Installation:

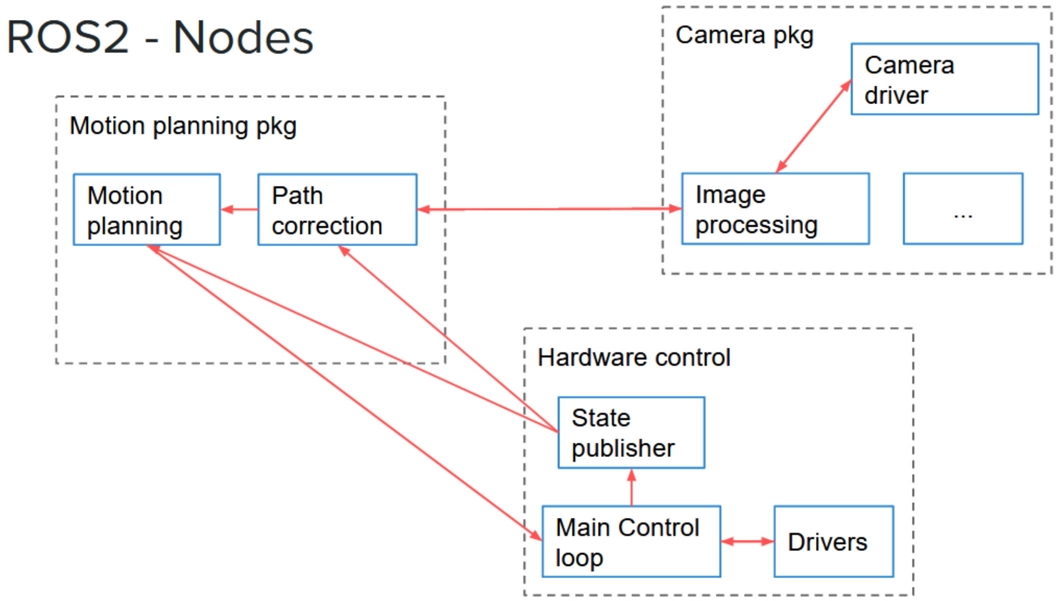
https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

https://docs.ros.org/en/foxy/Tutorials/Colcon-Tutorial.html

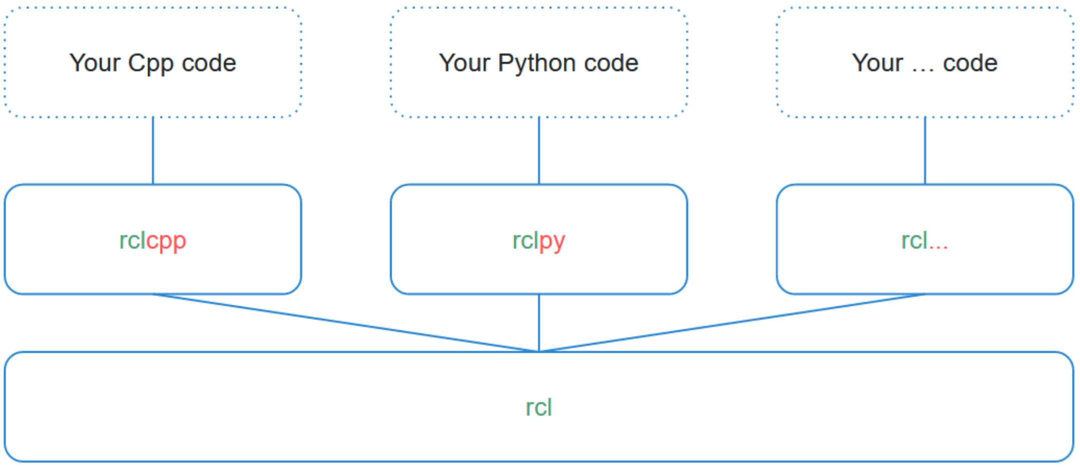
Before creating your first node you need to:

- Create a ROS2 workspace and source it.
- Create a (Python/Cpp) package.

Then, you write your node using the appropriate ROS2 client library: rclpy for Python, and rclcpp for Cpp. Both libraries will provide the same core functionalities.



ROS2 - Language Libraries



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

gedit ~/.bashrc
source /opt/ros/foxy/setup.bash
source ~/ros2_ws/install/setup.bash
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash

cd ros2_ws/src
ros2 pkg create my_py_pkg --build-type ament_python --dependencies rclpy
cd ..
colcon build
(colcon build –packages-select my_py_pkg)

cd ros2_ws/src
ros2 pkg create my_cpp_pkg --build-type ament_cmake --dependencies rclcpp
cd ..
colcon build



cd ros2_ws/src
ls
cd my_py_pkg/
ls
touch my_first_node.py
```

```
#!/usr/bin/env python3
import rclpy
```

```
from rclpy.node import Node


def main(args=None):
    rclpy.init(args=args)
    node = Node("my_test")
    node.get_logger().info("Hello ROS2")
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

```
chmod +x my_first_node.py
 ./my_first_node.py
```

edit setup.py

```
from setuptools import setup

package_name = 'my_py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main"
        ],
    },
)
```

```
cd ~/ros2_ws/install/my_py_pkg/lib/my_py_pkg$
./py_node
```

```
Ctrl+Alt+T
source .bashrc
```

```
ros2 run my_py_pkg py_node
```

Object oriented Programming in Python

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node


class MyNode(Node):

    def __init__(self):
        super().__init__("py_test")
        self.get_logger().info("Heelo ROS2")

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

change the program to

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node


class MyNode(Node):

    def __init__(self):
        super().__init__("py_test")
        self.get_logger().info("Hello ROS2")
        self.counter_ = 0
        self.create_timer(0.5, self.timer_callback)

    def timer_callback(self):
        self.counter_ += 1
        self.get_logger().info("Hello" + str(self.counter_))

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

```
if __name__ == '__main__':
    main()
```

```
cd ros2_ws/src/my_cpp_pkg/src/
touch my_first_node.cpp
```

write the code in the my_first_node.cpp file

Ctrl+Shift+P
C/C++ Edit Configuration JSON

creates a .vscode folder

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


int main(int argc, char **argv)
{
    rclcpp::init(argc,argv);
    auto node = std::make_shared<rclcpp::Node>("cpp_test");
    RCLCPP_INFO(node->get_logger(), "Cpp Node in ROS2");
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "/opt/ros/foxy/include"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "gnu17",
            "cppStandard": "gnu++14",
            "intelliSenseMode": "linux-gcc-x64"
        }
    ],
    "version": 4
}
```

add , "/opt/ros/foxy/include" for including rclcpp/rclcpp.hpp" file edit CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(my_cpp_pkg)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)

add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

install(TARGETS
  cpp_node
  DESTINATION lib/my_cpp_pkg
)
ament_package()
```

make the changes as shown in red.

```
Ctrl+Alt+T
cd ros2_ws
colcon build
Ctrl+Alt+T
cd install/my_cpp_pkg/lib/my_cpp_pkg
./cpp_node

Ctrl+Alt+T
source .bashrc
ros2 run my_cpp_pkg cpp_node
```

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

class MyNode: public rclcpp::Node
```

```
{
   public:
   MyNode():Node("cpp_test")
   {
      RCLCPP_INFO(this->get_logger(), "Cpp Node in ROS2");
   }

   private:
}
int main(int argc, char **argv)
{
   rclcpp::init(argc,argv);
   auto node = std::make_shared<MyNode>();
   rclcpp::spin(node);
   rclcpp::shutdown();
   return 0;
}
```

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

class MyNode: public rclcpp::Node

{
   public:
   MyNode() : Node("cpp_test")
   {
      RCLCPP_INFO(this->get_logger(), "Cpp Node in ROS2");
      timer_ = this->create_wall_timer(std::chrono::seconds(1),
                           std::bind(&MyNode::timerCallback, this));
   }

   private:

   void timerCallback()
   {
      RCLCPP_INFO(this->get_logger(),"Hello");
   }

   rclcpp::TimerBase::SharedPtr timer_;
};
int main(int argc, char **argv)
{
   rclcpp::init(argc,argv);
   auto node = std::make_shared<MyNode>();
   rclcpp::spin(node);
   rclcpp::shutdown();
   return 0;
}
```

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

class MyNode: public rclcpp::Node

{
   public:
   MyNode() : Node("cpp_test"), counter_(0)
   {
      RCLCPP_INFO(this->get_logger(), "Cpp Node in ROS2");
      timer_ = this->create_wall_timer(std::chrono::seconds(1),
                                  std::bind(&MyNode::timerCallback, this));
   }

   private:

   void timerCallback()
   {
      counter_++;
      RCLCPP_INFO(this->get_logger(),"Hello %d", counter_);
   }

   rclcpp::TimerBase::SharedPtr timer_;
   int counter_;
};
int main(int argc, char **argv)
{
   rclcpp::init(argc,argv);
   auto node = std::make_shared<MyNode>();
   rclcpp::spin(node);
   rclcpp::shutdown();
   return 0;
}
```

OOP Python Code Template for Nodes

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node


class MyCustomNode(Node): # MODIFY NAME
        def __init__(self):
                super().__init__("node_name") # MODIFY NAME


def main(args=None):
      rclpy.init(args=args)
      node = MyCustomNode() # MODIFY NAME
      rclpy.spin(node)
```

```
        rclpy.shutdown()

if __name__ == "__main__":
          main()
```

OOP C++ Code Template for Nodes

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

 class MyCustomNode : public rclcpp::Node // MODIFY NAME
 {
    public:
        MyCustomNode() : Node("node_name") // MODIFY NAME
        {
        }

    private:
 };

int main(int argc, char **argv)
{
      rclcpp::init(argc, argv);
      auto node = std::make_shared<MyCustomNode>(); // MODIFY NAME
      rclcpp::spin(node);
      rclcpp::shutdown();
      return 0;
 }
```

```
ros2 run my_cpp_pkg cpp_node
ros2 node list
ros2 node info
ros2 node info /py_test
ros2 node -h
rqt_graph
ros2 run my_cpp_pkg cpp_node
```

changing the name of node

```
ros2 run my_py_pkg py_node --ros-args --remap __node:=test
 ros2 run my_py_pkg py_node --ros-args -r __node:=node
```

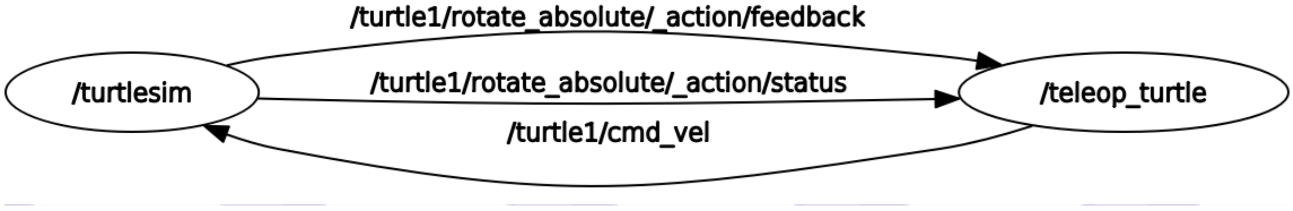
```
colcon build --packages-select my_py_pkg
```

```
colcon build --packages-select my_py_pkg --symlink-install
```

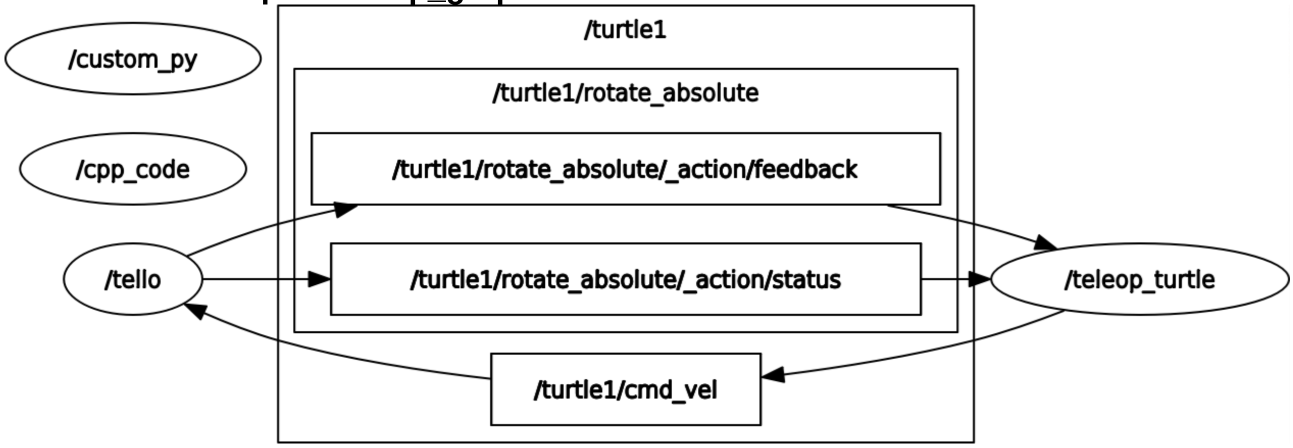
any changes in the code does not require colcon build every time (works only in python)

Turtlesim

```
sudo apt install ros-foxy-turtlesim
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
rqt_graph
```


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

Create nodes as per this rqt_graph



```
rqt_graph
ros2 run my_py_pkg py_node --ros-args -r __node:=custom_py
ros2 run my_cpp_pkg cpp_node --ros-args -r __node:=cpp_code
ros2 run turtlesim turtlesim_node --ros-args -r __node:=tello
ros2 run turtlesim turtle_teleop_key
```

refresh the rqt_gragh window:

a topic is:

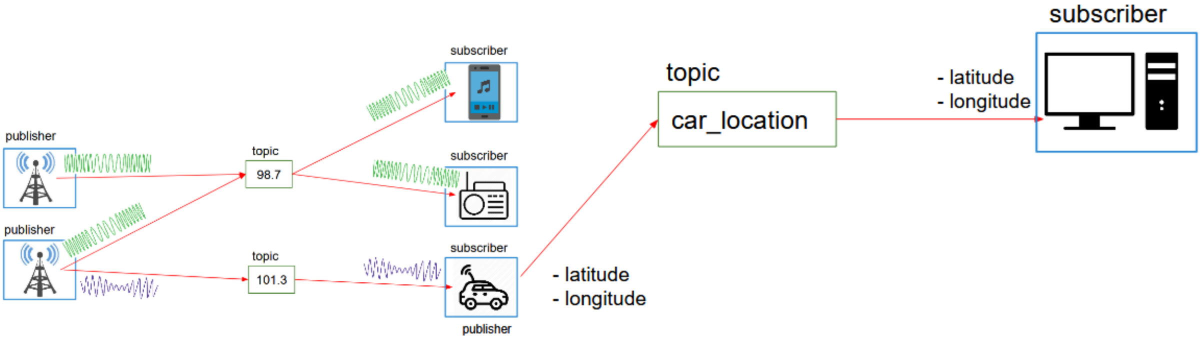
- A named bus over which nodes exchange messages
- Used for unidirectional data streams
- Anonymous: publishers don’t know who is subscribing, and subscribers don’t know who is publishing.

To implement topics in your ROS2 application:

- First create a node (or start from an existing one), then inside your node you can create any number of publishers/subscribers.
- A publisher and subscriber must publish/subscribe to the same topic name, and use the same data type. Those are the 2 conditions for successful topic communication.

- Then, once you've added some publishers/subscribers in your nodes, just launch your nodes, and the communication starts! You can debug them using the "ros2" command line tool, as well as rqt.

ROS2 - Topics



Publisher – Subscriber Nodes (Python)

```
cd ros2_ws/src/my_py_pkg//my_py_pkg/
touch robot_news_station.py
chmod +x robot_news_station.py
ros2 interface show example_interfaces/msg/String
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String

class RobotNewsStation(Node):

    def __init__(self):
        super().__init__("robot_news_station")
        self.robot_name_="ROBOT"
        self.publisher_ = self.create_publisher(String, "robot_news", 10)
        self.timer_ = self.create_timer(0.5, self.publish_news)
        self.get_logger().info("Node Started")


    def publish_news(self):
        msg = String()
        msg.data = "Hello " + str(self.robot_name_)
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = RobotNewsStation()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

setup.py

```
from setuptools import setup

package_name = 'my_py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main"
        ],
    },
)
```

```
colcon build --packages-select my_py_pkg –symlink-install
new terminal- source ~/.bashrc
ros2 run my_py_pkg robot_news_station

new terminal- source ~/.bashrc
ros2 topic echo /robot_news
```

Subscriber node

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch smartphone.py
```

```
chmod +x smartphone.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import String

class SmartPhoneNode(Node): # MODIFY NAME
    def __init__(self):
        super().__init__("smartphone") # MODIFY NAME
        self.subscriber_ = self.create_subscription(String, "robot_news",
self.callback_robot_news, 10)
        self.get_logger().info("Smartphone Node Started")

    def callback_robot_news(self, msg):
        self.get_logger().info(msg.data)


def main(args=None):
    rclpy.init(args=args)
    node = SmartPhoneNode() # MODIFY NAME
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

```
from setuptools import setup

package_name = 'my_py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main"
        ],
    },
)
```

```
colcon build --packages-select my_py_pkg --symlink-install
```

```
new terminal- source ~./bashrc
ros2 run my_py_pkg robot_news_station
```

```
new terminal- source ~./bashrc
ros2 run my_py_pkg smartphone
```

```
ros2 node list
ros2 topic list
```

```
cd ros2_ws/src/my_cpp_pkg/src
touch robot_news_station.cpp
```

```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/msg/string.hpp"

class RobotNewsStationNode : public rclcpp::Node // MODIFY NAME
{
public:
    RobotNewsStationNode() : Node("robot_news_station"), robot_name_("ROBOT") //
MODIFY NAME
    {
        publisher_ =
this->create_publisher<example_interfaces::msg::String>("robot_news",10);
        timer_ = this->create_wall_timer(std::chrono::milliseconds(580),
                      std::bind(&RobotNewsStationNode::publishNews, this));
        RCLCPP_INFO(this->get_logger(),"Started");
    }

private:
    void publishNews()
    {
        auto msg = example_interfaces::msg::String();
        msg.data = std::string("Hello") + robot_name_;
        publisher_->publish(msg);
    }

    std::string robot_name_;
    rclcpp::Publisher<example_interfaces::msg::String>::SharedPtr publisher_;
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<RobotNewsStationNode>(); // MODIFY NAME
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(my_cpp_pkg)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(example_interfaces REQUIRED)

add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

add_executable(robot_news_station src/robot_news_station.cpp)
ament_target_dependencies(robot_news_station rclcpp example_interfaces)


install(TARGETS
  cpp_node
  robot_news_station
  DESTINATION lib/my_cpp_pkg
)
ament_package()
```

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_cpp_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="asha@todo.todo">asha</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>example_interfaces</depend>
  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

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

```
ros2 run my_cpp_pkg  robot_news_station
new terminal – source ~/.bashrc
ros2 topic echo /robot_news
```

```
touch smartphone.cpp
```

```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/msg/string.hpp"

class SmartphoneNode : public rclcpp::Node // MODIFY NAME
{
    public:
        SmartphoneNode() : Node("node_name") // MODIFY NAME
        {
            subscriber_ =
this->create_subscription<example_interfaces::msg::String>("robot_news",10,

            std::bind(&SmartphoneNode::callbackRobotNews, this, std::placeholders::_1));
            RCLCPP_INFO(this->get_logger(),"Node Started");
        }

    private:

    void callbackRobotNews(const example_interfaces::msg::String::SharedPtr msg)
    {
        RCLCPP_INFO(this->get_logger(), "%s", msg->data.c_str());

    }
```

```
        rclcpp::Subscription<example_interfaces::msg::String>::SharedPtr subscriber_;
};

int main(int argc, char **argv)
{
      rclcpp::init(argc, argv);
      auto node = std::make_shared<SmartphoneNode>(); // MODIFY NAME
      rclcpp::spin(node);
      rclcpp::shutdown();
      return 0;
 }
```

```
cmake_minimum_required(VERSION 3.5)
project(my_cpp_pkg)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(example_interfaces REQUIRED)

add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

add_executable(robot_news_station src/robot_news_station.cpp)
ament_target_dependencies(robot_news_station rclcpp example_interfaces)


add_executable(smartphone src/smartphone.cpp)
ament_target_dependencies(smartphone rclcpp example_interfaces)

install(TARGETS
  cpp_node
  robot_news_station
  smartphone
  DESTINATION lib/my_cpp_pkg
)
ament_package()
```

```
colcon build –packages-select my_cpp_pkg
new terminal – source ~/.bashrc
ros2 run my_cpp_pkg  robot_news_station
new terminal – source ~/.bashrc
ros2 run my_cpp_pkg smartphone
```

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
ros2 node info /teleop_turtle
ros2 node info /turtlesim
ros2 interface show geometry_msgs/msg/Twist
ros2 interface show geometry_msgs/msg/Vector3
ros2 topic list
ros2 topic echo /turtle1/cmd_vel
rqt_graph
```

Exercise:

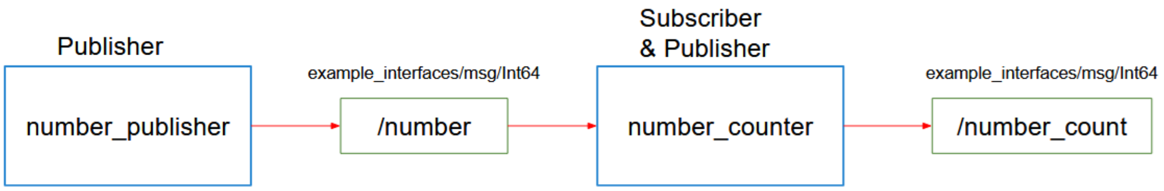
Create 2 nodes from scratch. First node has 1 publisher, the second has 1 publisher & 1 subscriber.

- The number_publisher node publishes a number (always the same) on the "/number" topic, with the existing type example_interfaces/msg/Int64.
- The number_counter node subscribes to the "/number" topic. It keeps a counter variable. Every time a new number is received, it's added to the counter. The node also has a publisher on the "/number_count" topic. When the counter is updated, the publisher directly publishes the new value on the topic.

A few hints:

- Check what to put into the example_interfaces/msg/Int64 with the "ros2 interface show" command line tool.
- It may be easier to do the activity in this order: first create the number_publisher node, check that the publisher is working with "ros2 topic". Then create the number_counter, focus on the subscriber. And finally create the last publisher.
- In the number_counter node, the publisher will publish messages directly from the subscriber callback.

ROS2 Topics - Activity



Publisher

number_publisher

example_interfaces/msg/Int64

/number

Subscriber
& Publisher

number_counter

example_interfaces/msg/Int64

/number_count

Solution:

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch number_publisher.py
chmod +x number_publisher.py
```

```
 #!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64

class NumberPublisherNode(Node): # MODIFY NAME
    def __init__(self):
        super().__init__("number_publisher") # MODIFY NAME
        self.number_ =2
        self.number_publisher_ = self.create_publisher(Int64, "number", 10)
        self.number_timer_ = self.create_timer(1.0, self.publish_number)
        self.get_logger().info("Number Publisher has started!!")
    def publish_number(self):
        msg =  Int64()
        msg.data  = self.number_
        self.number_publisher_.publish(msg)


def main(args=None):
        rclpy.init(args=args)
        node = NumberPublisherNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
        main()
```

```
entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main",
            "number_publisher = my_py_pkg.number_publisher:main"
        ],
```

terminal 1:

```
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg number_publisher
```

terminal 2:

```
ros2 topic list
ros2 topic info /number
ros2 topic echo /number
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64
```

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
 touch number_counter.py
chmod +x number_counter.py
```

```
class NumberCounterNode(Node):
    def __init__(self):
        super().__init__("number_counter")
        self.counter_ = 0
        self.number_subscriber_ = self.create_subscription(Int64, "number",
self.callback_number, 10)
        self.get_logger().info("Node started")


    def callback_number(self, msg):
        self.counter_ += msg.data
        self.get_logger().info(str(self.counter_))


def main(args=None):
        rclpy.init(args=args)
        node = NumberCounterNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
          main()
```

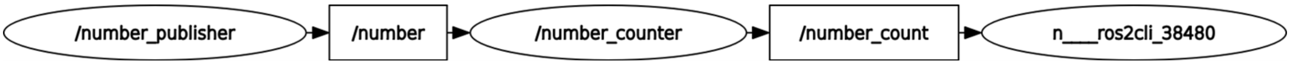
```
entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main",
            "number_publisher = my_py_pkg.number_publisher:main",
            "number_counter = my_py_pkg.number_counter:main"
        ],
```

```
cd ~/ros2_ws/
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg number_publisher
ros2 run my_py_pkg number_counter
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64


class NumberCounterNode(Node):
    def __init__(self):
        super().__init__("number_counter")
        self.counter_ = 0
        self.number_count_publisher_ = self.create_publisher(Int64, "number_count", 10)
        self.number_subscriber_ = self.create_subscription(Int64, "number",
self.callback_number, 10)
        self.get_logger().info("Node started")


    def callback_number(self, msg):
        self.counter_ += msg.data
        new_msg = Int64()
        new_msg.data = self.counter_
        self.number_count_publisher_.publish(new_msg)
        self.get_logger().info(str(self.counter_))


def main(args=None):
        rclpy.init(args=args)
        node = NumberCounterNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()
```

```
if __name__ == "__main__":
        main()
```

```
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg number_publisher


ros2 run my_py_pkg number_counter

ros2 topic list
ros2 topic echo /number_count

rqt_graph
```



```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch add_two_ints_server.py
chmod +x add_two_ins_server.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class AddTwoIntsServerNode(Node): # MODIFY NAME
   def __init__(self):
      super().__init__("add_two_ints_server")
      self.server_ = self.create_service(AddTwoInts, "add_two_ints",
self.callback_add_two_ints)
      self.get_logger().info("Add two ints server has been started")


   def callback_add_two_ints(self, request, response):
      response.sum = request.a + request.b
      self.get_logger().info(str(request.a)+ " + " + str(request.b) + " = " + str(response.sum))
      return response


def main(args=None):
      rclpy.init(args=args)
      node = AddTwoIntsServerNode() # MODIFY NAME
      rclpy.spin(node)
      rclpy.shutdown()

if __name__ == "__main__":
        main()
```

```
entry_points={
    'console_scripts': [
      "py_node = my_py_pkg.my_first_node:main",
      "robot_news_station = my_py_pkg.robot_news_station:main",
      "smartphone = my_py_pkg.smartphone:main",
      "number_publisher = my_py_pkg.number_publisher:main",
      "number_counter = my_py_pkg.number_counter:main",
      "add_two_ints_server = my_py_pkg.add_two_ints_server:main"
    ],
```

```
ros2 interface show example_interfaces/srv/AddTwoInts
source ~/.bashrc
colcon build --packages-select my_py_pkg --symlink-install
source ~/.bashrc
ros2 run my_py_pkg add_two_ints_server
source ~/.bashrc
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 3, b: 4}"
```

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch add_two_ints_client_no_oop.py
touch add_two_ints_client.py
chmod +x add_two_ints_client.py
chmod +x add_two_ints_client_no_oop.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts


def main(args=None):
    rclpy.init(args=args)
    node = Node("add_two_ints_no_oop") # MODIFY NAME


    client = node.create_client(AddTwoInts, "add_two_ints")
    while not client.wait_for_service(1.0):
      node.get_logger().warn("Waiting for Server Add Two Ints")
    request = AddTwoInts.Request()
    request.a = 3
    request.b = 4
    future = client.call_async(request)
    rclpy.spin_until_future_complete(node, future)
    try:
      response = future.result()
      node.get_logger().info(str(request.a)+ " + " + str(request.b) + " = " +
str(response.sum))

    except Exception as e:
      node.get_logger().error("Service call failed %r" % (e,))
    rclpy.shutdown()

if __name__ == "__main__":
     main()
```

```
entry_points={
    'console_scripts': [
      "py_node = my_py_pkg.my_first_node:main",
      "robot_news_station = my_py_pkg.robot_news_station:main",
      "smartphone = my_py_pkg.smartphone:main",
      "number_publisher = my_py_pkg.number_publisher:main",
      "number_counter = my_py_pkg.number_counter:main",
      "add_two_ints_server = my_py_pkg.add_two_ints_server:main",
      "add_two_ints_client_no_oop = my_py_pkg.add_two_ints_client_no_oop:main "
    ],
```

```
Terminal 1:
cd ros2_ws/
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg add_two_ints_server

Terminal 2: source ~/.bashrc
ros2 run my_py_pkg add_two_ints_client_no_oop

Terminal 3: source ~/.bashrc
ros2 run my_py_pkg add_two_ints_server
```

add_two_ints_client.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts
from functools import partial

class AddTwoIntClientNode(Node):
    def __init__(self):
        super().__init__("add_two_ints_client")
        self.call_add_two_int_server(6,7)


    def call_add_two_int_server(self, a, b):
        client = self.create_client(AddTwoInts, "add_two_ints")
        while not client.wait_for_service(1.0):
            self.get_logger().warn("Waiting for Server Add Two Ints")
        request = AddTwoInts.Request()
        request.a = a
        request.b = b
        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_two_ints, a=a, b=b))

    def callback_call_two_ints(self, future, a, b):
        try:
            response = future.result()
            self.get_logger().info(str(a)+ " + " + str(b) + " = " + str(response.sum))

        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))


def main(args=None):
    rclpy.init(args=args)
    node = AddTwoIntClientNode() # MODIFY NAME
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == "__main__":
     main()
```

```
entry_points={
    'console_scripts': [
      "py_node = my_py_pkg.my_first_node:main",
      "robot_news_station = my_py_pkg.robot_news_station:main",
      "smartphone = my_py_pkg.smartphone:main",
      "number_publisher = my_py_pkg.number_publisher:main",
      "number_counter = my_py_pkg.number_counter:main",
      "add_two_ints_server = my_py_pkg.add_two_ints_server:main",
      "add_two_ints_client_no_oop = my_py_pkg.add_two_ints_client_no_oop:main",
      "add_two_ints_client= my_py_pkg.add_two_ints_client:main"
    ],
```

```
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg add_two_ints_client
ros2 run my_py_pkg add_two_ints_server
```

```
cd ros2_ws/src/my_cpp_pkg/src/
touch add_two_ints_server.cpp
```

```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/srv/add_two_ints.hpp"
using std::placeholders::_1;
using std::placeholders::_2;


class AddTwoIntServerNode : public rclcpp::Node // MODIFY NAME
 {
    public:
        AddTwoIntServerNode() : Node("add_two_ints_server") // MODIFY NAME
        {
            server_ = this->create_service<example_interfaces::srv::AddTwoInts>(
            "add_two_ints", std::bind(&AddTwoIntServerNode::callbackAddTwoInts, this, _1, _2));
            RCLCPP_INFO(this->get_logger(), "Service Started");
        }

    private:
        void callbackAddTwoInts(const example_interfaces::srv::AddTwoInts::Request::SharedPtr request,
        const example_interfaces::srv::AddTwoInts::Response::SharedPtr response)
        {
            response->sum = request->a + request->b;
            RCLCPP_INFO(this->get_logger(), "%d + %d= %d", request->a, request->b, response->sum);
        }
        rclcpp::Service<example_interfaces::srv::AddTwoInts>::SharedPtr server_;
 };

int main(int argc, char **argv)
{
      rclcpp::init(argc, argv);
      auto node = std::make_shared<AddTwoIntServerNode>(); // MODIFY NAME
      rclcpp::spin(node);
      rclcpp::shutdown();
      return 0;
 }
```

In CmakeLists.txt

```
add_executable(add_two_ints_server src/add_two_ints_server.cpp)
ament_target_dependencies(add_two_ints_server rclcpp example_interfaces)

install(TARGETS
  cpp_node
  robot_news_station
  smartphone
  add_two_ints_server
  DESTINATION lib/my_cpp_pkg
)
```

```
colcon build --packages-select my_cpp_pkg --symlink-install
ros2 service list
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 5, b: 8}"

ros2 run my_cpp_pkg add_two_ints_server
```

```
cd ros2_ws/src/my_cpp_pkg/src/
touch add_two_int_client_no_oop.cpp
touch add_two_ints_client.cpp
```

add_two_int_client_no_oop.cpp

```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/srv/add_two_ints.hpp"

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("add_two_int_client_no_oop"); //
MODIFY NAME

    auto client =
node->create_client<example_interfaces::srv::AddTwoInts>("add_two_ints");
    while(!client->wait_for_service(std::chrono::seconds(1)))
    {
        RCLCPP_WARN(node->get_logger(), "waitig for server");
    }
    auto request = std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
    request->a = 3;
    request->b = 8;
    auto future = client->async_send_request(request);
    if (rclcpp::spin_until_future_complete(node, future) ==
rclcpp::executor::FutureReturnCode::SUCCESS)
    {
        RCLCPP_INFO(node->get_logger(), "%d + %d = %d", request->a, request->b,
future.get()->sum);
    }
    else
    {
        RCLCPP_ERROR(node->get_logger(), "Error");
    }

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

```
add_executable(add_two_int_client_no_oop src/add_two_int_client_no_oop)
ament_target_dependencies(add_two_int_client_no_oop rclcpp example_interfaces)
install(TARGETS
  cpp_node
  robot_news_station
  smartphone
  add_two_ints_server
  add_two_int_client_no_oop
  DESTINATION lib/my_cpp_pkg
)
```

CmakeLists.txt

```
colcon build --packages-select my_cpp_pkg --symlink-install
ros2 run my_cpp_pkg add_two_int_client_no_oop

ros2 run my_cpp_pkg add_two_ints_server
```

```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/srv/add_two_ints.hpp"
class AddTwoIntsClientNode : public rclcpp::Node // MODIFY NAME
{
    public:
        AddTwoIntsClientNode() : Node("node_name") // MODIFY NAME
        {
            thread1_ = std::thread(std::bind(&AddTwoIntsClientNode::callAddTwoIntService,
this, 1,4));
        }



void callAddTwoIntService(int a, int b)
{
    auto client = this->create_client<example_interfaces::srv::AddTwoInts>("add_two_ints");
    while(!client->wait_for_service(std::chrono::seconds(1)))
    {
        RCLCPP_WARN(this->get_logger(), "waiting for server");
    }
    auto request = std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
    request->a = a;
    request->b = b;
    auto future = client->async_send_request(request);
    try
    {
        auto response = future.get();
        RCLCPP_INFO(this->get_logger(), "%d + %d = %d", a, b, response->sum);
    }
    catch (const std::exception &e)
    {
        RCLCPP_ERROR(this->get_logger(),"Service call failed");
    }
}
   private:

   std::thread thread1_;
};

int main(int argc, char **argv)
{
      rclcpp::init(argc, argv);
      auto node = std::make_shared<AddTwoIntsClientNode>(); // MODIFY NAME
```

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

```
add_executable(add_two_ints_client src/add_two_ints_client.cpp)
ament_target_dependencies(add_two_ints_client rclcpp example_interfaces)
```

```
install(TARGETS
  cpp_node
  robot_news_station
  smartphone
  add_two_ints_server
  add_two_int_client_no_oop
  add_two_ints_client
  DESTINATION lib/my_cpp_pkg
)
```

```
colcon build --packages-select my_cpp_pkg --symlink-install
ros2 run my_cpp_pkg add_two_ints_client
```

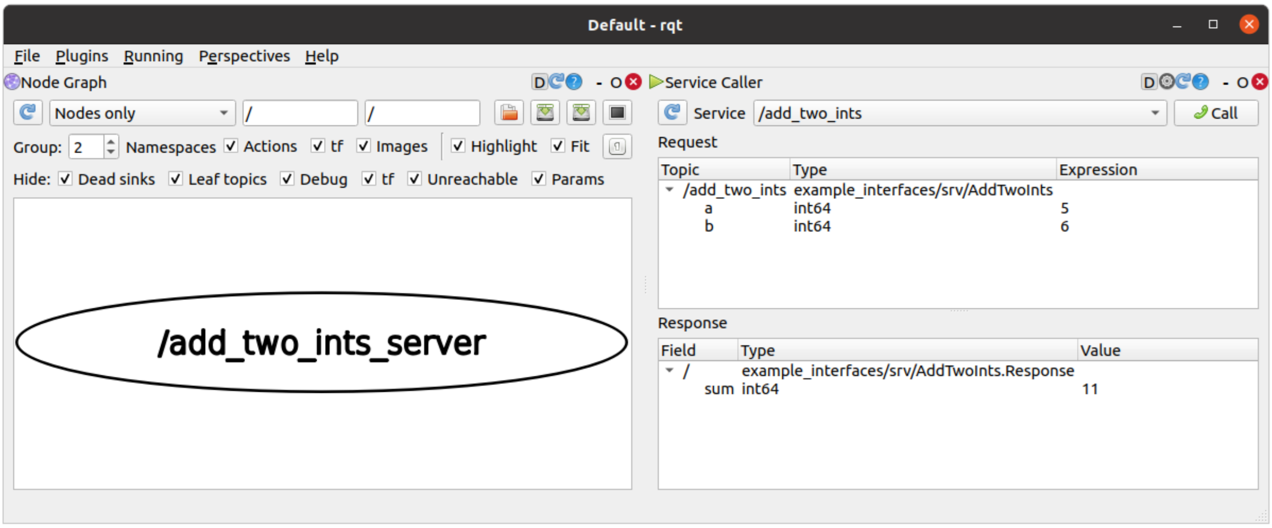
```
ros2 run my_cpp_pkg add_two_ints_server
```

Services:

```
ros2 run my_cpp_pkg add_two_ints_server
```

```
ros2 node list
ros2 service list
 ros2 service type /add_two_ints
ros2 interface show example_interfaces/srv/AddTwoInts
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 3, b: 4}"
rqt
```

plugins→services→service caller
service - /add_two_ints
Enter the values under Expression for a and b
Click call
Response is viewed in the second window



Nodes only / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

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

Service /add_two_ints

Request

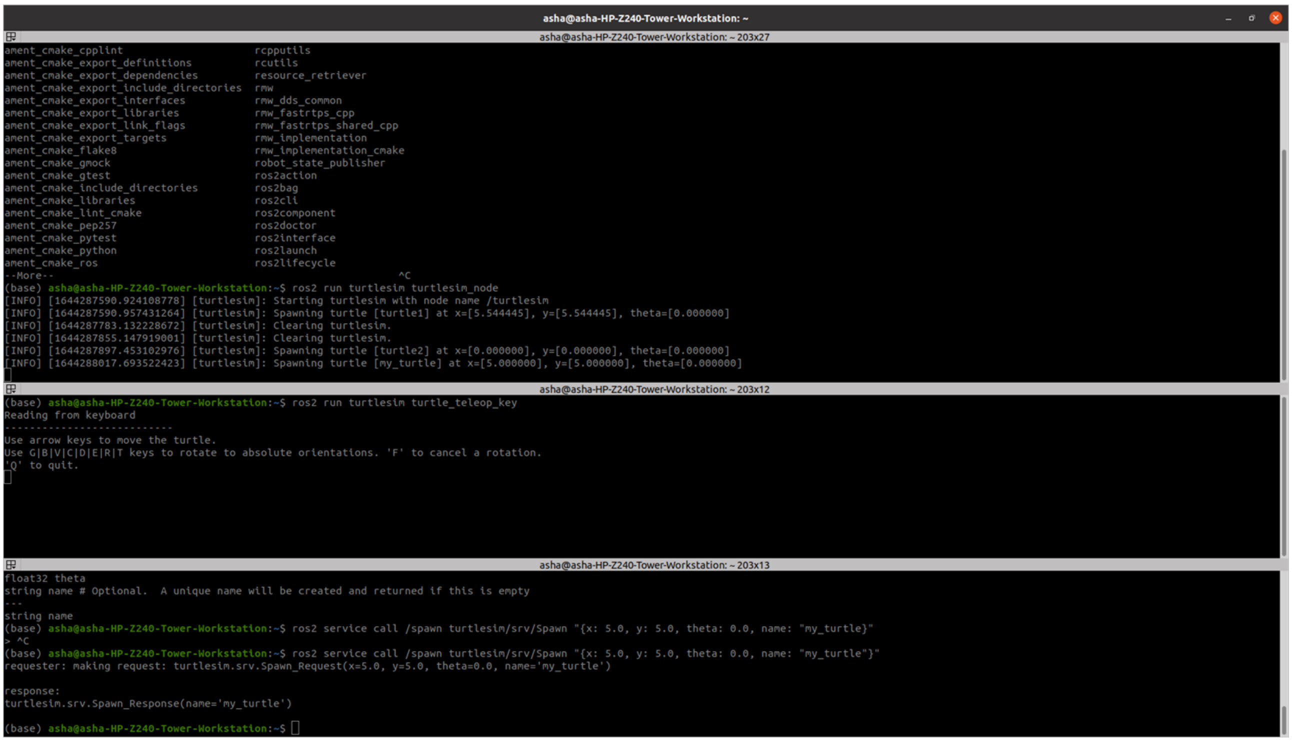
Topic	Type	Expression
/add_two_ints	example_interfaces/srv/AddTwoInts	
a	int64	5
b	int64	6

Response

Field	Type	Value
/	example_interfaces/srv/AddTwoInts.Response	
sum	int64	11

turtlesim services:

ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
ros2 service list
ros2 service type /clear
ros2 interface show std_srvs/srv/Empty
ros2 service call /clear std_srvs/srv/Empty
ros2 service type /spawn
ros2 interface show turtlesim/srv/Spawn
ros2 service call /spawn turtlesim/srv/Spawn
ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.0, y: 5.0, theta: 0.0, name: "my_turtle"}"

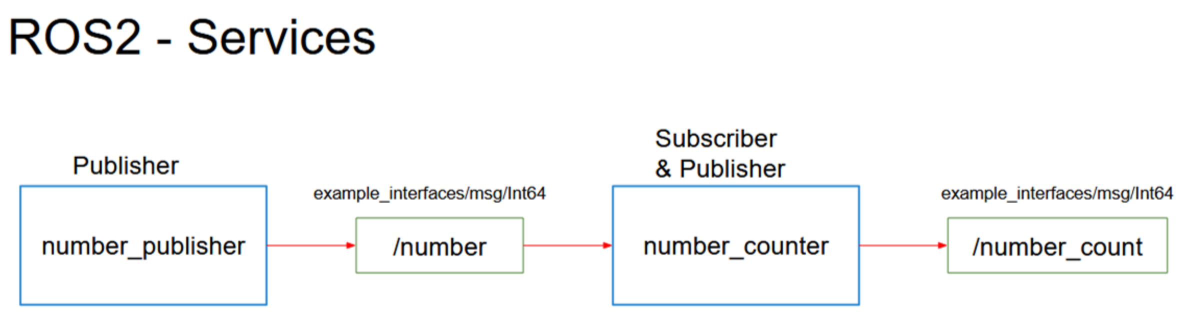

asha@asha-HP-Z240-Tower-Workstation: ~
asha@asha-HP-Z240-Tower-Workstation: ~ 203x27

```
ament_cmake_cpplint                      rcpputils
ament_cmake_export_definitions           rcutils
ament_cmake_export_dependencies          resource_retriever
ament_cmake_export_include_directories   rmw
ament_cmake_export_interfaces            rmw_dds_common
ament_cmake_export_libraries             rmw_fastrtps_cpp
ament_cmake_export_link_flags            rmw_fastrtps_shared_cpp
ament_cmake_export_targets               rmw_implementation
ament_cmake_flake8                       rmw_implementation_cmake
ament_cmake_gmock                        robot_state_publisher
ament_cmake_gtest                        ros2action
ament_cmake_include_directories          ros2bag
ament_cmake_libraries                    ros2cli
ament_cmake_lint_cmake                   ros2component
ament_cmake_pep257                       ros2doctor
ament_cmake_pytest                       ros2interface
ament_cmake_python                       ros2launch
ament_cmake_ros                          ros2lifecycle
--More--                                                          ^C
(base) asha@asha-HP-Z240-Tower-Workstation:~$ ros2 run turtlesim turtlesim_node
[INFO] [1644287590.924108778] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1644287590.957431264] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
[INFO] [1644287783.132228672] [turtlesim]: Clearing turtlesim.
[INFO] [1644287855.147919001] [turtlesim]: Clearing turtlesim.
[INFO] [1644287897.453102976] [turtlesim]: Spawning turtle [turtle2] at x=[0.000000], y=[0.000000], theta=[0.000000]
[INFO] [1644288017.693522423] [turtlesim]: Spawning turtle [my_turtle] at x=[5.000000], y=[5.000000], theta=[0.000000]
```

```
(base) asha@asha-HP-Z240-Tower-Workstation:~$ 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.
```

```
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
(base) asha@asha-HP-Z240-Tower-Workstation:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.0, y: 5.0, theta: 0.0, name: "my_turtle}"
> ^C
(base) asha@asha-HP-Z240-Tower-Workstation:~$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.0, y: 5.0, theta: 0.0, name: "my_turtle"}"
requester: making request: turtlesim.srv.Spawn_Request(x=5.0, y=5.0, theta=0.0, name='my_turtle')

response:
turtlesim.srv.Spawn_Response(name='my_turtle')

(base) asha@asha-HP-Z240-Tower-Workstation:~$
```

Exercises:



ROS2 - Services

Publisher

number_publisher

example_interfaces/msg/Int64

/number

Subscriber
& Publisher

number_counter

example_interfaces/msg/Int64

/number_count

The node “number_publisher” publishes a number on the /”number” topic.

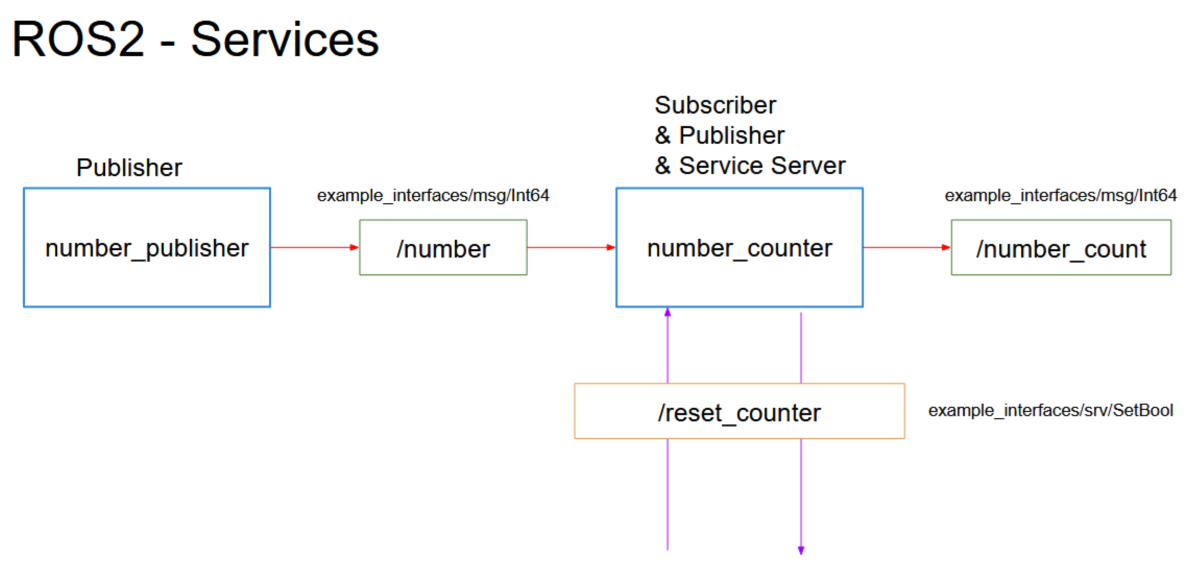
The node “number_counter” gets the number, adds it to a counter, and publishes the counter on the “/number_count” topic.

Add the following ros2 services

Add a functionality to reset the counter to zero:

- Create a service server inside the “number_counter” node.
- Service name: “/reset_counter”
- Service type: example_interfaces/srv/SetBool. Use “ros2 interface show” to discover what’s inside!
- When the server is called, you check the boolean data from the request. If true, you set the counter variable to 0.

We will then call the service directly from the command line. You can also decide - for more practice - to create your own custom node to call this “/reset_counter” service.



ROS2 - Services

Publisher

number_publisher

example_interfaces/msg/Int64

/number

Subscriber
& Publisher
& Service Server

number_counter

example_interfaces/msg/Int64

/number_count

/reset_counter

example_interfaces/srv/SetBool

Add a functionality to reset the counter to zero:

- Create a service server inside the “number_counter” node.
- Service name: “/reset_counter”
- Service type: example_interfaces/srv/SetBool. Use “ros2 interface show” to discover what’s inside!
- When the server is called, you check the boolean data from the request. If true, you set the counter variable to 0.

We will then call the service directly from the command line. You can also decide - for more practice - to create your own custom node to call this “/reset_counter” service.

Solution:

edit the code number_counter.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64
from example_interfaces.srv import SetBool

class NumberCounterNode(Node):
    def __init__(self):
        super().__init__("number_counter")
        self.counter_ = 0
        self.number_count_publisher_ = self.create_publisher(Int64, "number_count", 10)
        self.number_subscriber_ = self.create_subscription(Int64, "number",
self.callback_number, 10)
```

```
        self.reset_counter_service_ = self.create_service(SetBool, "reset_counter",
self.callback_reset_counter)
        self.get_logger().info("Node started")


    def callback_number(self, msg):
        self.counter_ += msg.data
        new_msg = Int64()
        new_msg.data = self.counter_
        self.number_count_publisher_.publish(new_msg)
        self.get_logger().info(str(self.counter_))

    def callback_reset_counter(self, request, response):
        if request.data:
            self.counter_ = 0
            response.success = True
            response.message = "Counter is reset"
        else:
            response.success = False
            response.message = "Counter is not reset"
        return response


def main(args=None):
        rclpy.init(args=args)
        node = NumberCounterNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
          main()
```

```
ros2 interface show example_interfaces/srv/SetBool
cd ros2_ws/
colcon build --packages-select my_py_pkg

 ros2 run my_py_pkg number_counter

ros2 topic list
ros2 topic echo /number_count
ros2 run my_py_pkg number_publisher

ros2 service call /reset_counter example_interfaces/srv/SetBool "{data: False}"
ros2 service call /reset_counter example_interfaces/srv/SetBool "{data: True}"
```

Summary:

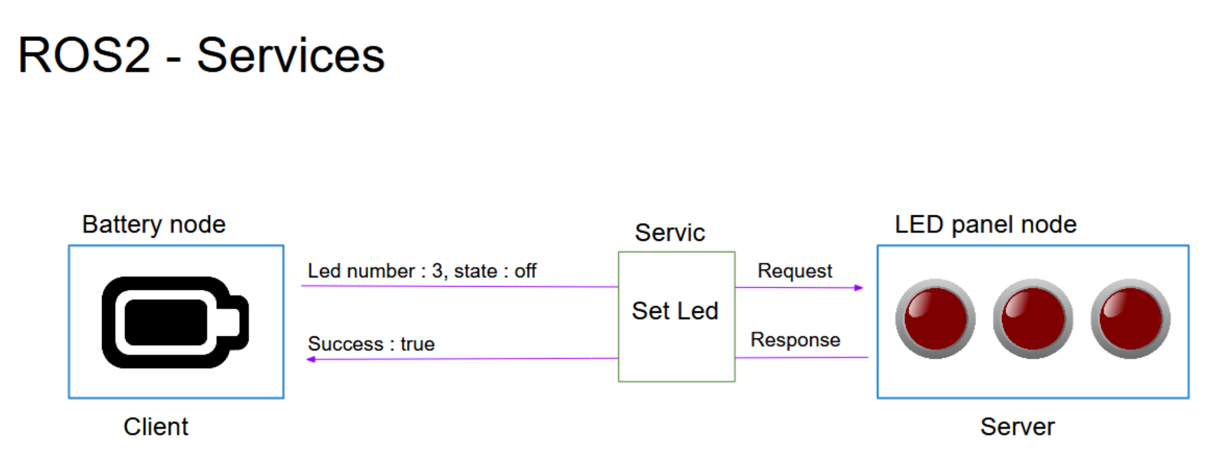
Services are:

- Used for client/server types of communication.
- Synchronous or asynchronous (though it’s recommended to use them asynchronously, even if you decide to wait after in the thread).

- Anonymous: a client does not know which node is behind the service, it just calls the service. And the server does not know which nodes are clients, it just receives requests and responds to them.

To implement Services inside your nodes:

- Create a node or start from an existing one. Add as many Service servers as you want (all with different names)
- When you call a Service server from a Service client, make sure that the Service name, as well as the Service type (request + response) are identical.
- You can only create one server for a Service, but you can create many clients.



ROS2 - Services

ROS2 interfaces:

https://github.com/ros2/example_interfaces
https://github.com/ros2/common_interfaces

Custom ROS2 messages

```
cd ros2_ws/src
ros2 pkg create my_robot_interfaces
ls
cd my_robot_interfaces/
rm -rf include/
rm -rf src/
mkdir msg
cd msg

touch HardwareStatus.msg
```

Edit the files as

1. HardwareStatus.msg

```
int64 temperature
bool are_motors_ready
string debug_message
```

2. CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(my_robot_interfaces)

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(my_robot_interfaces
"msg/HardwareStatus.msg"
)

ament_package()
```

3. 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_robot_interfaces</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="asha@todo.todo">asha</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <build_depend>rosidl_default_generators</build_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

```
cd ~/ros2_ws

colcon build --packages-select my_robot_interfaces
cd install/my_robot_interfaces/lib/python3.8/site-packages/my_robot_interfaces/msg
 gedit _hardware_status.py
```

```
ros2 interface show my_robot_interfaces/msg/HardwareStatus
```

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch hw_status_publisher.py
chmod +x hw_status_publisher.py
colcon build --packages-select my_py_pkg --symlink-install
ros2 run my_py_pkg hw_status_publisher
```

edit hw_status_publisher.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interfaces.msg import HardwareStatus

class HardwareStatusPublisherNode(Node):
    def __init__(self):
        super().__init__("hardware_status_publisher") # MODIFY NAME
        self.hw_status_publisher_ = self.create_publisher(HardwareStatus,
"hardware_status", 10)
        self.timer_ = self.create_timer(1.0, self.publish_hw_status)
        self.get_logger().info("Hardware Publisher Started")

    def publish_hw_status(self):
        msg = HardwareStatus()
        msg.temperature = 45
        msg.are_motors_ready = True
        msg.debug_message = "Nothing"
        self.hw_status_publisher_.publish(msg)

def main(args=None):
        rclpy.init(args=args)
        node = HardwareStatusPublisherNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
        main()
```

CMakeLists.txt

```
entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main",
            "number_publisher = my_py_pkg.number_publisher:main",
            "number_counter = my_py_pkg.number_counter:main",
            "add_two_ints_server = my_py_pkg.add_two_ints_server:main",
            "add_two_ints_client_no_oop = my_py_pkg.add_two_ints_client_no_oop:main",
            "add_two_ints_client= my_py_pkg.add_two_ints_client:main",
            "hw_status_publisher = my_py_pkg.hw_status_publisher:main"
        ],
```

package.xml

```
 <depend>rclpy</depend>
  <depend>example_interfaces</depend>
  <depend>my_robot_interfaces</depend>
```

```
ros2 topic list
ros2 topic info /hardware_status
ros2 topic echo /hardware_status
ros2 node list
```

```
cd ros2_ws/src/my_cpp_pkg/src/
touch hw_status_publisher.cpp
```

```
#include "rclcpp/rclcpp.hpp"
#include "my_robot_interfaces/msg/hardware_status.hpp"

class HardwareStatusPublisher : public rclcpp::Node // MODIFY NAME
{
public:
    HardwareStatusPublisher() : Node("hardware_status_publisher") // MODIFY NAME
    {
        pub_ = this->create_publisher<my_robot_interfaces::msg::HardwareStatus>(
            "hardware_status", 10);
        timer_ = this->create_wall_timer(
            std::chrono::seconds(1),
            std::bind(&HardwareStatusPublisher::publishHardwareStatus, this));
        RCLCPP_INFO(this->get_logger(), "Hardware status publisher started");
    }

private:
    void publishHardwareStatus()
    {
        auto msg = my_robot_interfaces::msg::HardwareStatus();
        msg.temperature = 45;
        msg.are_motors_ready = false;
        msg.debug_message = "Motors Hot";
        pub_->publish(msg);
    }
    rclcpp::Publisher<my_robot_interfaces::msg::HardwareStatus>::SharedPtr pub_;
    rclcpp::TimerBase::SharedPtr timer_;
};


int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<HardwareStatusPublisher>(); // MODIFY NAME
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
 }
```

package.xml

```
  <depend>rclcpp</depend>
  <depend>example_interfaces</depend>
  <depend>my_robot_interfaces</depend>
```

CmakeList.txt

```
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(example_interfaces REQUIRED)
find_package(my_robot_interfaces REQUIRED)

add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

add_executable(robot_news_station src/robot_news_station.cpp)
ament_target_dependencies(robot_news_station rclcpp example_interfaces)


add_executable(smartphone src/smartphone.cpp)
ament_target_dependencies(smartphone rclcpp example_interfaces)


add_executable(add_two_ints_server src/add_two_ints_server.cpp)
ament_target_dependencies(add_two_ints_server rclcpp example_interfaces)

add_executable(add_two_ints_client src/add_two_ints_client.cpp)
```

```
ament_target_dependencies(add_two_ints_client rclcpp example_interfaces)

add_executable(add_two_int_client_no_oop src/add_two_int_client_no_oop.cpp)
ament_target_dependencies(add_two_int_client_no_oop rclcpp example_interfaces)

add_executable(hardware_status_publisher src/hw_status_publisher.cpp)
ament_target_dependencies(hardware_status_publisher rclcpp my_robot_interfaces)

install(TARGETS
  cpp_node
  robot_news_station
  smartphone
  add_two_ints_server
  add_two_int_client_no_oop
  add_two_ints_client
  hardware_status_publisher
  DESTINATION lib/my_cpp_pkg
)
```

.vscode->c_cpp_properties.json

```
 "includePath": [
                "${workspaceFolder}/**",
                "/opt/ros/foxy/include",
                "~/ros2_ws/install/my_robot_interfaces/include"
            ],
```

.vscode→settings.json

```
"python.autoComplete.extraPaths": [
        "/home/asha/ros2_ws/build/my_py_pkg",
        "/home/asha/ros2_ws/install/my_py_pkg/lib/python3.8/site-packages",
        "/home/asha/ros2_ws/install/first_package/lib/python3.8/site-packages",
        "/opt/ros/foxy/lib/python3.8/site-packages",
        "~/ros2_ws/install/my_robot_interfaces/lib/python3.8/site-packages/my_robot_interfaces"
    ],
```

```
colcon build --packages-select my_cpp_pkg my_robot_interfaces

ros2 run my_cpp_pkg hardware_status_publisher

ros2 topic list
ros2 topic echo /hardware_status


cd ros2_ws/src/my_robot_interfaces/
mkdir srv
cd srv
touch ComputeRectangleArea.srv
```

```
float64 length
float64 width
```

```
---
float64 area
```

CmakeList.txt

```
rosidl_generate_interfaces(my_robot_interfaces
"msg/HardwareStatus.msg"
"srv/ComputeRectangleArea.srv"
)
```

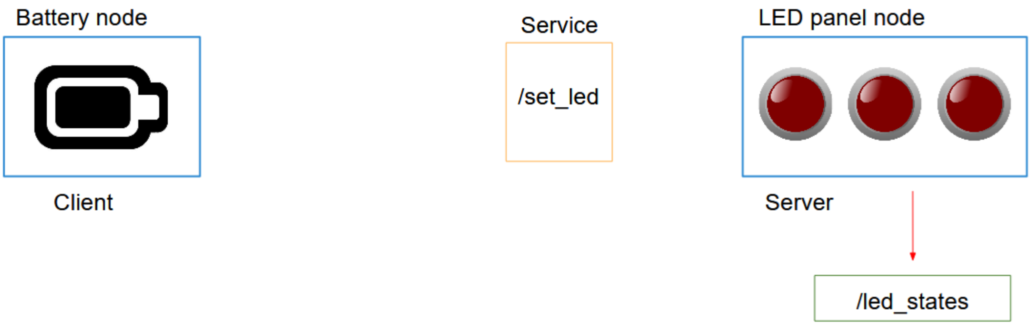
```
colcon build --packages-select my_robot_interfaces
```

```
ros2 interface show my_robot_interfaces/srv/ComputeRectangleArea
```

```
ros2 interface list
ros2 interface package sensor_msgs
ros2 interface show example_interfaces/msg/String
ros2 run my_py_pkg hw_status_publisher
```

```
ros2 node list
ros2 node info /hardware_status_publisher
ros2 topic list
ros2 topic info /hardware_status
ros2 interface show my_robot_interfaces/msg/HardwareStatus
ros2 service list
```

Exercise:



Implement the battery + led panel example that we used to understand Services in the previous section. When the battery is empty we power on a LED, and when the battery is full we power it off.

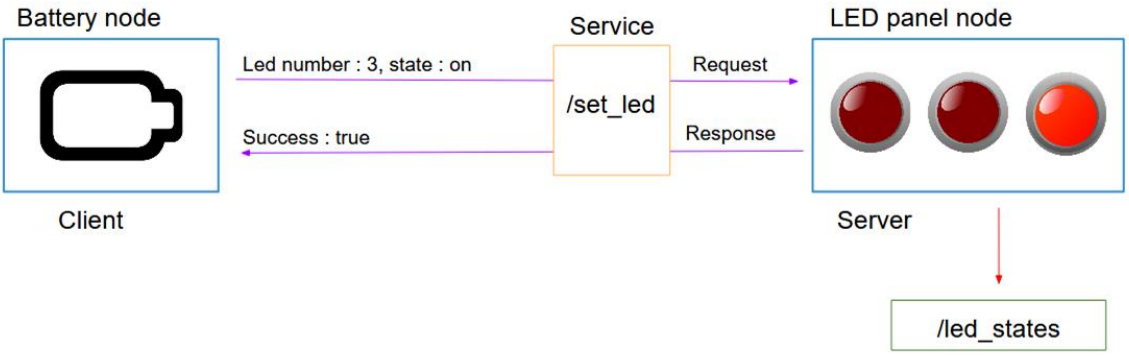
Blue boxes are for nodes, orange for services, and green for topics.

You can simply represent the state of the battery by a “battery_state” variable inside the battery node, and the LED panel by an integer array, inside the LED panel node.

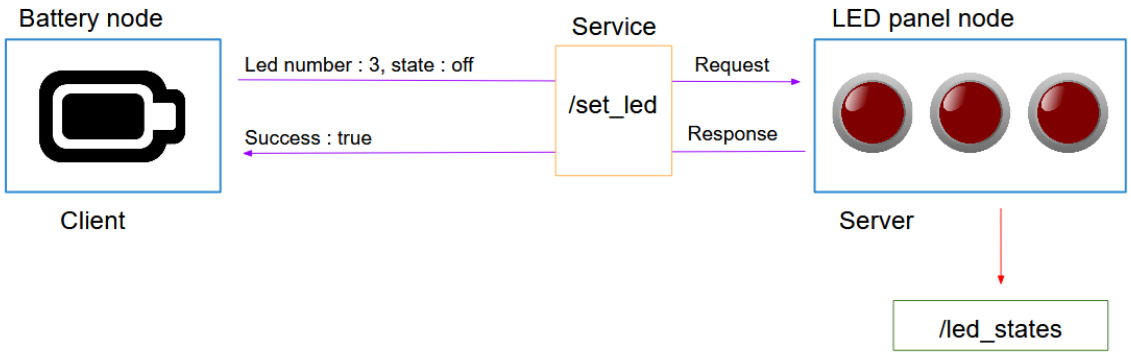
At first, the battery is full, and all the LEDs are powered off ([0, 0, 0]).

Now, you will fake the battery state evolution. Let’s say that the battery is empty after 4 seconds, and then after 6 more seconds it’s full again. And so on.

When the battery is empty, the battery node will send a request to the LED panel, to power on one LED.



And, 6 seconds later, when the battery is full again, it will send another request to power off the LED.



This will create inside our “src” directory a new package with some files in it

Hints: And you can continue looping between those 2 states indefinitely (until you press CTRL+C).
You will have to create:

- 1 node for the battery
- 1 node for the LED panel
- A custom msg definition for the “led_states” topic
- A custom srv definition for the “set_led” service

Solution:

```
cd ros2_ws/src/my_robot_interfaces/msg/
touch LedStateArray.msg
```

LedStateArray.msg

```
int64[] led_status
```

CmakeList.txt

```
rosidl_generate_interfaces(my_robot_interfaces
"msg/HardwareStatus.msg"
"msg/LedStateArray.msg"
"srv/ComputeRectangleArea.srv"
)
```

```
colcon build –packages-select my_robot_interfaces
```

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch led_panel.py
chmod +x led_panel.py
```

led_panel.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interfaces.msg import LedStateArray

class LedPanelNode(Node):
    def __init__(self):
        super().__init__("led_panel")
        self.led_status_ = [0,0,0]
        self.led_status_publisher_ = self.create_publisher(LedStateArray,"led_status",10)
        self.led_status_timer_ = self.create_timer(4, self.publish_led_status)
        self.get_logger().info("LED panel started")
    def  publish_led_status(self):
        msg = LedStateArray()
        msg.led_status = self.led_status_
        self.led_status_publisher_.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    node = LedPanelNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

setup.py

```
entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main",
            "number_publisher = my_py_pkg.number_publisher:main",
            "number_counter = my_py_pkg.number_counter:main",
            "add_two_ints_server = my_py_pkg.add_two_ints_server:main",
            "add_two_ints_client_no_oop = my_py_pkg.add_two_ints_client_no_oop:main",
            "add_two_ints_client= my_py_pkg.add_two_ints_client:main",
            "hw_status_publisher = my_py_pkg.hw_status_publisher:main",
            "led_panel = my_py_pkg.led_panel:main"
        ],
```

```
colcon build --packages-select my_py_pkg my_robot_interfaces --symlink-install
```

```
ros2 topic list
ros2 topic info /led_status
ros2 topic echo /led_status
```

```
cd ros2_ws/src/my_robot_interfaces/srv
touch SetLed.srv
```

SetLed.srv

```
int64 led_number
int64 state
---
bool success
```

CmakeList.txt

```
rosidl_generate_interfaces(my_robot_interfaces
"msg/HardwareStatus.msg"
"msg/LedStateArray.msg"
"srv/ComputeRectangleArea.srv"
"srv/SetLed.srv"
)
```

```
colcon build --packages-select my_robot_interfaces
```

led_panel.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interfaces.msg import LedStateArray
from my_robot_interfaces.srv import SetLed
class LedPanelNode(Node):
    def __init__(self):
        super().__init__("led_panel")
        self.led_status_ = [0,0,0]
        self.led_status_publisher_ = self.create_publisher(LedStateArray,"led_status",10)
        self.led_status_timer_ = self.create_timer(4, self.publish_led_status)
        self.set_led_Service_ = self.create_service(SetLed,"set_led", self. callback_set_led)
        self.get_logger().info("LED panel started")
    def  publish_led_status(self):
        msg = LedStateArray()
        msg.led_status = self.led_status_
        self.led_status_publisher_.publish(msg)

    def callback_set_led(self,request,response):
        led_number = request.led_number
        state = request.state


        if led_number > len(self.led_status_) or led_number<=0:
            response.success = False
            return response


        if state not in [0, 1]:
            response.success = False
            return response

        self.led_status_[led_number-1]= state
        response.success = True
        return response

def main(args=None):
    rclpy.init(args=args)
    node = LedPanelNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

```
colcon build --packages-select my_py_pkg --symlink-install
```

```
ros2 service list
ros2 service type /set_led
ros2 service call /set_led my_robot_interfaces/srv/SetLed "{led_number: 1, state: 1}"
```

```
ros2 run my_py_pkg led_panel
```

```
ros2 topic list
ros2 topic echo /led_status
```

```
cd ros2_ws/src/my_py_pkg/my_py_pkg/
touch battery.py
chmod +x battery.py
```

battery.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node


class BatteryNode(Node):
    def __init__(self):
        super().__init__("battery") # MODIFY NAME
        self.battery_state_ = "full"
        self.last_time_battery_state_changed_ = self.get_current_time_seconds()
        self.battery_timer_ = self.create_timer(0.1, self.check_battery_state)
        self.get_logger().info("Battery Started")

    def get_current_time_seconds(self):
        secs, nsecs = self.get_clock().now().seconds_nanoseconds()
        return secs + nsecs / 1000000000.0

    def check_battery_state(self):
        time_now = self.get_current_time_seconds()
        if self.battery_state_ == "full":
            if time_now - self.last_time_battery_state_changed_ > 4.0:
                self.battery_state_ = "empty"
                self.get_logger().info("Battery is charging")
                self.last_time_battery_state_changed_ = time_now
        else:
            if time_now - self.last_time_battery_state_changed_ > 6.0:
                self.battery_state_ = "full"
                self.get_logger().info("Battery is full")
                self.last_time_battery_state_changed_ = time_now


def main(args=None):
    rclpy.init(args=args)
    node = BatteryNode()
    rclpy.spin(node)
    rclpy.shutdown()



if __name__ == "__main__":
          main()
```

setup.py

```
 entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main",
            "robot_news_station = my_py_pkg.robot_news_station:main",
            "smartphone = my_py_pkg.smartphone:main",
            "number_publisher = my_py_pkg.number_publisher:main",
            "number_counter = my_py_pkg.number_counter:main",
            "add_two_ints_server = my_py_pkg.add_two_ints_server:main",
            "add_two_ints_client_no_oop = my_py_pkg.add_two_ints_client_no_oop:main",
            "add_two_ints_client= my_py_pkg.add_two_ints_client:main",
            "hw_status_publisher = my_py_pkg.hw_status_publisher:main",
            "led_panel = my_py_pkg.led_panel:main",
            "battery = my_py_pkg.battery:main"
        ],
```

```
colcon build --packages-select my_py_pkg
```

```
ros2 run my_py_pkg battery
```

battery.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from my_robot_interfaces.srv import SetLed
from functools import partial


class BatteryNode(Node):
    def __init__(self):
        super().__init__("battery") # MODIFY NAME
        self.battery_state_ = "full"
        self.last_time_battery_state_changed_ = self.get_current_time_seconds()
        self.battery_timer_ = self.create_timer(0.1, self.check_battery_state)
        self.get_logger().info("Battery Started")

    def get_current_time_seconds(self):
        secs, nsecs = self.get_clock().now().seconds_nanoseconds()
        return secs + nsecs / 1000000000.0

    def check_battery_state(self):
        time_now = self.get_current_time_seconds()
        if self.battery_state_ == "full":
            if time_now - self.last_time_battery_state_changed_ > 4.0:
                self.battery_state_ = "empty"
                self.get_logger().info("Battery is charging")
                self.last_time_battery_state_changed_ = time_now
                self.call_set_led_server(3, 1)
        else:
            if time_now - self.last_time_battery_state_changed_ > 6.0:
                self.battery_state_ = "full"
                self.get_logger().info("Battery is full")
                self.last_time_battery_state_changed_ = time_now
                self.call_set_led_server(3, 0)

    def call_set_led_server(self, led_number, state):
        client = self.create_client(SetLed, "set_led")
        while not client.wait_for_service(1.0):
            self.get_logger().warn("Waiting for Server")
        request = SetLed.Request()
        request.led_number = led_number
        request.state = state
        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_led_response,
led_number=led_number, state=state))

    def callback_led_response(self, future, led_number, state):
        try:
            response = future.result()
            self.get_logger().info(str(response.success))
        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = BatteryNode()
    rclpy.spin(node)
    rclpy.shutdown()



if __name__ == "__main__":
        main()
```

```
colcon build --packages-select my_py_pkg

ros2 run my_py_pkg led_panel

ros2 run my_py_pkg battery

ros2 topic echo /led_status
```

create a custom interface:

- Create a new package only for your msg and srv definitions.
- Setup the package (CMakeLists.txt and package.xml)
- Create a msg/ and srv/ folders, place your custom msg definitions and srv definitions here.

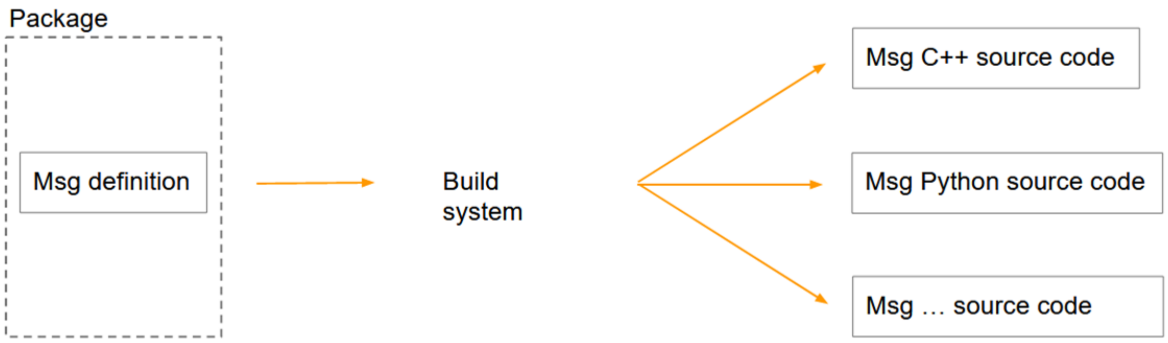
Once you’ve setup your package, adding a new interface is really simple:

- Add a new file in the right folder: msg/ or srv/
- Add one line into CMakeLists.txt
- Compile with “colcon build”
- And don’t forget to source your ROS2 workspace when you want to use those messages!

Here’s what you can use inside a msg or srv definition:

- Any primitive type defined by ROS2 (most common ones: int64, float64, bool, string, and array of those)
- Any message you’ve already created in this package.
- Any message from another package. In this case don’t forget to add a dependency for the other package in both package.xml and CMakeLists.txt.

ROS2 - Msg and Srv



```
ros2 pkg create <package_name> --build-type ament_cmake –dependencies
<package_dependencies>
```

The package_name is the name of the package you want to create, and the package_dependencies are the names of other ROS packages that your package depends on.

ROS2 parameters:

```
ros2 param list
```

```
ros2 param get /number_counter use_sim_time
```

```
ros2 run my_py_pkg number_counter
```

add

```
self.declare_parameter("test123")
self.declare_parameter("param")
```

in the NumberPublisherNode

```
colcon build --packages-select my_py_pkg
```

```
ros2 run my_py_pkg number_publisher --ros-args -p test123:=3 -p param:="hi"
ros2 param list
ros2 param get /number_publisher test123
ros2 param get /number_publisher param
```

Modify the number_publisher.py as

```
 #!/usr/bin/env python3
```

```
import rclpy
from rclpy.node import Node
from example_interfaces.msg import Int64

class NumberPublisherNode(Node): # MODIFY NAME
    def __init__(self):
        super().__init__("number_publisher") # MODIFY NAME
        #self.declare_parameter("test123")
        #self.declare_parameter("param")
        self.declare_parameter("publish_frequency", 1.0)
        self.frequency_=self.get_parameter("publish_frequency").value
        self.declare_parameter("number_to_publish",2)
        self.number_ = self.get_parameter("number_to_publish").value
        #self.number_ =2
        self.number_publisher_ = self.create_publisher(Int64, "number", 10)
        self.number_timer_ = self.create_timer(1.0/self.frequency_, self.publish_number)
        self.get_logger().info("Number Publisher has started!!")
    def publish_number(self):
        msg =  Int64()
        msg.data  = self.number_
        self.number_publisher_.publish(msg)


def main(args=None):
        rclpy.init(args=args)
        node = NumberPublisherNode() # MODIFY NAME
        rclpy.spin(node)
        rclpy.shutdown()

if __name__ == "__main__":
          main()
```

```
colcon build --packages-select my_py_pkg
```

```
ros2 run my_py_pkg number_publisher --ros-args -p number_to_publish:=4 -p publish_frequency:=2.0
```

```
ros2 topic echo /number
```

```
ros2 topic hz /number
```

Exercise

in C++

```
this→declare_parameter("number_to_publish",2);
this→declare_parameter("publish_frequency",1.0);

number_=this→get_parameter("number_to_publish").as_int();
double publish_frequency=this→get_parameter("publish_frequency"),as_double;
```

```
number_timer_=this→create_wall_timer(std::chrono::milliseconds((int)1000.0/
publish_frequency)
```

```
colcon build –packages-select my_cpp_pkg
ros2 run number_publisher
```

```
ros2 run number_publisher –ros-args -p publish_frequency:=6.0 -p number_to_publish:=7
ros2 param list
ros2 param get /number_publisher publish_frequency
```

```
ros2 topic hz number
```

remove number_

1.

Do you remember one of the first node we created in the Topic section, with the robot news radio? This node publishes a string on a topic, similar to this “Hello R2D2”.

Now, it would be better if we could set the robot’s name at run time, so we can launch the node multiple times with different robot names.

Add a “robot_name” parameter, and use the value to publish the string on the “robot_news” topic. Your string template (“Hi, this is <robot_name> from the Robot News Station!”) will now use the name you set at runtime.

```
self.declare_parameter(“robot_name”,”C3P0”)

self.robot_name_ = self.get_parameter( “ robot_name”).value
```

```
colcon build –packages-select my_py_pkg

ros2 run my_py_pkg robot_news_station  --ros-args -p  robot_name:=”R2D2”

ros2 param list

ros2 run my_py_pkg robot_news_station --ros-args -r __node:=news_station  -p
robot_name:=”R2D2”
```

```
ros2 param list
```

2.

Go back to the “led_panel_node”. Here you have an int array representing the states of your LEDs (0 for powered off, 1 for powered on). Set this array with a parameter named “led_states”.

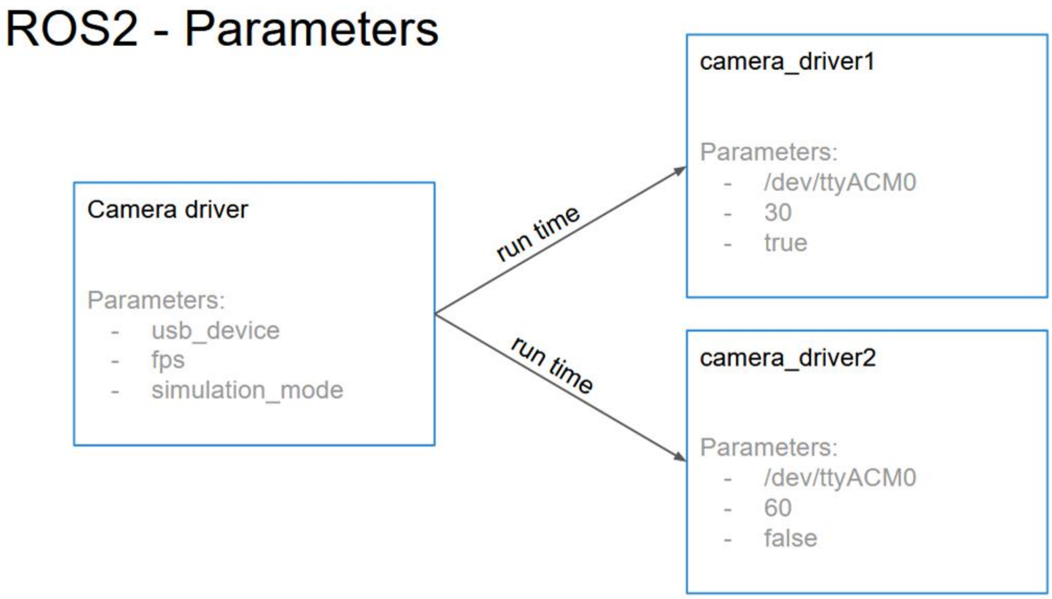
```
self.declare_parameter(“led_states”,[0,0,0])
self.led_states_=self.get_parameter(“led_states”).value
```

```
colcon build –packages-select my_py_pkg
```

```
ros2 run my_py_pkg led_panel –rosargs -p led_states:=[0,0,0,1,1]
```

C++

```
this→declare_parameter(“led_states”,std::vector<int64_t{0,0,0})’
led_states_=this→get_parameter(“led_states”).as_integer_array();
```



ROS2 - Parameters

Camera driver

Parameters:
- usb_device
- fps
- simulation_mode

camera_driver1

Parameters:
- /dev/ttyACM0
- 30
- true

camera_driver2

Parameters:
- /dev/ttyACM0
- 60
- false

To handle parameters:

- !!! Don’t forget to declare any parameter before you even try to use it !!!
- When you run your node, set values for your parameters.
- In your node’s code, get the parameters’ values and use them. You can also define default values (best practice to avoid errors at run-time).

Launch Files:

```
cd ros2_ws/src/
ros2 pkg create my_robot_bringup
cd my_robot_bringup/
rm -rf include/
rm -rf src/
mkdir launch
cd launch/
touch number_app.launch.py
```

```
chmod +x number_app.launch.py
```

number_app.launch.py

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

def generate_launch_description():
    ld = LaunchDescription()

    number_publisher_node = Node(
        package="my_py_pkg",
        executable="number_publisher"
    )


    number_counter_node = Node(
        package="my_py_pkg",
        executable="number_counter"
    )


    ld.add_action(number_publisher_node)
    ld.add_action(number_counter_node)

    return ld
```

CmakeList.txt

```
cmake_minimum_required(VERSION 3.5)
project(my_robot_bringup)

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
"Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)

install(DIRECTORY
  launch
  DESTINATION share/my_robot_bringup
  )
ament_package()
```

```
<?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_robot_bringup</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="asha@todo.todo">asha</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>


  <exec_depend>my_py_pkg</exec_depend>

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

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

```
cd ~/ros2_ws/
colcon build --packages-select my_robot_bringup --symlink-install

ros2 launch my_robot_bringup number_app.launch.py

ros2 node list

ros2 topic list

ros2 topic echo /number
```

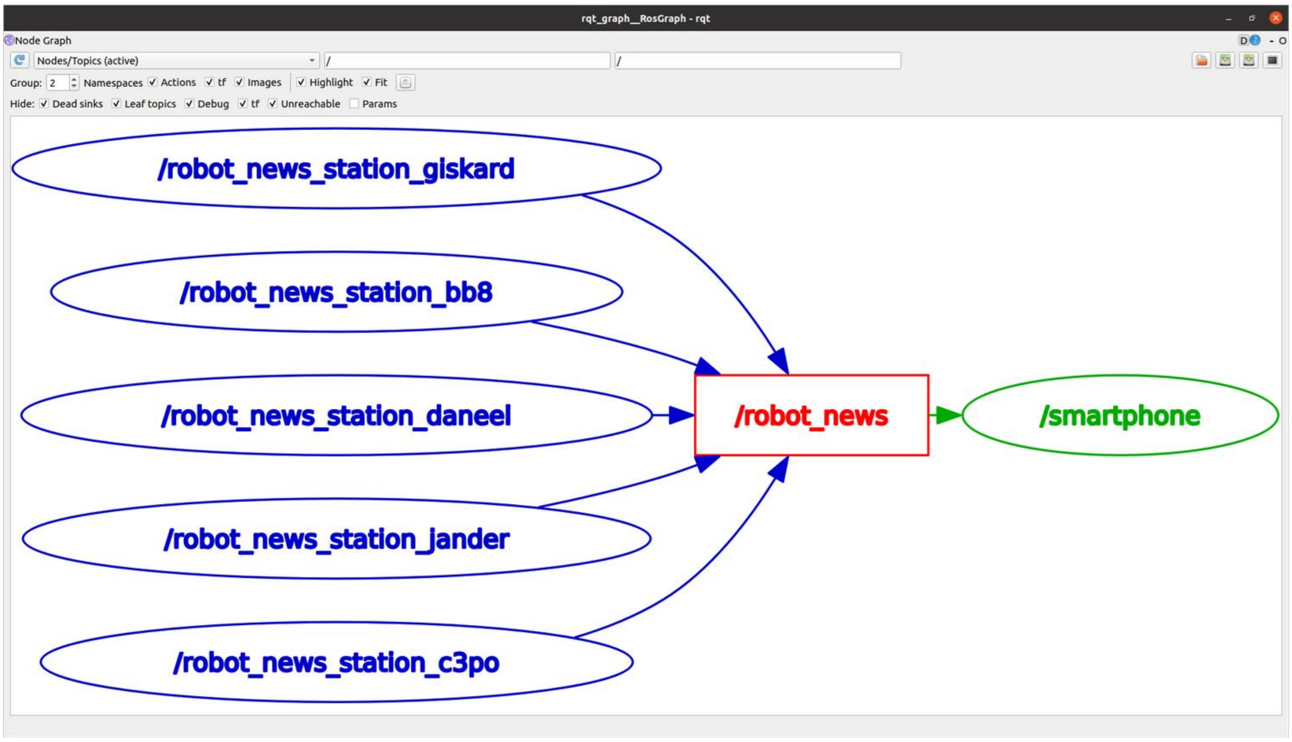
```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    ld = LaunchDescription()

    number_publisher_node = Node(
        package="my_py_pkg",
        executable="number_publisher",
        name="my_number_publisher"
    )


    number_counter_node = Node(
        package="my_py_pkg",
        executable="number_counter",
        name="my_number_counter"
    )



    ld.add_action(number_publisher_node)
    ld.add_action(number_counter_node)

    return ld


from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    ld = LaunchDescription()

    number_publisher_node = Node(
        package="my_py_pkg",
        executable="number_publisher",
        name="my_number_publisher",
        remappings = [
            ("number","my_number")
            ]
    )


    number_counter_node = Node(
        package="my_py_pkg",
        executable="number_counter",
        name="my_number_counter",
         remappings = [
            ("number","my_number"),
            ("number_count","my_number_count")
            ]
    )



    ld.add_action(number_publisher_node)
    ld.add_action(number_counter_node)

    return ld



from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    ld = LaunchDescription()
    remap_number_topic=("number","my_number")
    number_publisher_node = Node(
        package="my_py_pkg",
        executable="number_publisher",
        name="my_number_publisher",
        remappings = [
            remap_number_topic
            ]
    )


    number_counter_node = Node(
        package="my_py_pkg",
        executable="number_counter",
        name="my_number_counter",
         remappings = [
            remap_number_topic,
            ("number_count","my_number_count")
            ]
    )


    ld.add_action(number_publisher_node)
    ld.add_action(number_counter_node)

    return ld

from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    ld = LaunchDescription()

    remap_number_topic=("number","my_number")
    number_publisher_node = Node(
        package="my_py_pkg",
        executable="number_publisher",
        name="my_number_publisher",
        remappings = [
            remap_number_topic
            ],
        parameters=[
            {"number_to_publish": 4},
            {"publish_frequency": 5}
        ]

    )


    number_counter_node = Node(
        package="my_py_pkg",
        executable="number_counter",
        name="my_number_counter",
         remappings = [
            remap_number_topic,
            ("number_count","my_number_count")
            ]
    )


    ld.add_action(number_publisher_node)
    ld.add_action(number_counter_node)

    return ld
```

Exercise:

Goal:

- Start 5 “robot_news_station” nodes and 1 smartphone node.
- Each “robot_news_station” will need a different name, and will publish "Hi, this is <robot_name> from the Robot News Station!"
- The “smartphone” node gets all the messages from all other nodes.

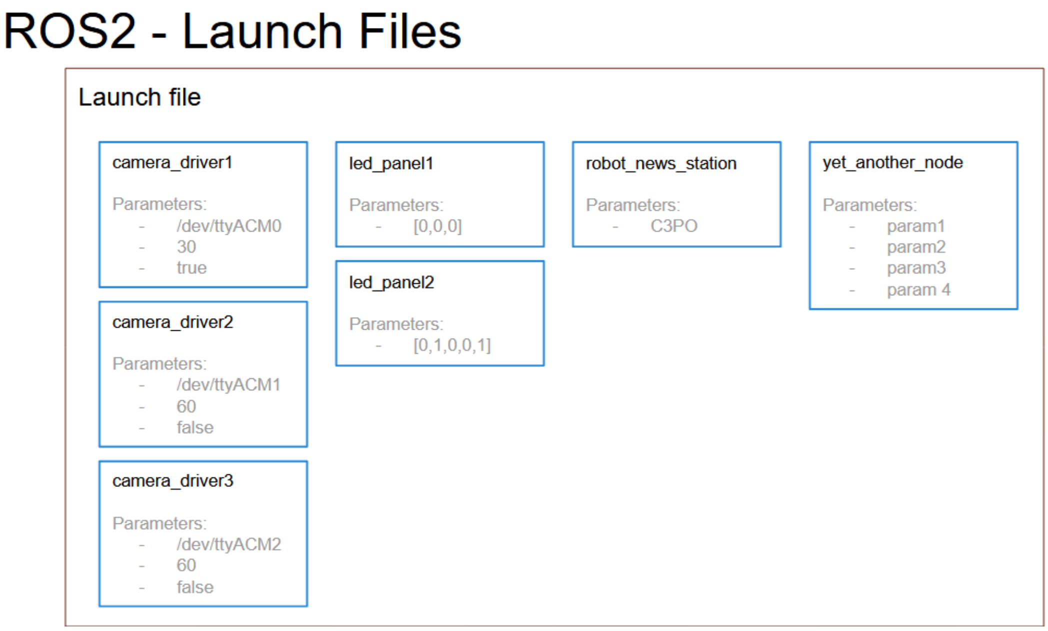
Here’s the graph you should get:


rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (active)
/
/
Group: 2 ☑ Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit
Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ tf ☑ Unreachable □ Params
/robot_news_station_giskard
/robot_news_station_bb8
/robot_news_station_daneel
/robot_news_station_jander
/robot_news_station_c3po
/robot_news
/smartphone

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


def generate_launch_description():
    ld=LaunchDescription()
    robot_names = ["giskard","bb8", "Daneil"]
    robot_news_station_nodes = []
    for name in robot_names:
        robot_news_station_nodes.append(Node(
            package="my_py_pkg",
            executable="robot_news_station",
            name="robot_news_station" + name.lower(),
            parameters=[{"robot_name": name}]

    ))

    smartphone = Node(
        package="my_py_pkg",
        executable="smartphone",
        name="smartphone"
    )

    for node in robot_news_station_nodes:
        ld.add_action(node)
    ld.add_action(smartphone)

    return ld
```

```
colcon build --packages-select my_robot_bringup --symlink-install
```

```
ros2 node list
```



ROS2 - Launch Files

Launch file

camera_driver1

Parameters:
- /dev/ttyACM0
- 30
- true

camera_driver2

Parameters:
- /dev/ttyACM1
- 60
- false

camera_driver3

Parameters:
- /dev/ttyACM2
- 60
- false

led_panel1

Parameters:
- [0,0,0]

led_panel2

Parameters:
- [0,1,0,0,1]

robot_news_station

Parameters:
- C3PO

yet_another_node

Parameters:
- param1
- param2
- param3
- param 4

Setup for launch files:

- Create a new package <robot_name>_bringup (best practice).
- Create a launch/ folder at the root of the package.
- Configure CMakeLists.txt to install files from this launch/ folder.
- Create any number of files you want inside the launch/ folder, ending with .launch.py.

Run a launch file:

- After you've written your file, use "colcon build" to install the file.
- Don't forget to source your environment
- Start the launch file with "ros2 launch <package> <name_of_the_file>

First try to design the application by yourself. Don't write code! Just take a piece of paper and make the design. What nodes should you create? How do the nodes communicate between each other? Which functionality should you add, and where to put them? Etc.

- Then, either you directly start on your own (let's call this the hardcore mode), or go to the next lecture where I give you some tips for the design.
- Then, work step by step on each functionality/communication.

You will use 3 nodes:

- The turtlesim_node from the turtlesim package
- A custom node to control the turtle (named "turtle1") which is already existing in the turtlesim_node. This node can be called turtle_controller.
- A custom node to spawn turtles on the window, and to manage which turtle is still "alive" (on the screen). This node can be called turtle_spawner.

You can create a new package (for example turtlesim_catch_them_all) to put your new nodes.

The turtle_spawner node will have to:

- Call the /spawn service to create a new turtle (choose random coordinates between 0.0 and 11.0 for both x and y), and call the /kill service to remove a turtle from the screen. Both those services are already advertised by the turtlesim_node.
- Publish the list of currently alive turtles with coordinates on a topic /alive_turtles.
- Handle a service server to "catch" a turtle, which means to call the /kill service and remove the turtle from the array of alive turtles.

The turtle_controller node will have to:

- Run a control loop (for example using a timer with a high rate) to reach a given target point. The first turtle on the screen “turtle1” will be the “master” turtle to control. To control the turtle you can subscribe to /turtle1/pose and publish to /turtle1/cmd_vel.
- The control loop will use a simplified P controller.
- Subscribe to the /alive_turtles topic to get all current turtles with coordinates. From that info, select a turtle to target (to catch).
- When a turtle has been caught by the master turtle, call the service /catch_turtle advertised by the turtle_spawner node.

You will need to create some custom interfaces:

- Turtle.msg and TurtleArray.msg to send the list of turtles (name + coordinates) on the /alive_turtles topic
- CatchTurtle.srv to send the name of the turtle which was caught. The client will be the turtle_controller node and the server will be the turtle_spawner node.
- → you can create messages in the my_robot_interfaces package.

Here’s the rqt_graph with the nodes and topics:

After you’ve created that, you will be able to scale the application with parameters and launch files. This will be the focus on the last part of the solution.

Here are the parameters you can have:

/turtle_controller:

catch_closest_turtle_first

use_sim_time

/turtle_spawner:

spawn_frequency

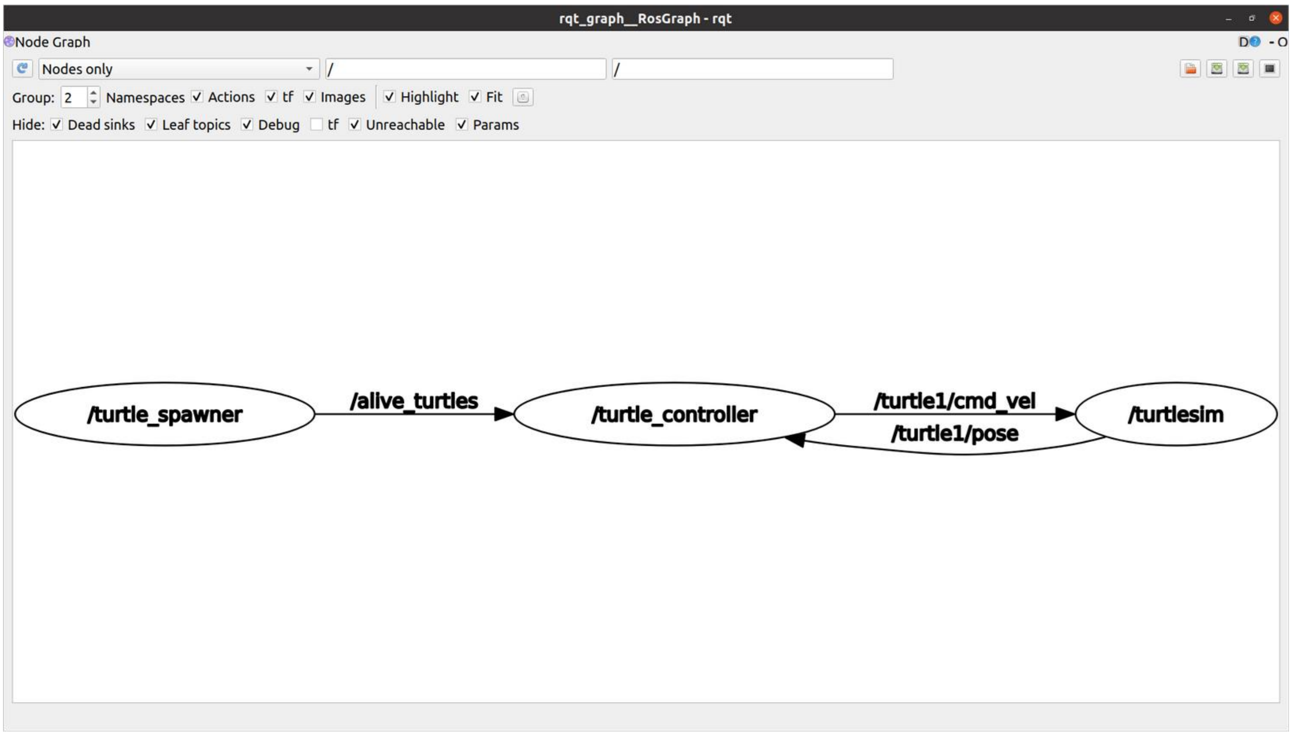
turtle_name_prefix

use_sim_time

For the launch file, you can create it inside the my_robot_bringup package. This will launch the 3 nodes along with parameters.

Steps for the solution videos:

- Step 1: Create the turtle_controller node, subscribe to /turtle1/pose. Create a control loop to reach a given target (for now an arbitrary one). A little bit of math will be required to find the distances and angles. And send the command to the /turtle1/cmd_vel topic.
- Step 2: Create the turtle_spawner node. With a timer, spawn a new turtle at a given rate. To spawn a turtle, call the /spawn service.
- Step 3: Keep an array of alive turtles (name + coordinates) in the turtle_spawner node. Publish this array on the /alive_turtles topic. On the turtle_controller node, subscribe to the topic, get the array, and choose to select the first turtle on the array as the new target.
- Step 4: Create a service /catch_turtle in turtle_spawner. Once the turtle_controller has reached a turtle, it will send the name of the turtle to that service. Then, from the turtle_spawner node, call the /kill service , remove the turtle from the array, and publish an updated array to /alive_turtles.
- Step 5: Improve the turtle_controller to select the closest turtle instead of the first turtle on the array.
- Step 6: Add parameters and create a launch file.



Nodes only / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

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

After you’ve created that, you will be able to scale the application with parameters and launch files. This will be the focus on the last part of the solution.

Here are the parameters you can have:

/turtle_controller:

catch_closest_turtle_first

use_sim_time

/turtle_spawner:

spawn_frequency

turtle_name_prefix

use_sim_time

For the launch file, you can create it inside the my_robot_bringup package. This will launch the 3 nodes along with parameters.

Steps for the solution videos:

- Step 1: Create the turtle_controller node, subscribe to /turtle1/pose. Create a control loop to reach a given target (for now an arbitrary one). A little bit of math will be required to find the distances and angles. And send the command to the /turtle1/cmd_vel topic.
- Step 2: Create the turtle_spawner node. With a timer, spawn a new turtle at a given rate. To spawn a turtle, call the /spawn service.
- Step 3: Keep an array of alive turtles (name + coordinates) in the turtle_spawner node. Publish this array on the /alive_turtles topic. On the turtle_controller node, subscribe to the topic, get the array, and choose to select the first turtle on the array as the new target.
- Step 4: Create a service /catch_turtle in turtle_spawner. Once the turtle_controller has reached a turtle, it will send the name of the turtle to that service. Then, from the turtle_spawner node, call the /kill service , remove the turtle from the array, and publish an updated array to /alive_turtles.
- Step 5: Improve the turtle_controller to select the closest turtle instead of the first turtle on the array.
- Step 6: Add parameters and create a launch file.

```
cd ros2_ws/src

ros2 pkg create turtlesim_catvh_them_all --build-type ament_python
```

```
cd turtlesim_catch_them_all/
touch turtle_controller.py
chmod +x turtle_controller.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.target_x = 8.0
        self.target_y = 4.0
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def callback_turtle_pose(self,msg):
        self.pose_ = msg



    def control_loop(self):
        if self.pose_ == None:
            return
        dist_x = self.target_x - self.pose_.x
        dist_y = self.target_y - self.pose_.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        msg = Twist()
        if distance > 0.5:
            msg.linear.x = 2*distance
            goal_theta = math.atan2(dist_y, dist_x)
            diff = goal_theta - self.pose_.theta

            if diff > math.pi:
                diff -= 2*math.pi
            elif diff < -math.pi:
                diff += 2*math.pi

            msg.angular.z = 6*diff
        else:
            msg.linear.x = 0.0
            msg.angular.z = 0.0

        self.cmd_vel_publisher_.publish(msg)
```

```
def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
        main()
```

setup.py

```
entry_points={
        'console_scripts': [
            "turtlesim_controller = turtlesim_catch_them_all.turtle_controller:main"
        ],
    },
```

package.xml

```
  <depend>rclpy</depend>
  <depend>turtlesim</depend>
```

```
ros2 run turtlesim turtlesim_node

colcon build --packages-select turtlesim_catch_them_all --symlink-install
ros2 run turtlesim_catch_them_all turtlesim_controller


ros2 run turtlesim turtlesim_node

ros2 service list
ros2 service type /spawn
ros2 interface show turtlesim/srv/Spawn

cd ros2_ws/src/turtlesim_catch_them_all/turtlesim_catch_them_all/
touch turtle_spawner.py
chmod +x turtle_spawner.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.srv import Spawn
from functools import partial
import random
import math

class TurtleSpawner(Node):
    def __init__(self):
        super().__init__("turtle_spawner") # MODIFY NAME
        self.turtle_name_prefix_ = "turtle"
        self.turtle_counter_ = 0
        self.spawn_turtle_timer_ = self.create_timer(2.0, self.spawn_new_turtle)

    def spawn_new_turtle(self):
        self.turtle_counter_ += 1
        name = self.turtle_name_prefix_ + str(self.turtle_counter_)
        x = random.uniform(0.0, 11.0)
        y = random.uniform(0.0, 11.0)
        theta = random.uniform(0.0, 2*math.pi)
        self.call_spawn_server(name,x,y,theta)

    def call_spawn_server(self, turtle_name, x, y, theta):
        client = self.create_client(Spawn, "spawn")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Spawn.Request()
        request.x = x
        request.y = y
        request.theta = theta
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_spawn, turtle_name=turtle_name,
x=x, y=y, theta=theta))

    def callback_call_spawn(self, future, turtle_name, x, y, theta):
        try:
            response = future.result()
            if response.name != "":
                self.get_logger().info("Turtle " + response.name + " is now alive")

        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleSpawner()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

setup.py

```
entry_points={
        'console_scripts': [
            "turtlesim_controller = turtlesim_catch_them_all.turtle_controller:main",
            "turtlesim_spawner = turtlesim_catch_them_all.turtle_spawner:main"
        ],
    },
```

```
colcon build --packages-select turtlesim_catch_them_all --symlink-install
ros2 run turtlesim_catch_them_all turtlesim_spawner
```

```
cd ros2_ws/src/my_robot_interfaces/msg/
touch Turtle.msg
string name
float64 x
float64 y
float64 theta
```

```
colcon build --packages-select my_robot_interfaces
touch TurtleArray.msg
Turtle[] turtles
```

turtle_spawner.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.srv import Spawn
from functools import partial
import random
import math
from my_robot_interfaces.msg import Turtle
from my_robot_interfaces.msg import TurtleArray

class TurtleSpawner(Node):
    def __init__(self):
        super().__init__("turtle_spawner") # MODIFY NAME
        self.turtle_name_prefix_ = "turtle"
        self.turtle_counter_ = 0
        self.alive_turtles_ = []
        self.alive_turtles_publisher_ = self.create_publisher(
            TurtleArray, "alive_turtles", 10)
        self.spawn_turtle_timer_ = self.create_timer(2.0, self.spawn_new_turtle)

    def publish_alive_turtles(self):
        msg = TurtleArray()
        msg.turtles = self.alive_turtles_
        self.alive_turtles_publisher_.publish(msg)

    def spawn_new_turtle(self):
        self.turtle_counter_ += 1
        name = self.turtle_name_prefix_ + str(self.turtle_counter_)
        x = random.uniform(0.0, 11.0)
        y = random.uniform(0.0, 11.0)
        theta = random.uniform(0.0, 2*math.pi)
        self.call_spawn_server(name, x, y, theta)

    def call_spawn_server(self, turtle_name, x, y, theta):
        client = self.create_client(Spawn, "spawn")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Spawn.Request()
        request.x = x
        request.y = y
        request.theta = theta
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_spawn, turtle_name=turtle_name,
x=x, y=y, theta=theta))

    def callback_call_spawn(self, future, turtle_name, x, y, theta):
        try:
            response = future.result()
            if response.name != "":
                self.get_logger().info("Turtle " + response.name + " is now alive")
                new_turtle = Turtle()
                new_turtle.name = response.name
                new_turtle.x = x
                new_turtle.y = y
                new_turtle.theta = theta
                self.alive_turtles_.append(new_turtle)
                self.publish_alive_turtles()
        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleSpawner()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

package.xml

```
  <depend>rclpy</depend>
  <depend>turtlesim</depend>
  <depend>geometry_msgs</depend>
  <depend>my_robot_interfaces</depend>
```

```
colcon build --packages-select turtlesim_catch_them_all --symlink-install

ros2 run turtlesim_catch_them_all turtlesim_spawner

ros2 run turtlesim turtlesim_node

ros2 topic echo /alive_turtles
```

turtle_controller.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
from my_robot_interfaces.msg import Turtle
from my_robot_interfaces.msg import TurtleArray


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.turtle_to_catch_ = None

        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.alive_turtles_subscriber_ = self.create_subscription(TurtleArray, "alive_turtles",
self.callback_alive_turtles, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def callback_turtle_pose(self,msg):
        self.pose_ = msg

    def callback_alive_turtles(self, msg):
        if len(msg.turtles)>0:
            self.turtle_to_catch_ = msg.turtles[0]

    def control_loop(self):
        if self.pose_ == None or self.turtle_to_catch_ == None:
            return
        dist_x = self.turtle_to_catch_.x - self.pose_.x
        dist_y = self.turtle_to_catch_.y - self.pose_.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        msg = Twist()
        if distance > 0.5:
            msg.linear.x = 2*distance
            goal_theta = math.atan2(dist_y, dist_x)
            diff = goal_theta - self.pose_.theta

            if diff > math.pi:
                diff -= 2*math.pi
            elif diff < -math.pi:
                diff += 2*math.pi

            msg.angular.z = 6*diff
        else:
            msg.linear.x = 0.0
            msg.angular.z = 0.0

        self.cmd_vel_publisher_.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
        main()
```

```
colcon build --packages-select turtlesim_catch_them_all --symlink-install
ros2 run turtlesim_catch_them_all turtlesim_controller

ros2 run turtlesim turtlesim_node

ros2 run turtlesim_catch_them_all turtlesim_spawner
```

```
cd ros2_ws/src/my_robot_interfaces/srv
touch CatchTurtle.srv
string name
---
bool success
```

```
colcon build –packages-select my_rorbot_interfaces –symlink -install

ros2 run turtlesim turtlesim_node
ros2 service type
ros2 service type /kill
ros2 interface show turtlesim/srv/Kill
```

turtlesim_controller.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
from functools import partial
from my_robot_interfaces.msg import Turtle
from my_robot_interfaces.msg import TurtleArray
from my_robot_interfaces.srv import CatchTurtle


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.turtle_to_catch_ = None

        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.alive_turtles_subscriber_ = self.create_subscription(TurtleArray, "alive_turtles",
self.callback_alive_turtles, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def callback_turtle_pose(self,msg):
        self.pose_ = msg

    def callback_alive_turtles(self, msg):
        if len(msg.turtles)>0:
            self.turtle_to_catch_ = msg.turtles[0]

    def control_loop(self):
        if self.pose_ == None or self.turtle_to_catch_ == None:
            return
        dist_x = self.turtle_to_catch_.x - self.pose_.x
        dist_y = self.turtle_to_catch_.y - self.pose_.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        msg = Twist()
        if distance > 0.5:
            msg.linear.x = 2*distance
            goal_theta = math.atan2(dist_y, dist_x)
            diff = goal_theta - self.pose_.theta

            if diff > math.pi:
                diff -= 2*math.pi
            elif diff < -math.pi:
                diff += 2*math.pi
            msg.angular.z = 6*diff
        else:
            msg.linear.x = 0.0
            msg.angular.z = 0.0
            self.call_catch_turtle_server(self.turtle_to_catch_.name)
            self.turtle_to_catch_ = None
        self.cmd_vel_publisher_.publish(msg)

    def call_catch_turtle_server(self, turtle_name):
        client = self.create_client(CatchTurtle, "catch_turtle")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = CatchTurtle.Request()
        request.name = turtle_name
        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_catch_turtle,
turtle_name=turtle_name))

    def callback_call_catch_turtle(self, future, turtle_name):
        try:
            response = future.result()
            if not response.success:
                self.get_logger().error("Turlte" + str(turtle_name) + " could not be caught")


        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

turtlesim_spawner.py

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.srv import Spawn
from functools import partial
import random
import math
from my_robot_interfaces.msg import Turtle
from turtlesim.srv import Kill
from my_robot_interfaces.msg import TurtleArray
from my_robot_interfaces.srv import CatchTurtle

class TurtleSpawner(Node):
    def __init__(self):
        super().__init__("turtle_spawner") # MODIFY NAME
        self.turtle_name_prefix_ = "turtle"
        self.turtle_counter_ = 0
        self.alive_turtles_ = []
        self.alive_turtles_publisher_ = self.create_publisher(
            TurtleArray, "alive_turtles", 10)
        self.spawn_turtle_timer_ = self.create_timer(2.0, self.spawn_new_turtle)
        self.catch_turtle_service_ = self.create_service(CatchTurtle, "catch_turtle",
self.callback_catch_turtle)

    def callback_catch_turtle(self, request, response):
        self.call_kill_server(request.name)
        response.success = True
        return response

    def publish_alive_turtles(self):
        msg = TurtleArray()
        msg.turtles = self.alive_turtles_
        self.alive_turtles_publisher_.publish(msg)

    def spawn_new_turtle(self):
        self.turtle_counter_ += 1
        name = self.turtle_name_prefix_ + str(self.turtle_counter_)
        x = random.uniform(0.0, 11.0)
        y = random.uniform(0.0, 11.0)
        theta = random.uniform(0.0, 2*math.pi)
        self.call_spawn_server(name, x, y, theta)

    def call_spawn_server(self, turtle_name, x, y, theta):
        client = self.create_client(Spawn, "spawn")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Spawn.Request()
        request.x = x
        request.y = y
        request.theta = theta
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_spawn, turtle_name=turtle_name,
x=x, y=y, theta=theta))

    def callback_call_spawn(self, future, turtle_name, x, y, theta):
        try:
            response = future.result()
            if response.name != "":
                self.get_logger().info("Turtle " + response.name + " is now alive")
                new_turtle = Turtle()
                new_turtle.name = response.name
                new_turtle.x = x
                new_turtle.y = y
                new_turtle.theta = theta
                self.alive_turtles_.append(new_turtle)
                self.publish_alive_turtles()
        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))


    def call_kill_server(self, turtle_name):
        client = self.create_client(Kill, "kill")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Kill.Request()
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_kill, turtle_name=turtle_name))

    def callback_call_kill(self, future, turtle_name):
        try:
            future.result()
            for (i,turtle) in enumerate(self.alive_turtles_):
                if turtle.name == turtle_name:
                    del self.alive_turtles_[i]
                    self.publish_alive_turtles()
                    break

        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleSpawner()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

```
colcon build --packages-select turtlesim_catch_them_all --symlink-install

ros2 run turtlesim turtlesim_node

ros2 run turtlesim_catch_them_all turtlesim_controller

 ros2 run turtlesim_catch_them_all turtlesim_spawner
```

edit turtlesim_controller.py to catch the closest turtle

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
from functools import partial
from my_robot_interfaces.msg import Turtle
from my_robot_interfaces.msg import TurtleArray
from my_robot_interfaces.srv import CatchTurtle


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.turtle_to_catch_ = None
        self.catch_closest_turtle_first_ = True
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.alive_turtles_subscriber_ = self.create_subscription(TurtleArray, "alive_turtles",
self.callback_alive_turtles, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)

    def callback_turtle_pose(self,msg):
        self.pose_ = msg

    def callback_alive_turtles(self, msg):
        if len(msg.turtles)>0:
            if self.catch_closest_turtle_first_:
                closest_turtle = None
                closest_turtle_distance = None

                for turtle in msg.turtles:
                    dist_x = turtle.x - self.pose_.x
                    dist_y = turtle.y - self.pose_.y
                    distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
                    if closest_turtle == None or distance < closest_turtle_distance:
                        closest_turtle = turtle
                        closest_turtle_distance = distance
                self.turtle_to_catch_ = closest_turtle
            else:
                self.turtle_to_catch_ = msg.turtles[0]

    def control_loop(self):
        if self.pose_ == None or self.turtle_to_catch_ == None:
            return
        dist_x = self.turtle_to_catch_.x - self.pose_.x
        dist_y = self.turtle_to_catch_.y - self.pose_.y
        distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
        msg = Twist()
        if distance > 0.5:
            msg.linear.x = 2*distance
            goal_theta = math.atan2(dist_y, dist_x)
            diff = goal_theta - self.pose_.theta

            if diff > math.pi:
                diff -= 2*math.pi
            elif diff < -math.pi:
                diff += 2*math.pi

            msg.angular.z = 6*diff
        else:
            msg.linear.x = 0.0
            msg.angular.z = 0.0
            self.call_catch_turtle_server(self.turtle_to_catch_.name)
            self.turtle_to_catch_ = None
        self.cmd_vel_publisher_.publish(msg)

    def call_catch_turtle_server(self, turtle_name):
        client = self.create_client(CatchTurtle, "catch_turtle")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = CatchTurtle.Request()
        request.name = turtle_name
        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_catch_turtle,
turtle_name=turtle_name))

    def callback_call_catch_turtle(self, future, turtle_name):
        try:
            response = future.result()
            if not response.success:
                self.get_logger().error("Turlte" + str(turtle_name) + " could not be caught")


        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

```
colcon build --packages-select turtlesim_catch_them_all --symlink-install

ros2 run turtlesim turtlesim_node

ros2 run turtlesim_catch_them_all turtlesim_controller

 ros2 run turtlesim_catch_them_all turtlesim_spawner
```

create launch file

```
cd ros2_ws/src/my_robot_bringup/launch
touch turtlesim_catch_them_all.launch.py
chmod +x turtlesim_catch_them_all.launch.py

colcon build --packages-select my_robot_bringup
```

package.xml of my_robot_bringup

```
  <exec_depend>my_py_pkg</exec_depend>
  <exec_depend>my_cpp_pkg</exec_depend>
  <exec_depend>turlesim</exec_depend>
  <exec_depend>tutlesim_catch_them_all</exec_depend>
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math
from functools import partial
from my_robot_interfaces.msg import Turtle
from my_robot_interfaces.msg import TurtleArray
from my_robot_interfaces.srv import CatchTurtle


class TurtleControllerNode(Node):
    def __init__(self):
        super().__init__("turtle_controller")
        self.declare_parameter("catch_closest_turtle_first_", True)
        self.turtle_to_catch_ = None
        self.catch_closest_turtle_first_ =
self.get_parameter("catch_closest_turtle_first_").value
        self.pose_ = None
        self.cmd_vel_publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)
        self.pose_subscriber_ = self.create_subscription(Pose, "turtle1/pose",
self.callback_turtle_pose, 10)
        self.alive_turtles_subscriber_ = self.create_subscription(TurtleArray, "alive_turtles",
self.callback_alive_turtles, 10)
        self.control_loop_timer_ = self.create_timer(0.01, self.control_loop)
```

```
def callback_turtle_pose(self,msg):
    self.pose_ = msg

def callback_alive_turtles(self, msg):
    if len(msg.turtles)>0:
        if self.catch_closest_turtle_first_:
            closest_turtle = None
            closest_turtle_distance = None

            for turtle in msg.turtles:
                dist_x = turtle.x - self.pose_.x
                dist_y = turtle.y - self.pose_.y
                distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
                if closest_turtle == None or distance < closest_turtle_distance:
                    closest_turtle = turtle
                    closest_turtle_distance = distance
            self.turtle_to_catch_ = closest_turtle
        else:
            self.turtle_to_catch_ = msg.turtles[0]

def control_loop(self):
    if self.pose_ == None or self.turtle_to_catch_ == None:
        return
    dist_x = self.turtle_to_catch_.x - self.pose_.x
    dist_y = self.turtle_to_catch_.y - self.pose_.y
    distance = math.sqrt(dist_x * dist_x + dist_y * dist_y)
    msg = Twist()
    if distance > 0.5:
        msg.linear.x = 2*distance
        goal_theta = math.atan2(dist_y, dist_x)
        diff = goal_theta - self.pose_.theta

        if diff > math.pi:
            diff -= 2*math.pi
        elif diff < -math.pi:
            diff += 2*math.pi

        msg.angular.z = 6*diff
    else:
        msg.linear.x = 0.0
        msg.angular.z = 0.0
        self.call_catch_turtle_server(self.turtle_to_catch_.name)
        self.turtle_to_catch_ = None
    self.cmd_vel_publisher_.publish(msg)

def call_catch_turtle_server(self, turtle_name):
    client = self.create_client(CatchTurtle, "catch_turtle")
    while not client.wait_for_service(1.0):
                    self.get_logger().warn("Waiting for Server")
    request = CatchTurtle.Request()
    request.name = turtle_name
    future = client.call_async(request)
```

```
        future.add_done_callback(partial(self.callback_call_catch_turtle,
turtle_name=turtle_name))

    def callback_call_catch_turtle(self, future, turtle_name):
        try:
            response = future.result()
            if not response.success:
                self.get_logger().error("Turlte" + str(turtle_name) + " could not be caught")


        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from turtlesim.srv import Spawn
from functools import partial
import random
import math
from my_robot_interfaces.msg import Turtle
from turtlesim.srv import Kill
from my_robot_interfaces.msg import TurtleArray
from my_robot_interfaces.srv import CatchTurtle

class TurtleSpawner(Node):
    def __init__(self):
        super().__init__("turtle_spawner") # MODIFY NAME
        self.declare_parameter("spawn_frequency", 1.0)
        self.declare_parameter("turtle_name_prefix_", "turtle")
        self.spawn_frequency_ = self.get_parameter("spawn_frequency").value
        self.turtle_name_prefix_ = self.get_parameter("turtle_name_prefix_").value
        self.turtle_counter_ = 0
        self.alive_turtles_ = []
        self.alive_turtles_publisher_ = self.create_publisher(
            TurtleArray, "alive_turtles", 10)
        self.spawn_turtle_timer_ = self.create_timer(1.0/self.spawn_frequency_ ,
self.spawn_new_turtle)
```

```
        self.catch_turtle_service_ = self.create_service(CatchTurtle, "catch_turtle",
self.callback_catch_turtle)

    def callback_catch_turtle(self, request, response):
        self.call_kill_server(request.name)
        response.success = True
        return response

    def publish_alive_turtles(self):
        msg = TurtleArray()
        msg.turtles = self.alive_turtles_
        self.alive_turtles_publisher_.publish(msg)

    def spawn_new_turtle(self):
        self.turtle_counter_ += 1
        name = self.turtle_name_prefix_ + str(self.turtle_counter_)
        x = random.uniform(0.0, 11.0)
        y = random.uniform(0.0, 11.0)
        theta = random.uniform(0.0, 2*math.pi)
        self.call_spawn_server(name, x, y, theta)

    def call_spawn_server(self, turtle_name, x, y, theta):
        client = self.create_client(Spawn, "spawn")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Spawn.Request()
        request.x = x
        request.y = y
        request.theta = theta
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_spawn, turtle_name=turtle_name,
x=x, y=y, theta=theta))

    def callback_call_spawn(self, future, turtle_name, x, y, theta):
        try:
            response = future.result()
            if response.name != "":
                self.get_logger().info("Turtle " + response.name + " is now alive")
                new_turtle = Turtle()
                new_turtle.name = response.name
                new_turtle.x = x
                new_turtle.y = y
                new_turtle.theta = theta
                self.alive_turtles_.append(new_turtle)
                self.publish_alive_turtles()
        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))


    def call_kill_server(self, turtle_name):
        client = self.create_client(Kill, "kill")
        while not client.wait_for_service(1.0):
                        self.get_logger().warn("Waiting for Server")
        request = Kill.Request()
        request.name = turtle_name

        future = client.call_async(request)
        future.add_done_callback(partial(self.callback_call_kill, turtle_name=turtle_name))

    def callback_call_kill(self, future, turtle_name):
        try:
            future.result()
            for (i,turtle) in enumerate(self.alive_turtles_):
                if turtle.name == turtle_name:
                    del self.alive_turtles_[i]
                    self.publish_alive_turtles()
                    break

        except Exception as e:
            self.get_logger().error("Service call failed %r" % (e,))

def main(args=None):
    rclpy.init(args=args)
    node = TurtleSpawner()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
          main()
```

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


def generate_launch_description():
    ld = LaunchDescription()


    turtlesim_node = Node(
        package = "turtlesim",
        executable = "turtlesim_node"

    )

    turtle_spawner_node = Node(
        package = "turtlesim_catch_them_all",
        executable = "turtlesim_spawner",
        parameters=[
            {"spawn_frequency": 0.5},
            {"turtle_name_prefix": "my_turtle"}
            ]
    )

    turtle_controller_node = Node(
        package = "turtlesim_catch_them_all",
        executable = "turtlesim_controller",
        parameters=[
            {"catch_closest_turtle_first_": True}
        ]
    )


    ld.add_action(turtlesim_node)
    ld.add_action(turtle_spawner_node)
    ld.add_action(turtle_controller_node)

    return ld
```

```
colcon build --packages-select my_robot_bringup

ros2 launch my_robot_bringup turtlesim_catch_them_all.launch.py


ros2 run my_py_pkg number_publisher



mkdir bags
cd bags
ros2 bag record /number
ros2 bag record /number -o test
ros2 bag info test/
ros2 bag play test

ros2 launch my_robot_bringup number_app.launch.py

ros2 bag record /my_number /my_number_count -o test2
ros2 bag info test2/
ros2 bag play test2
ros2 bag record -a -o test4
```

OpenCV + ROS2

```
cd ros2_ws/src/
ros2 pkg create --build-type ament_python cv_basics --dependencies rclpy
image_transport cv_bridge sensor_msgs std_msgs opencv2
touch webcam_pub.py
chmod +x webcam_pub.py
```

```
#!/usr/bin/env python3
```

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class ImagePublisher(Node):
  def __init__(self):
    super().__init__('image_publisher')
    self.publisher_ = self.create_publisher(Image, 'video_frames', 10)
    timer_period = 0.1  # seconds
    self.timer = self.create_timer(timer_period, self.timer_callback)
    self.cap = cv2.VideoCapture(0)
    self.br = CvBridge()

  def timer_callback(self):
    ret, frame = self.cap.read()
    if ret == True:
      self.publisher_.publish(self.br.cv2_to_imgmsg(frame))
    self.get_logger().info('Publishing video frame')

def main(args=None):

  rclpy.init(args=args)
  image_publisher = ImagePublisher()
  rclpy.spin(image_publisher)
  image_publisher.destroy_node()
  rclpy.shutdown()

if __name__ == '__main__':
  main()
```

```
touch webcam_sub.py
chmod +x webcam_sub.py
```

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class ImageSubscriber(Node):
  def __init__(self):
    super().__init__('image_subscriber')
    self.subscription = self.create_subscription(Image, 'video_frames', self.listener_callback,
10)
    self.subscription
    self.br = CvBridge()

  def listener_callback(self, data):
    self.get_logger().info('Receiving video frame')
    current_frame = self.br.imgmsg_to_cv2(data)
    cv2.imshow("camera", current_frame)
    cv2.waitKey(1)

def main(args=None):
  rclpy.init(args=args)
  image_subscriber = ImageSubscriber()
  rclpy.spin(image_subscriber)
  image_subscriber.destroy_node()
  rclpy.shutdown()

if __name__ == '__main__':
  main()
```

package.xml

```
entry_points={
        'console_scripts': [
            'img_publisher = cv_basics.webcam_pub:main',
            'img_subscriber = cv_basics.webcam_sub:main'
        ],
```

```
colcon build --packages-select cv_basics
```

```
ros2 run cv_basics img_publisher
```

```
ros2 run cv_basics img_subscriber
```

```
ros2 topic list -t
```

```
rqt_graph
```

TURTLESIM:

```
sudo apt update
sudo apt install ros-foxy-turtlesim
ros2 pkg executables turtlesim
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key

ros2 node list

ros2 topic list

ros2 service list

ros2 action list

sudo apt install ~nros-foxy-rqt*

ros2 run turtlesim turtlesim_node

ros2 run turtlesim turtle_teleop_key

rqt
```

Plugins > Services > Service Caller
Click the dropdown list in the middle of the window, and select the /spawn service.
launch a new turtle at coordinate x=1.0, y=1.0. The name of this new turtle will be **turtle2**.
Click Call to call the service.

Go to the Service dropdown list, and scroll down to /turtle1/set_pen.

Click Call to call the service.

```
ros2 run turtlesim turtle_teleop_key
```

to move turtle2 around the screen

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

BAG

```
ros2 run turtlesim turtlesim_node
```

```
ros2 run turtlesim turtle_teleop_key
```

```
mkdir bag_files
cd bag_files
```

```
ros2 topic list
rqt_graph
```

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

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

ctrl + C

```
dir
```

rosbag2_year_month_day-hour_minute_second

ros2 bag info <name_of_bag_file>

```
ros2 bag play <name_of_bag_file>
```

Camera

References:

https://github.com/ros-perception/image_common/tree/ros2
https://github.com/ros-drivers/usb_cam/tree/ros2

```
sudo apt install ros-foxy-image-common
cd /opt/ros/foxy
ls
cd share
ls
ls | grep image
cd ~
cd ros2_ws/
cd src
git clone https://github.com/ros-drivers/usb_cam.git
cd usb_cam/
git switch ros2
cd ..
colcon build --packages-select usb_cam
source install/setup.bash
ros2 launch ./src/usb_cam/launch/demo_launch.py
```

```
sudo apt install ros-foxy-image-transport-plugins
```

Execute in terminal 1

```
ros2 pkg create --build-type ament_python first_package --dependencies rclpy
ros2 run <package_name> <executable_file>
```

n order to check that our package has been created successfully, we can use some ROS commands related to packages.

In the terminal:

```
ros2 pkg list
ros2 pkg list | grep first_package
```

ros2 pkg list: Gives you a list with all of the packages in your ROS system.
ros2 pkg list | grep first_package: Filters, from all of the packages located in the ROS system, the package named first_package.

Compile package:

When you create a package, you will need to compile it in order to make it work.

```
colcon build –symlink-install
```

This command will compile your whole src directory, and it needs to be issued in your ros2 directory in order to work. This is MANDATORY.

```
colcon build --symlink-install --packages-select <package_name>
```

This command will only compile the packages specified and their dependencies.

```
colcon build --symlink-install --packages-select first_package
```

When compilation ends, you will need to source your workspace. You can do that with the following command

```
source ~/ros2/install/setup.bash
```

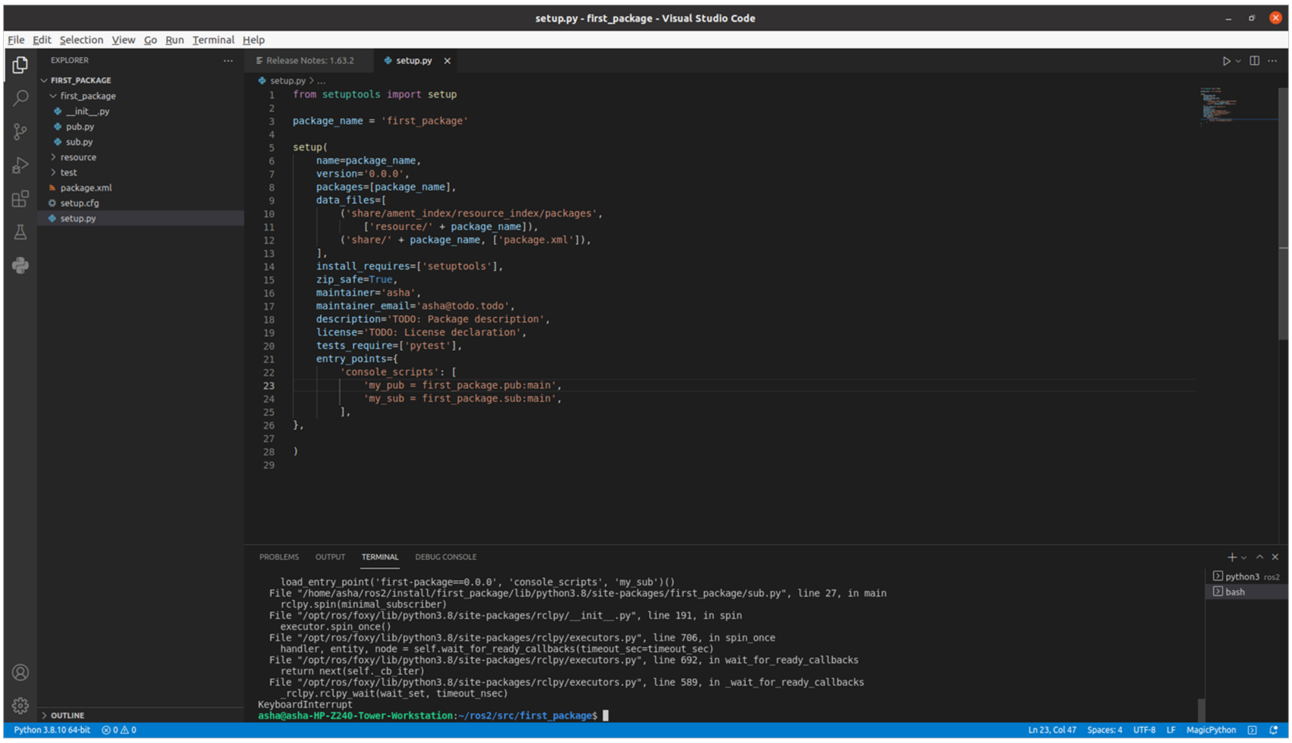
pub.py

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1


def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

sub.py

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning
    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

```
from setuptools import setup
package_name = 'first_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_pub = first_package.pub:main',
            'my_sub = first_package.sub:main',
        ],
},

)
```



```python
from setuptools import setup

package_name = 'first_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_pub = first_package.pub:main',
            'my_sub = first_package.sub:main',
        ],
},

)

```

```
    load_entry_point('first-package==0.0.0', 'console_scripts', 'my_sub')()
  File "/home/asha/ros2/install/first_package/lib/python3.8/site-packages/first_package/sub.py", line 27, in main
    rclpy.spin(minimal_subscriber)
  File "/opt/ros/foxy/lib/python3.8/site-packages/rclpy/__init__.py", line 191, in spin
    executor.spin_once()
  File "/opt/ros/foxy/lib/python3.8/site-packages/rclpy/executors.py", line 706, in spin_once
    handler, entity, node = self.wait_for_ready_callbacks(timeout_sec=timeout_sec)
  File "/opt/ros/foxy/lib/python3.8/site-packages/rclpy/executors.py", line 692, in wait_for_ready_callbacks
    return next(self._cb_iter)
  File "/opt/ros/foxy/lib/python3.8/site-packages/rclpy/executors.py", line 589, in _wait_for_ready_callbacks
    _rclpy.rclpy_wait(wait_set, timeout_nsec)
KeyboardInterrupt
asha@asha-HP-Z240-Tower-Workstation:~/ros2/src/first_package$ 
```

1. cd ~
2. cd ros2
3. source ~/ros2/install/setup.bash (run this while opening new terminal)
4. colcon build
5. ros2 run first_package my_pub
6. ros2 run first_package my_sub

ROS nodes are basically programs made in ROS. The ROS command to see what nodes are actually running in a computer is:
ros2 node list

turtle.py

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist


class Turtle(Node):

    def __init__(self):
        super().__init__('turtlesim_node')
        self.publisher_ = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        msg = Twist()
        msg.linear.x=1.0
        msg.linear.y=0.0
        self.publisher_.publish(msg)
```

```
def main(args=None):
    rclpy.init(args=args)
    minimal_publisher = Turtle()
    rclpy.spin(minimal_publisher)
    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

```
from setuptools import setup

package_name = 'first_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_pub = first_package.pub:main',
            'my_sub = first_package.sub:main',
            'my_turtle = first_package.turtle:main',
        ],
},

)
```

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

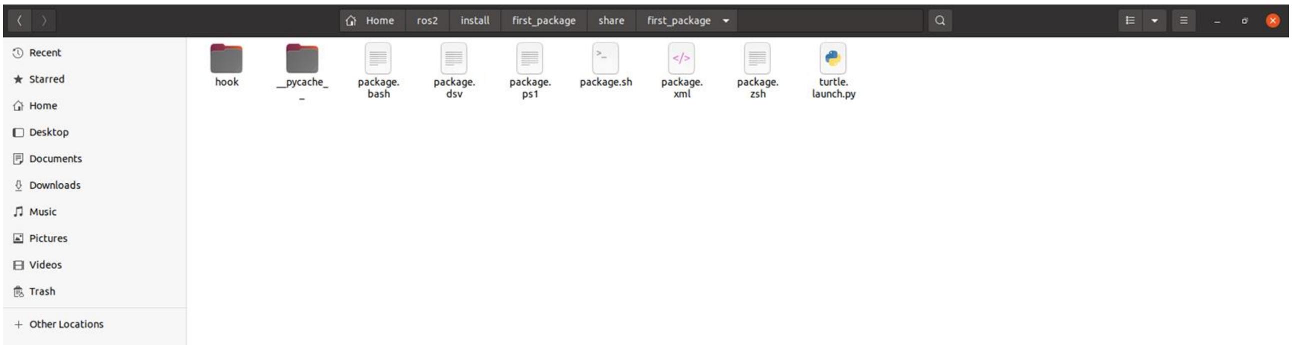
```
ros2 topic list
ros2 node list
ros2 topic info turtle1/pose
rqt_gragh


ros2 topic pub -l /topic message_type value

ros2 topic pub -1 turtle1/cmd_vel geometry_msgs/msg/Twist “{linear:{x=-4.0,y=0.0}}”

ros2 topic pub -1 turtle1/cmd_vel geometry_msgs.msg.Twist (linear=
geometry_msgs.msg.Vector3(x=3.0,y=1.0,z=0.0),angular=
geometry_msgs.msg.Vector3( x=0.0,y=1.0,z=0.0))
```

gotogoal.py

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
from math import pow, atan2, sqrt

def get_pose(data):
    global bot_pose
    bot_pose=data
    bot_pose.x=data.x
    bot_pose.y=data.y


def go_to_goal():
    global bot_pose, pub, goal
    dist=sqrt(pow((goal_x-bot_pose.x),2)+pow((goal_y-bot_pose.y),2))
    angle=atan2(goal_y-bot_pose.y, goal_x-bot_pose.x)
    turn_angle=angle-bot_pose.theta
    new_vel=Twist()
    new_vel.linear.x=dist
    new_vel.angular.z=turn_angle
    if (dist>=0.5):
        pub.publish(new_vel)


def main(args=None):
    rclpy.init(args=args)
    global node, pub, goal_x, goal_y
    goal_x=8.0
    goal_y=8.0

    node=Node('go_to_goal')
    node.create_subscription(Pose,'/turtle1/pose', get_pose, 10)
    pub=node.create_publisher(Twist,'/turtle1/cmd_vel',10)
    node.create_timer(1,go_to_goal)
    rclpy.spin(node)
    rclpy.shutdown()



if __name__ == '__main__':
    main()
```

create launch folder
turtle.launch.py

```
"""Launch a talker and a listener."""
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([

        Node(
        package='turtlesim',
        executable='turtlesim_node',
        name='node1'
        ),

        Node(
        package='first_package',
        executable='gotogoal',
        name='node2',
        ),
])

from setuptools import setup
import os
import glob

package_name = 'first_package'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name), glob('launch/*.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='asha',
    maintainer_email='asha@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_pub = first_package.pub:main',
            'my_sub = first_package.sub:main',
            'my_turtle = first_package.turtle:main',
            'gotogoal = first_package.go_to_goal:main',

        ],
},

)
```



```
ros2 pkg create --build-type ament_cmake dolly --dependencies rclpy

cd ..
colcon build
```

create 2 folders inside dolly
launch
urdf
create file
dolly.urdf

```
<?xml version="1.0" ?>
  <robot name="dolly">



  <link name="base">
  <visual>
    <geometry>
        <box size="0.75 0.4 0.1"/>
    </geometry>
    <material name="gray">
      <color rgba=".2 .2 .2 1" />
    </material>
  </visual>

  <inertial>
      <mass value="1" />
      <inertia ixx="0.01" ixy="0.0" ixz="0" iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <collision>
    <geometry>
        <box size="0.75 0.4 0.1"/>
    </geometry>
  </collision>

</link>





<link name="wheel_right_link">
  <inertial>
          <mass value="2" />
          <inertia ixx="0.01" ixy="0.0" ixz="0"
          iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <visual>
    <geometry>
        <cylinder radius="0.15" length="0.1"/>
    </geometry>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>

  <collision>

    <geometry>
        <cylinder radius="0.15" length="0.1"/>
    </geometry>
    <contact_coefficients mu="1" kp="1e+13" kd="1.0"/>
  </collision>
</link>
```

```
<joint name="wheel_right_joint" type="continuous">
  <origin xyz="0.2 0.25 0.0" rpy="1.57 0.0 0.0"/>
  <parent link="base"/>
  <child link="wheel_right_link"/>
  <axis xyz="0.0 0.0 1.0"/>
</joint>



<link name="wheel_left_link">
  <inertial>
          <mass value="2" />
          <inertia ixx="0.01" ixy="0.0" ixz="0"
          iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <visual>

    <geometry>
        <cylinder radius="0.15" length="0.1"/>
    </geometry>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>

  <collision>
    <geometry>
        <cylinder radius="0.15" length="0.1"/>
    </geometry>
    <contact_coefficients mu="1" kp="1e+13" kd="1.0"/>
  </collision>
</link>

<joint name="wheel_left_joint" type="continuous">
  <origin xyz="0.2 -0.25 0.0" rpy="1.57 0.0 0.0"/>
  <parent link="base"/>
  <child link="wheel_left_link"/>
  <axis xyz="0.0 0.0 1.0"/>

</joint>



<link name="caster">
  <inertial>
          <mass value="1" />
          <inertia ixx="0.01" ixy="0.0" ixz="0"
          iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <visual>
```

```
    <geometry>
      <sphere radius=".08" />
    </geometry>
    <material name="white" />
  </visual>

  <collision>
    <origin/>
    <geometry>
      <sphere radius=".08" />
    </geometry>
  </collision>
</link>

<joint name="caster_joint" type="continuous">
  <origin xyz="-0.3 0.0 -0.07" rpy="0.0 0.0 0.0"/>
  <axis xyz="0 0 1" />
  <parent link="base"/>
  <child link="caster"/>
</joint>

<link name="camera">
  <inertial>
          <mass value="0.1" />
          <inertia ixx="0.01" ixy="0.0" ixz="0"
          iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <visual>

    <geometry>
      <box size="0.1 0.1 0.05"/>
    </geometry>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>

  <collision>
    <geometry>
        <box size="0.1 0.1 0.05"/>
    </geometry>
  </collision>
</link>

<joint name="camera_joint" type="fixed">
  <origin xyz="-0.35 0 0.01" rpy="0 0.0 3.14"/>
  <parent link="base"/>
  <child link="camera"/>
  <axis xyz="0.0 0.0 1.0"/>
```

```
</joint>

<link name="lidar">
  <inertial>
          <mass value="0.5" />
          <inertia ixx="0.01" ixy="0.0" ixz="0"
          iyy="0.01" iyz="0" izz="0.01" />
  </inertial>

  <visual>

    <geometry>
      <cylinder radius="0.1" length="0.05"/>
    </geometry>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>

  <collision>
    <geometry>
        <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>
</link>

<joint name="lidar_joint" type="fixed">
  <origin xyz="-0.285 0 0.075" rpy="0 0.0 1.57"/>
  <parent link="base"/>
  <child link="lidar"/>
  <axis xyz="0.0 0.0 1.0"/>

</joint>


<gazebo reference="base">
  <material>Gazebo/Black</material>
</gazebo>
<gazebo reference="wheel_left_link">
  <material>Gazebo/Blue</material>
</gazebo>
<gazebo reference="wheel_right_link">
  <material>Gazebo/Blue</material>
</gazebo>
<gazebo reference="caster">
  <material>Gazebo/Grey</material>
</gazebo>
<gazebo reference="lidar">
  <material>Gazebo/Orange</material>
</gazebo>
<gazebo reference="camera">
  <material>Gazebo/Red</material>
  </gazebo>



  <!-- DIFFENERNTIAL DRIVEEEEEEEEEEEEEE -->

  <gazebo>
    <plugin filename="libgazebo_ros_diff_drive.so" name="gazebo_base_controller">
      <odometry_frame>odom</odometry_frame>
      <commandTopic>cmd_vel</commandTopic>
      <publish_odom>true</publish_odom>
      <publish_odom_tf>true</publish_odom_tf>
      <update_rate>15.0</update_rate>

      <left_joint>wheel_left_joint</left_joint>
      <right_joint>wheel_right_joint</right_joint>

      <wheel_separation>0.5</wheel_separation>
      <wheel_diameter>0.3</wheel_diameter>
      <max_wheel_acceleration>0.7</max_wheel_acceleration>
      <max_wheel_torque>8</max_wheel_torque>
      <robotBaseFrame>base</robotBaseFrame>
    </plugin>
  </gazebo>

<!-- CAMERAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA -->
<gazebo reference="camera">
  <sensor type="camera" name="camera1">
    <visualize>true</visualize>
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>60.0</updateRate>
      <cameraName>/camera</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>info_camera</cameraInfoTopicName>
      <frameName>camera</frameName>
      <hackBaseline>0.07</hackBaseline>
    </plugin>
  </sensor>
```

```
</gazebo>



<!-- LIDAAAAAAAAAAAAAAAAAAAAAAAAAAR -->
<gazebo reference="lidar">
  <sensor name="gazebo_lidar" type="ray">
   <visualize>true</visualize>
    <update_rate>12.0</update_rate>
    <plugin filename="libgazebo_ros_ray_sensor.so" name="gazebo_lidar">
    <output_type>sensor_msgs/LaserScan</output_type>
    <frame_name>lidar</frame_name>
    </plugin>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples>
          <resolution>1</resolution>
          <min_angle>0.00</min_angle>
          <max_angle>3.14</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.120</min>
        <max>3.5</max>
        <resolution>0.015</resolution>
      </range>

    </ray>
  </sensor>
</gazebo>


</robot>
```

rviz.launch.py

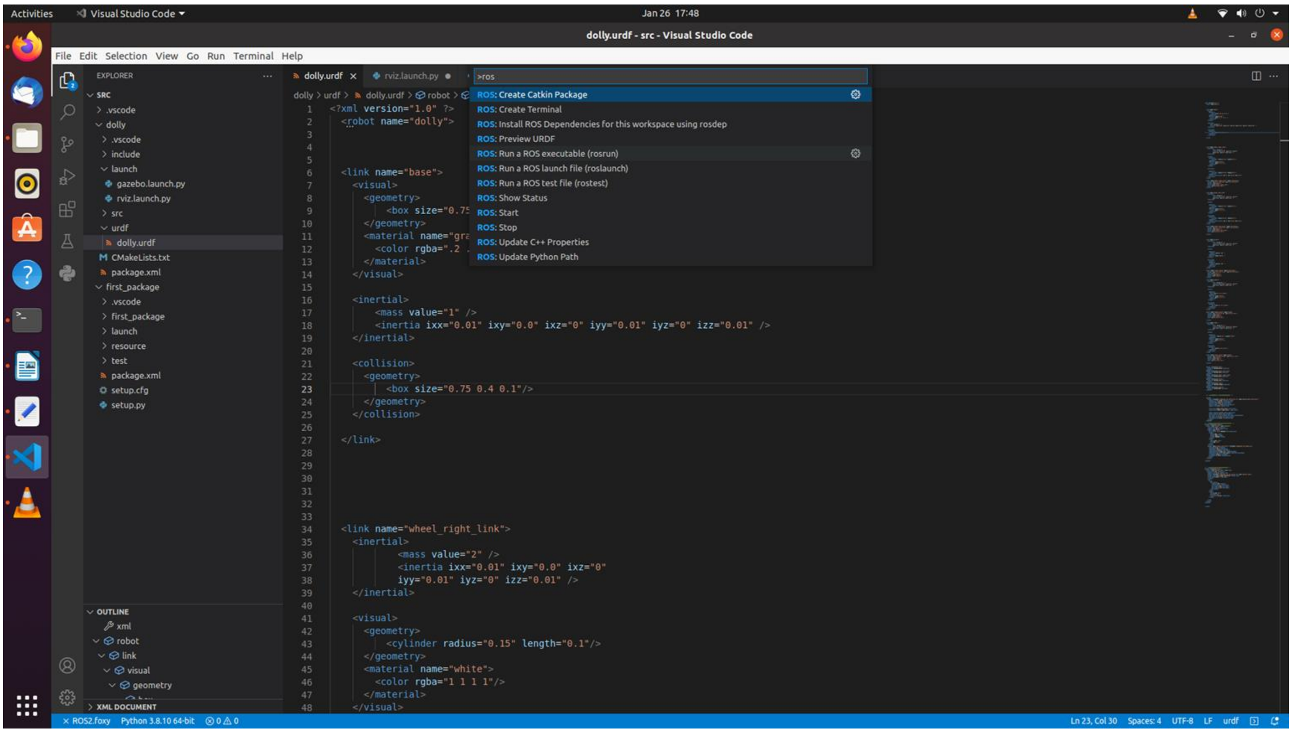
```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    urdf = '/home/luqman/beginners_ws/src/dolly/urdf/dolly.urdf'
    # rviz_config_file=os.path.join(package_dir,'config.rviz')

    return LaunchDescription([
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='robot_state_publisher',
            output='screen',
            arguments=[urdf]),
        Node(
            package='joint_state_publisher_gui',
            executable='joint_state_publisher_gui',
            name='joint_state_publisher_gui',
            arguments=[urdf]),

        Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        # arguments=['-d',rviz_config_file],
        output='screen'),


    ])
```

gazebo.launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import  ExecuteProcess
def generate_launch_description():
    urdf = '/home/luqman/beginners_ws/src/dolly/urdf/dolly.urdf'

    return LaunchDescription([

        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='robot_state_publisher',
            output='screen',
            arguments=[urdf]),
        Node(
            package='joint_state_publisher',
            executable='joint_state_publisher',
            name='joint_state_publisher',
            arguments=[urdf]),
#  Gazebo related stuff required to launch the robot in simulation
        ExecuteProcess(
            cmd=['gazebo', '--verbose', '-s', 'libgazebo_ros_factory.so'],
            output='screen'),
        Node(
            package='gazebo_ros',
            executable='spawn_entity.py',
            name='urdf_spawner',
            output='screen',
            arguments=["-topic", "/robot_description", "-entity", "dolly"])
  ])
```

```python
from setuptools import setup
import os
from glob import glob
package_name = 'dolly'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name), glob('urdf/*')),
        (os.path.join('share', package_name), glob('launch/*')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='luqman',
    maintainer_email='noshluk2@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'my_node = dolly.my_node:main'
        ],
    },
)
```

install extension
xml
ros
xml complete
ctrl+shift+p

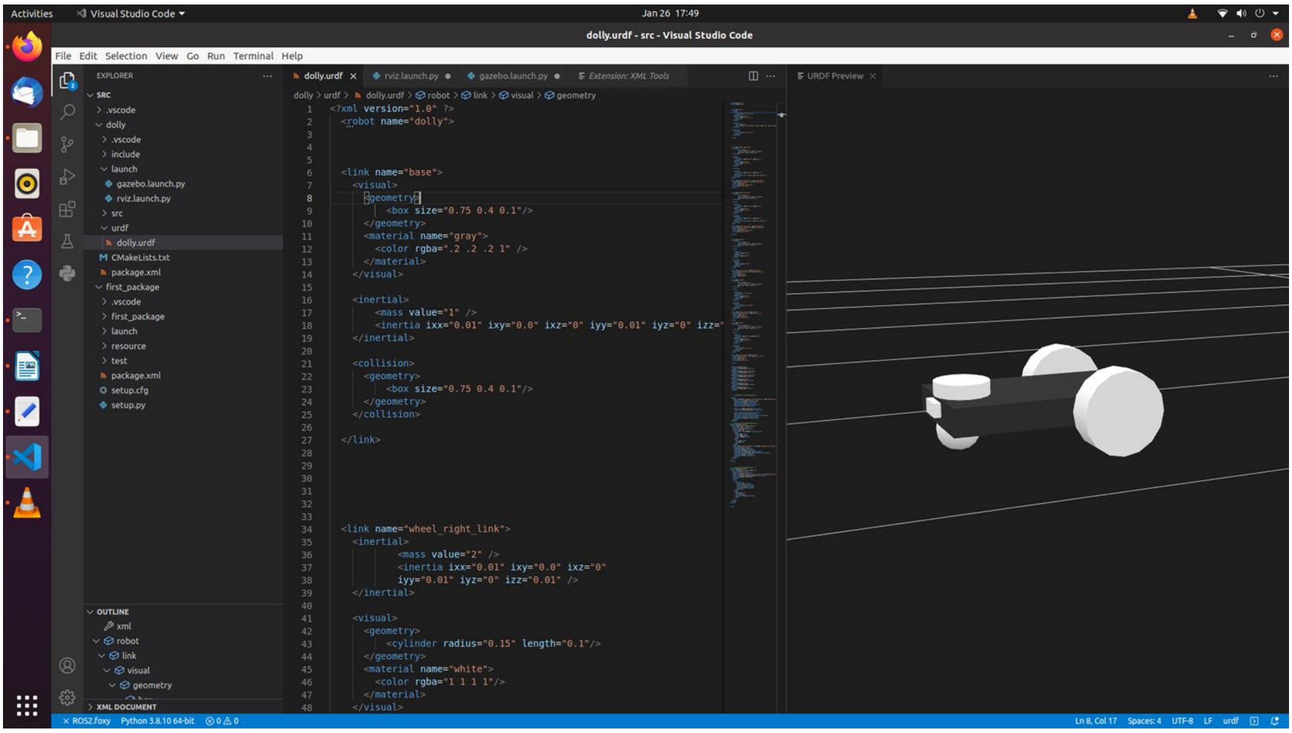


dolly.urdf - src - Visual Studio Code

```
>ros
```

- ROS: Create Catkin Package
- ROS: Create Terminal
- ROS: Install ROS Dependencies for this workspace using rosdep
- ROS: Preview URDF
- ROS: Run a ROS executable (rosrun)
- ROS: Run a ROS launch file (roslaunch)
- ROS: Run a ROS test file (rostest)
- ROS: Show Status
- ROS: Start
- ROS: Stop
- ROS: Update C++ Properties
- ROS: Update Python Path

Jan 26 17:48
dolly.urdf
rviz.launch.py
dolly > urdf > dolly.urdf > robot
Ln 23, Col 30 Spaces: 4 UTF-8 LF urdf



```
<?xml version="1.0" ?>
  <robot name="dolly">



  <link name="base">
    <visual>
      <geometry>
        <box size="0.75 0.4 0.1"/>
      </geometry>
      <material name="gray">
        <color rgba=".2 .2 .2 1" />
      </material>
    </visual>

    <inertial>
        <mass value="1" />
        <inertia ixx="0.01" ixy="0.0" ixz="0" iyy="0.01" iyz="0" izz="0.01" />
    </inertial>

    <collision>
      <geometry>
        <box size="0.75 0.4 0.1"/>
      </geometry>
    </collision>

  </link>






  <link name="wheel_right_link">
    <inertial>
            <mass value="2" />
            <inertia ixx="0.01" ixy="0.0" ixz="0"
            iyy="0.01" iyz="0" izz="0.01" />
    </inertial>

    <visual>
      <geometry>
        <cylinder radius="0.15" length="0.1"/>
      </geometry>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
```

Jan 26 17:49
dolly.urdf - src - Visual Studio Code
dolly.urdf
rviz.launch.py
gazebo.launch.py
Extension: XML Tools
URDF Preview
dolly > urdf > dolly.urdf > robot > link > visual > geometry
Ln 8, Col 17 Spaces: 4 UTF-8 LF urdf