_2019_10_11-05_18_45'.
[INFO] [rosbag2_transport]: Listening for topics...
[INFO] [rosbag2_transport]: Subscribed to topic '/turtle1/cmd_vel'
[INFO] [rosbag2_transport]: All requested topics are subscribed. Stopping discovery...
```

Now `ros2 bag` is recording the data published on the `/turtle1/cmd_vel` topic. Return to the teleop terminal and move the turtle around again. The movements don't matter, but try to make a recognizable pattern to see when you replay the data later.

Press `Ctrl+C` to stop recording.

The data will be accumulated in a new bag directory with a name in the pattern of `rosbag2_year_month_day-hour_minute_second`. This directory will contain a `metadata.yaml` along with the bag file in the recorded format.

3.2 Record multiple topics

You can also record multiple topics, as well as change the name of the file `ros2 bag` saves to.

Run the following command:

```
ros2 bag record -o subset /turtle1/cmd_vel /turtle1/pose
```

The `-o` option allows you to choose a unique name for your bag file. The following string, in this case `subset`, is the file name.

To record more than one topic at a time, simply list each topic separated by a space.

You will see the following message, confirming that both topics are being recorded.

```
[INFO] [rosbag2_storage]: Opened database 'subset'.
[INFO] [rosbag2_transport]: Listening for topics...
[INFO] [rosbag2_transport]: Subscribed to topic '/turtle1/cmd_vel'
[INFO] [rosbag2_transport]: Subscribed to topic '/turtle1/pose'
[INFO] [rosbag2_transport]: All requested topics are subscribed. Stopping discovery...
```

You can move the turtle around and press `Ctrl+C` when you're finished.

Note

There is another option you can add to the command, `-a`, which records all the topics on your system.

4 ros2 bag info

You can see details about your recording by running:

```
ros2 bag info <bag_file_name>
```

Running this command on the `subset` bag file will return a list of information on the file:

```
ros2 bag info subset
```

```
Files:             subset.db3
Bag size:          228.5 KiB
Storage id:        sqlite3
Duration:          48.47s
Start:             Oct 11 2019 06:09:09.12 (1570799349.12)
End                Oct 11 2019 06:09:57.60 (1570799397.60)
Messages:          3013
Topic information: Topic: /turtle1/cmd_vel | Type: geometry_msgs/msg/Twist | Count: 9 |
Serialization Format: cdr
                 Topic: /turtle1/pose | Type: turtlesim/msg/Pose | Count: 3004 | Serialization
Format: cdr
```

5 ros2 bag play

Before replaying the bag file, enter `Ctrl+C` in the terminal where the teleop is running. Then make sure your turtlesim window is visible so you can see the bag file in action.

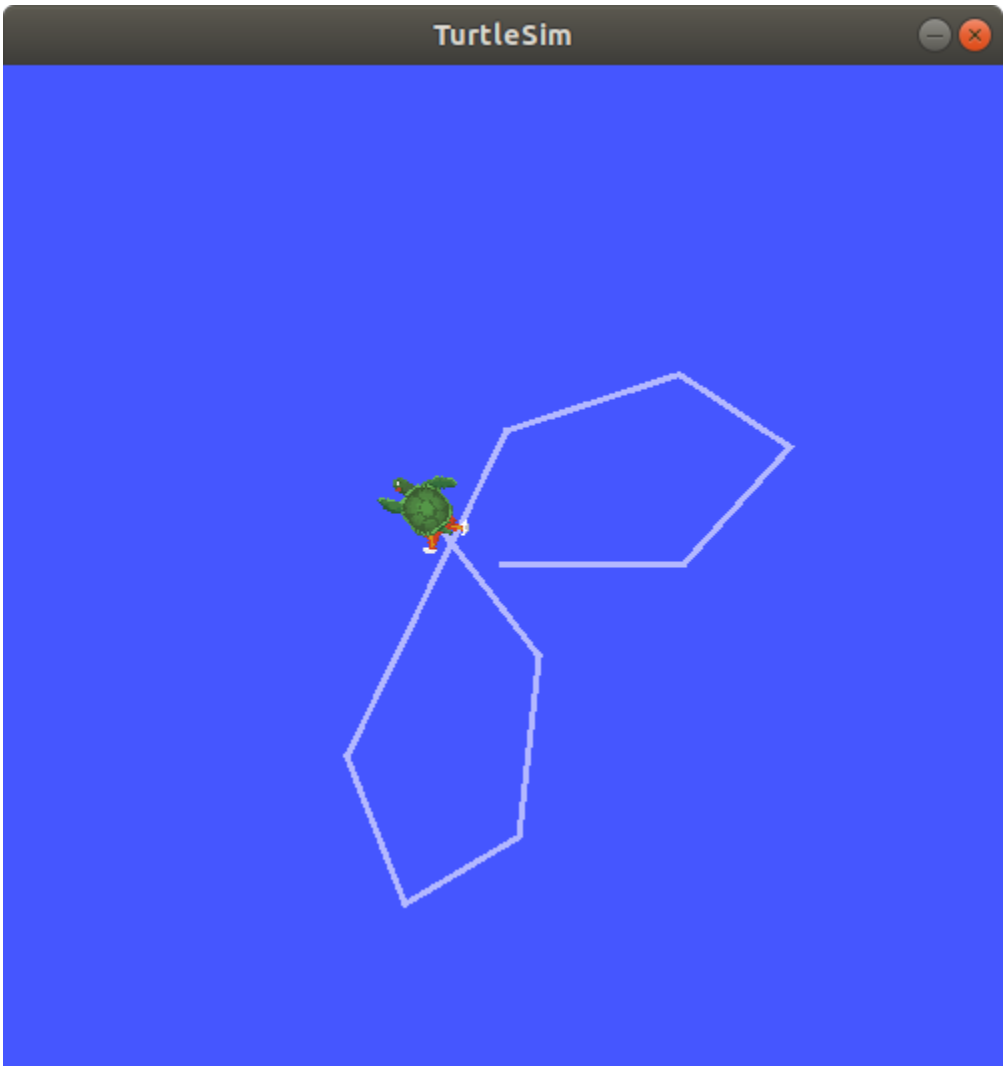
Enter the command:

```
ros2 bag play subset
```

The terminal will return the message:

```
[INFO] [rosbag2_storage]: Opened database 'subset'.
```

Your turtle will follow the same path you entered while recording (though not 100% exactly; turtlesim is sensitive to small changes in the system's timing).



Because the `subset` file recorded the `/turtle1/pose` topic, the `ros2 bag play` command won't quit for as long as you had turtlesim running, even if you weren't moving.

This is because as long as the `/turtlesim` node is active, it publishes data on the `/turtle1/pose` topic at regular intervals. You may have noticed in the `ros2 bag info` example result above that the `/turtle1/cmd_vel` topic's `Count` information was only 9; that's how many times we pressed the arrow keys while recording.

Notice that `/turtle1/pose` has a `Count` value of over 3000; while we were recording, data was published on that topic 3000 times.

To get an idea of how often position data is published, you can run the command:

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

Summary

You can record data passed on topics in your ROS 2 system using the `ros2 bag` command. Whether you're sharing your work with others or introspecting your own experiments, it's a great tool to know about.

Related content

A more thorough explanation of `ros2 bag` can be found in the README here. For more information on QoS compatibility and `ros2 bag`, see rosbag2: Overriding QoS Policies.

References

- ROS 2 Humble documentation: https://docs.ros.org/en/humble/
- Suggested ROS 2 book (introductory): “A Concise Introduction to Robot Programming with ROS2” by Francisco Martin Rico (https://www.routledge.com/A-Concise-Introduction-to-Robot-Programming-with-ROS2/Rico/p/book/9781032264653; https://github.com/fmrico/book_ros2)
- ROS 2 book (advanced): “A Very Informal Journey through ROS 2” by Marco Matteo Bassa (https://leanpub.com/averyinformaljourneythroughros2)
- Other ROS 1/2 books: https://wiki.ros.org/Books
- ROS2 demos: https://github.com/ros2/demos