# ROS2 Guide Book

Prepared by
Dominic Nightingale
Department of Mechanical and Aerospace Engineering
University of California, San Diego
9500 Gilman Dr, La Jolla, CA 92093

UC San Diego
JACOBS SCHOOL OF ENGINEERING

# TABLE OF CONTENTS

# Chapter 1: General Information

| Command | Description |
|---|---|
| **ros2 launch <package_name> <launch_file.py>** | Launch ROS programs |
| **ros2 node list** | Shows active nodes |
| **ros2 node info <node_name>** | Shows all connections node has |
| **ros2 run <package_name> <python_file.py>** | Run specific python scripts |
| **ros2 daemon stop** | Stop "master" node (Do this if nodes/topics/services are missing) |
| **ros2 daemon start** | Start "master" node |
| **ros2 bag record <topic_name>** | Record data from a topic |
| **ros2 bag record -o <bag_file_name> <topic_name>** | Record data from a topic to specific bagfile name |
| **ros2 bag info <bag_file_name>** | Details of recording |
| **ros2 bag play <bag_file_name>** | Play data from a bag file |

# Create a New Package (steps)

| Command | Description |
|---|---|
| **source /opt/ros/foxy/setup.bash** | source ROS2 to be able to use ROS2 command-line tools |
| **cd ~/ros2_ws/src** | Change directories into ros2_ws/src |
| **ros2 pkg create --build-type ament_python <package_name> --dependencies <package_dependencies>** | Create a new python package |
| **ros2 pkg create --build-type ament_python my_package --dependencies rclpy** | Create a new package with rclpy dependency (ROS client libraries allow nodes written in various programming languages to communicate. A core ROS client library (RCL) implements the standard functionality needed by various ROS APIs. This makes it easier to write language-specific client libraries.) |
| **cd ~/ros2_ws** | Change directories into ros2_ws |
| **colcon build** | Compile packages |
| **source install/setup.bash** | Sets newly generated messages/packages |
| **Command Options** | **Description** |
| **colcon build --packages-select <package_name>** | Only compiles <package_name> and its dependencies. |
| **ros2 pkg list** | Gives a list with all packages in local ROS2 system |
| **ros2 pkg list \| grep my_package** | Filters, from all of the packages located in the local ROS2 system, the package is named my_package. |

# Chapter 2: Topics

| Command | Description |
|---|---|
| **ros2 topic list** | Shows all available topics |
| **ros2 topic list \| grep '<topic>’** | Shows specified topic (if it exists) |
| **ros2 topic info /<topic>** | Shows information about topic |
| **ros2 topic hz /<topic>** | Shows the publishing frequency |
| **ros2 topic echo /<topic>** | Shows realtime output from specified topic |
| **ros2 topic -h** | Displays list of options this command has |
| **ros2 topic pub --once <topic_name> <message_type> "{<message structure>}"** | Publish specified message in topic only **once** |
| **ros2 topic pub <topic_name> <message_type> "{<message structure>}"** | Publish specified message to a topic continuously |
| **ros2 interface -h** | helper for interface |
| **ros2 interface list** | Lists the available interfaces |
| **ros2 interface show <message>** | Shows information about a specified message |
| **ros2 interface proto <message>** | Display structure of message |

## Simple Topic Publisher

**counter_publisher.py**

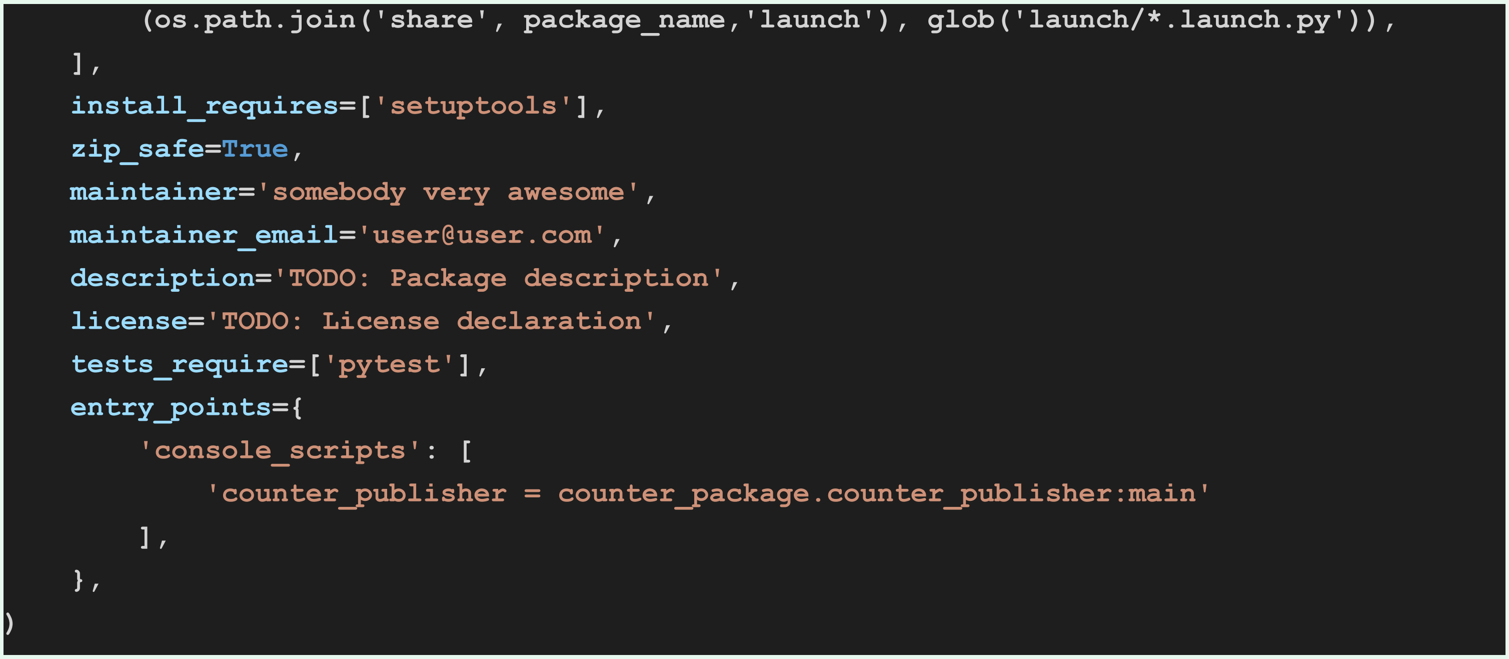
```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32

class SimplePublisher(Node):
    def __init__(self):
        # call super() in the constructor in order to initialize the Node object with
node name as only parameter
        super().__init__('counter_publisher')
        self.publisher_ = self.create_publisher(Int32, '/counter', 1)
        self.count = Int32()
        self.count.data = 0
        timer_period = 1.0 # define the timer period
        self.timer = self.create_timer(timer_period, self.talker_callback)

    def talker_callback(self):
        self.count.data+=1
        self.publisher_.publish(self.count)

def main(args=None):
    rclpy.init(args=args) # initialize the ROS communication
    simple_publisher = SimplePublisher() # declare the node constructor
    rclpy.spin(simple_publisher) # pause the program execution, waits for a request to
kill the node (ctrl+c)
    simple_publisher.destroy_node() # Explicitly destroy the node
    rclpy.shutdown() # shutdown the ROS communication

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

counter_package_launch_file.launch.py

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


def generate_launch_description():
   package_name = 'counter_package'

   ld = LaunchDescription()

   pub_counter_node = Node(
           package=package_name,
           executable='counter_publisher',
           output='screen')

   ld.add_action(pub_counter_node)
   return ld
```

setup.py

```
from setuptools import setup
import os
from glob import glob

package_name = 'counter_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,'launch'), glob('launch/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='somebody very awesome',
    maintainer_email='user@user.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'counter_publisher = counter_package.counter_publisher:main'
        ],
    },
)
```

**Running the script above:**
**Create Package and files**

```
cd ~/ros2_ws
cd src
ros2 pkg create --build-type ament_python counter_package --dependencies rclpy
std_msgs
cd counter_package/counter_package
touch counter_publisher.py
chmod +x counter_publisher.py
**Then add the code to the counter_publisher.py file
cd ~/ros2_ws/src/counter_package
mkdir launch
cd ~/ros2_ws/src/counter_package/launch
touch counter_package_launch_file.launch.py
chmod +x counter_package_launch_file.launch.py
**Then add the publisher node to counter_package_launch_file.launch.py file
**Then update setup.py file
**Then compile package
cd ~/ros2_ws
colcon build
source ~/ros2_ws/install/setup.bash
```

**Running**

```
ros2 launch counter_package counter_package_launch_file.launch.py
ros2 topic echo /counter
```

## Simple Topic Subscriber

counter_subscriber.py

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32


class SimpleSubscriber(Node):
    def __init__(self):
        super().__init__('counter_subscriber')
        self.subscriber=
self.create_subscription(Int32,'/counter',self.listener_callback, 1)
        self.subscriber
        self.view_count = Int32()

    def listener_callback(self, msg):
        self.view_count.data = msg.data
        self.get_logger().info('Current count: %s' % (msg.data))
        # print(self.view_count)
        print(msg.data)

def main(args=None):
    rclpy.init(args=args) # initialize the ROS communication
    simple_subscriber = SimpleSubscriber() # declare the node constructor
    rclpy.spin(simple_subscriber) # pause the program execution, waits for a request to
kill the node (ctrl+c)
    simple_subscriber.destroy_node() # Explicitly destroy the node
    rclpy.shutdown() # shutdown the ROS communication

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

**counter_package_launch_file.launch.py**

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


def generate_launch_description():
    package_name = 'counter_package'

    ld = LaunchDescription()

    pub_counter_node = Node(
            package=package_name,
            executable='counter_publisher',
            output='screen')

    sub_counter_node = Node(
            package=package_name,
            executable='counter_subscriber',
            output='screen')

    ld.add_action(pub_counter_node)
    ld.add_action(sub_counter_node)
    return ld
```

**setup.py**

```
from setuptools import setup
import os
from glob import glob

package_name = 'counter_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/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='somebody very awesome',
    maintainer_email='user@user.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'counter_publisher = counter_package.counter_publisher:main',
            'counter_subscriber = counter_package.counter_subscriber:main'
        ],
    },
)
```

**Running the script above:**
**Create Package and files**

```
     cd ~/ros2_ws/src/counter_package/counter_package
     touch counter_subscriber.py
     chmod +x counter_subscriber.py
     **Then add the code to the counter_subscriber.py file
     **Then add the subscriber node to counter_package_launch_file.launch.py file
     **Then update setup.py file
     **Then compile package
     cd ~/ros2_ws
     colcon build --packages-select counter_package
     source install/setup.bash
```

**Running**

```
     ros2 launch counter_package counter_package_launch_file.launch.py
```

## Create Custom Interface

cd ~/ros2_ws/src
ros2 pkg create --build-type **ament_cmake** custom_interfaces

To create a new message, do the following:

1. Create a directory named msg inside your package
2. Inside this directory, create a file named Name_of_your_message.msg (more - information below)
3. Modify the CMakeLists.txt file (more information below)
4. Modify package.xml file (more information below)
5. Compile and source
6. Check interface was created successfully

**Example**

1. Create a directory msg in your package
   cd ~/ros2_ws/src/custom_interfaces
   mkdir msg
   touch **Age.msg**
   chmod +x **Age.msg**
2. Copy and paste the following in the **Age.msg** file:
```
float32 years
float32 months
float32 days
```
3. Edit two sections inside **CMakeLists.txt**:
```
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)
```
```
# add at the end before ament_package()
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Age.msg"
```
4. Add following lines to **package.xml** file
```
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

```
5.  roscd; cd ..
    cd ~/ros2_ws
    colcon build --packages-select custom_interfaces
    source install/setup.bash
6.  ros2 interface show custom_interfaces/msg/Age
```

## Topics Summary

A topic is like a pipe. **Nodes use topics to publish information for other nodes** so that they can communicate. You can find out, at any time, the number of topics in the system by doing a **rostopic list**. You can also check for a specific topic.

**A publisher is a node that keeps publishing a message into a topic**.

**A topic is a channel that acts as a pipe, where other ROS nodes can either publish or read information**.

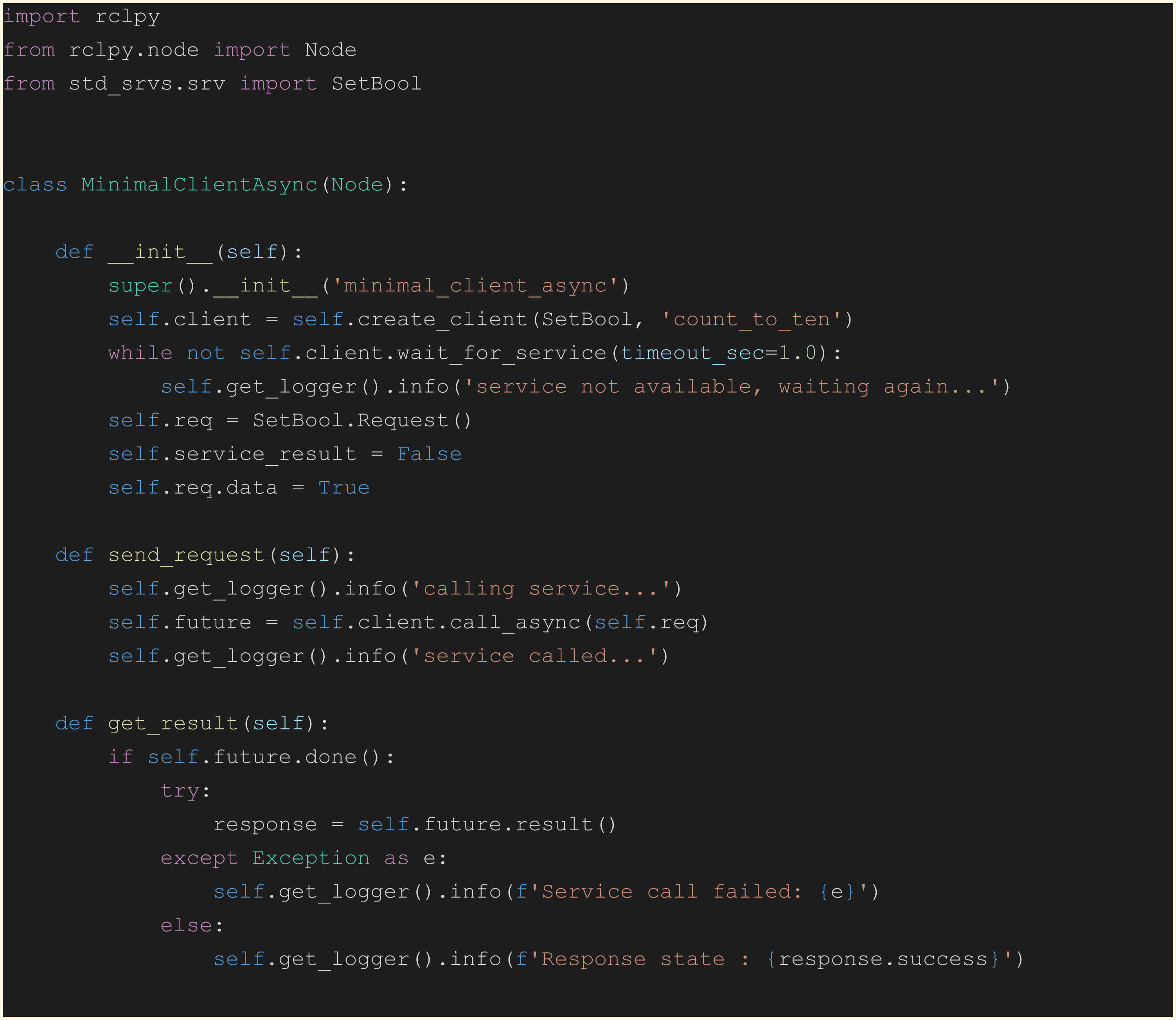
Topics handle information through messages. There are many different types of messages.
Messages are defined in **.msg** files, which are located inside a **msg** directory of a package.

# Chapter 3: Services

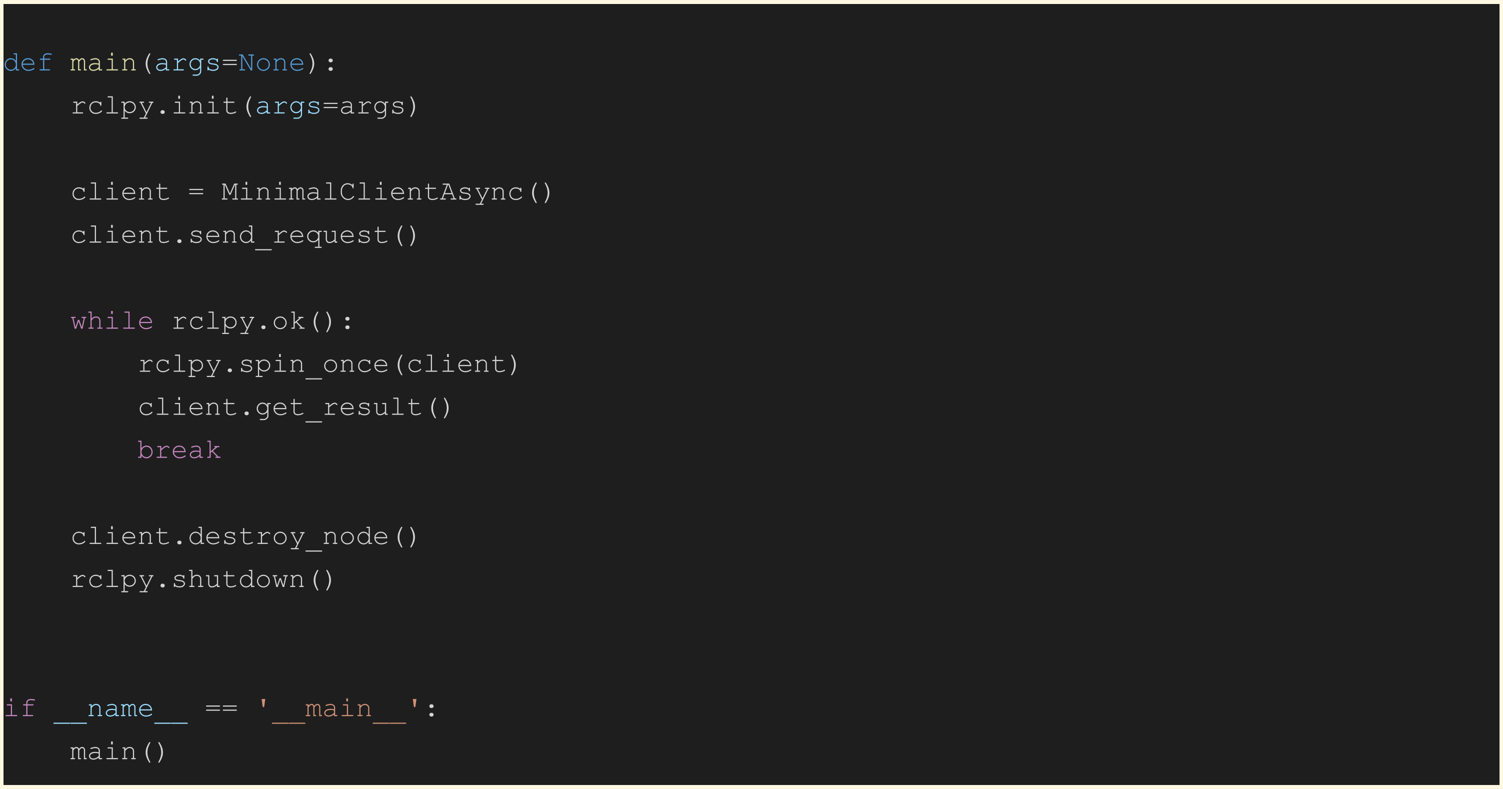
| Command | Description |
|---|---|
| `ros2 service list` | Shows list of services running |
| `ros2 service type /<service_name>` | Shows information about a specified service |
| `ros2 service call <service_name> <service_type> <value>` | Call a service (for testing) |
| `ros2 interface show /<service_type>` | Shows structure of service |

## Simple Service Client

**service_client.py**



```python
import rclpy
from rclpy.node import Node
from std_srvs.srv import SetBool


class MinimalClientAsync(Node):

    def __init__(self):
        super().__init__('minimal_client_async')
        self.client = self.create_client(SetBool, 'count_to_ten')
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')
        self.req = SetBool.Request()
        self.service_result = False
        self.req.data = True

    def send_request(self):
        self.get_logger().info('calling service...')
        self.future = self.client.call_async(self.req)
        self.get_logger().info('service called...')

    def get_result(self):
        if self.future.done():
            try:
                response = self.future.result()
            except Exception as e:
                self.get_logger().info(f'Service call failed: {e}')
            else:
                self.get_logger().info(f'Response state : {response.success}')
```

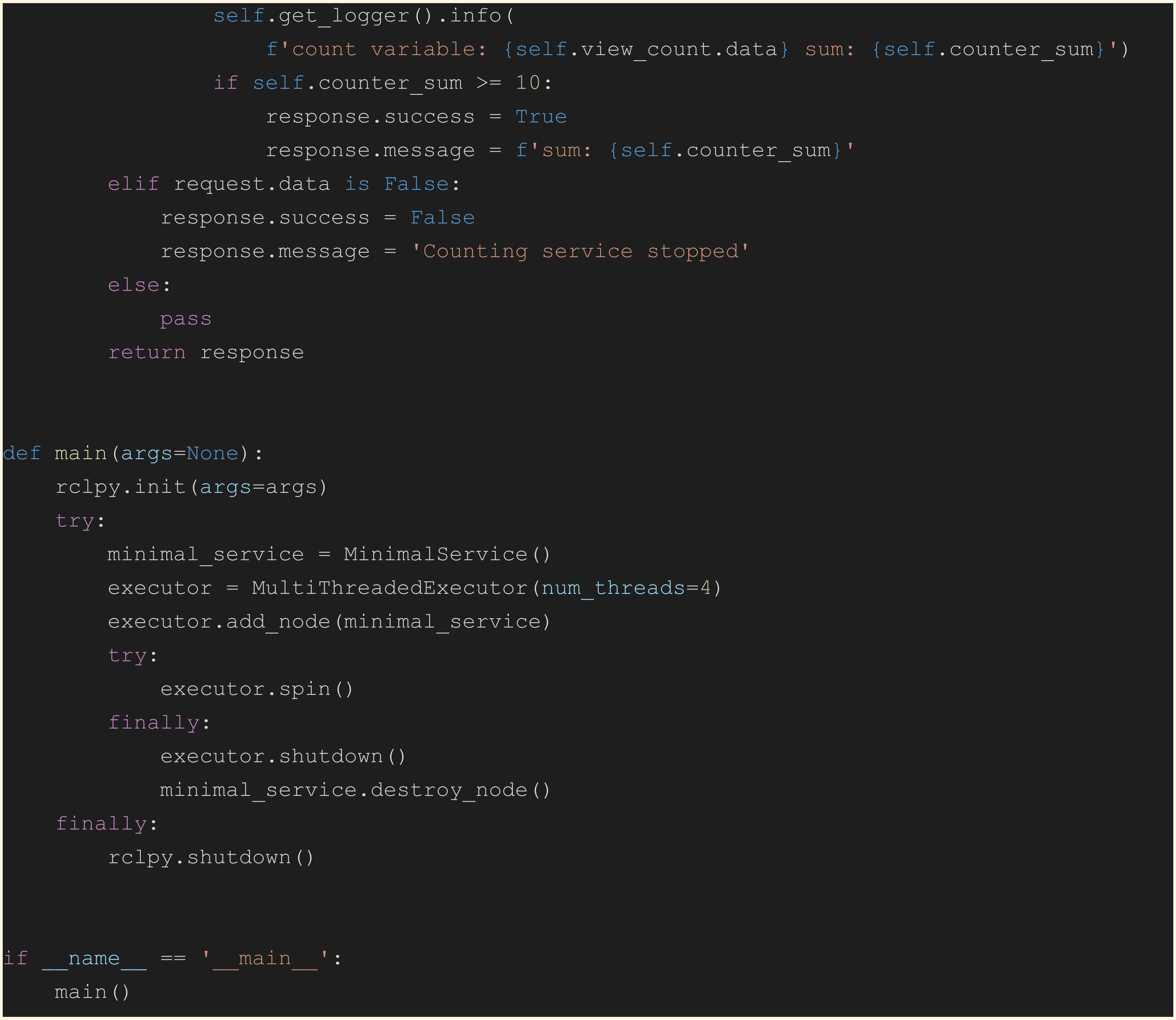


```python
def main(args=None):
    rclpy.init(args=args)

    client = MinimalClientAsync()
    client.send_request()

    while rclpy.ok():
        rclpy.spin_once(client)
        client.get_result()
        break

    client.destroy_node()
    rclpy.shutdown()


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

## Simple Service Server

**service_server.py**

```
from std_srvs.srv import SetBool
import rclpy
from rclpy.node import Node
from rclpy.callback_groups import MutuallyExclusiveCallbackGroup
from rclpy.executors import MultiThreadedExecutor
from std_msgs.msg import Int32


class MinimalService(Node):

    def __init__(self):
        super().__init__('count_to_ten')

        # Multithreading
        self.service_thread = MutuallyExclusiveCallbackGroup()
        self.count_thread = MutuallyExclusiveCallbackGroup()
        self.srv = self.create_service(SetBool, 'count_to_ten',
self.count_to_ten_callback, callback_group=self.service_thread)
        self.counter_subscriber = self.create_subscription(Int32, '/counter',
self.counter_callback, 10, callback_group=self.count_thread)
        self.counter_subscriber
        self.counter_sum = 0
        self.view_count = Int32()

    def counter_callback(self, msg):
        self.view_count.data = msg.data
        self.counter_sum += msg.data

    def count_to_ten_callback(self, request, response):
        if request.data is True:
            while self.counter_sum <= 10:
```



```
                self.get_logger().info(
                    f'count variable: {self.view_count.data} sum: {self.counter_sum}')
                if self.counter_sum >= 10:
                    response.success = True
                    response.message = f'sum: {self.counter_sum}'
        elif request.data is False:
            response.success = False
            response.message = 'Counting service stopped'
        else:
            pass
        return response


def main(args=None):
    rclpy.init(args=args)
    try:
        minimal_service = MinimalService()
        executor = MultiThreadedExecutor(num_threads=4)
        executor.add_node(minimal_service)
        try:
            executor.spin()
        finally:
            executor.shutdown()
            minimal_service.destroy_node()
    finally:
        rclpy.shutdown()


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

**start_simple_service_launch.launch.py**

```python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, GroupAction
from launch.substitutions import LaunchConfiguration
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node
from launch_ros.actions import PushRosNamespace
import yaml


def generate_launch_description():
    service_package = 'simple_service_pkg'
    service_server_node_name = 'simple_service_server'
    service_client_node_name = 'simple_service_client'
    counter_publisher_node_name = 'counter_publisher'
    ld = LaunchDescription()

    service_server = Node(
        package=service_package,
        executable=service_server_node_name,
        output='screen')
    service_client = Node(
        package=service_package,
        executable=service_client_node_name,
        output='screen')
    counter_publisher = Node(
            package=service_package,
            executable=counter_publisher_node_name,
            output='screen')
    ld.add_action(service_server)
    ld.add_action(service_client)
    ld.add_action(counter_publisher)
    return ld
```

**setup.py**

```
from setuptools import setup
import os
from glob import glob


package_name = 'simple_service_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']),
        (os.path.join('share', package_name), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'simple_service_server = simple_service_pkg.simple_service_server:main',
            'simple_service_client = simple_service_pkg.simple_service_client:main',
            'counter_publisher = counter_package.counter_publisher:main',
        ],
    },
)
```

```
Running the scripts above:

(Notice I used the counter publisher from another package for this exercise)

Create Package and files
     cd ~/ros2_ws/src
     ros2 pkg create --build-type ament_python simple_service_pkg --dependencies
     rclpy std_srvs
     cd simple_service_pkg
     touch simple_service_server.py
     touch simple_service_client.py
     mkdir launch
     touch launch/start_simple_service_launch.launch.py
     chmod +x service_server.py simple_service_client.py
     chmod +x launch/start_simple_service_launch.launch.py
     (copy and paste code above into each corresponding python files)
     cd ~/ros2_ws
     colcon build --packages-select counter_package
     colcon build --packages-select simple_service_pkg
     source ~/ros2_ws/install/setup.bash
Running
     Terminal 1: ros2 launch simple_service_pkg
     start_simple_service_launch.launch.py
Output
[INFO] [launch]: All log files can be found below
/home/user/.ros/log/2022-04-15-22-18-09-517721-1_xterm-31880
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [simple_service_server-1]: process started with pid [31882]
[INFO] [simple_service_client-2]: process started with pid [31884]
[INFO] [counter_publisher-3]: process started with pid [31886]
[simple_service_client-2] [INFO] [1650061090.841002784] [minimal_client_async]:
calling service...
[simple_service_client-2] [INFO] [1650061090.841910705] [minimal_client_async]:
service called...
[simple_service_server-1] [INFO] [1650061090.949675998] [count_to_ten]: count
variable: 2 sum: 3
[simple_service_server-1] [INFO] [1650061090.950183924] [count_to_ten]: count
variable: 3 sum: 6
[simple_service_server-1] [INFO] [1650061091.142073051] [count_to_ten]: count
variable: 4 sum: 10
[simple_service_client-2] [INFO] [1650061091.143524907] [minimal_client_async]:
Response state : True
[INFO] [simple_service_client-2]: process has finished cleanly [pid 31884]
```

## Simple Service Launch and Setup.py

**start_simple_service_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, GroupAction
from launch.substitutions import LaunchConfiguration
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node
from launch_ros.actions import PushRosNamespace
import yaml


def generate_launch_description():
    service_package = 'simple_service_pkg'
    counter_package = 'counter_package'
    service_server_node_name = 'simple_service_server'
    service_client_node_name = 'simple_service_client'
    counter_publisher_node_name = 'counter_publisher'
    ld = LaunchDescription()

    service_server = Node(
        package=service_package,
        executable=service_server_node_name,
        output='screen')
    service_client = Node(
        package=service_package,
        executable=service_client_node_name,
        output='screen')
    counter_publisher = Node(
            package=counter_package,
            executable=counter_publisher_node_name,
            output='screen')
    ld.add_action(service_server)
    ld.add_action(service_client)
    ld.add_action(counter_publisher)
    return ld
```

**setup.py**

```
from setuptools import setup
import os
from glob import glob


package_name = 'simple_service_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']),
        (os.path.join('share', package_name), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'simple_service_server = simple_service_pkg.simple_service_server:main',
            'simple_service_client = simple_service_pkg.simple_service_client:main',
            'counter_publisher = counter_package.counter_publisher:main',
        ],
    },
)
```

**Running the scripts above:**

**(Notice I used the counter publisher from another package for this exercise)**

**Create Package and files**

```
     cd ~/ros2_ws/src
     ros2 pkg create --build-type ament_python simple_service_pkg --dependencies
     rclpy std_srvs
     cd simple_service_pkg
     touch simple_service_server.py
     touch simple_service_client.py
     mkdir launch
     touch launch/start_simple_service_launch.launch.py
     chmod +x service_server.py simple_service_client.py
     chmod +x launch/start_simple_service_launch.launch.py
     (copy and paste code above into each corresponding python files)
     cd ~/ros2_ws
     colcon build --packages-select counter_package
     colcon build --packages-select simple_service_pkg
     source ~/ros2_ws/install/setup.bash
```

**Running**

```
     Terminal 1: ros2 launch simple_service_pkg
     start_simple_service_launch.launch.py
```

**Output**

```
[INFO] [launch]: All log files can be found below
/home/user/.ros/log/2022-04-15-22-18-09-517721-1_xterm-31880
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [simple_service_server-1]: process started with pid [31882]
[INFO] [simple_service_client-2]: process started with pid [31884]
[INFO] [counter_publisher-3]: process started with pid [31886]
[simple_service_client-2] [INFO] [1650061090.841002784] [minimal_client_async]:
calling service...
[simple_service_client-2] [INFO] [1650061090.841910705] [minimal_client_async]:
service called...
[simple_service_server-1] [INFO] [1650061090.949675998] [count_to_ten]: count
variable: 2 sum: 3
[simple_service_server-1] [INFO] [1650061090.950183924] [count_to_ten]: count
variable: 3 sum: 6
[simple_service_server-1] [INFO] [1650061091.142073051] [count_to_ten]: count
variable: 4 sum: 10
[simple_service_client-2] [INFO] [1650061091.143524907] [minimal_client_async]:
Response state : True
[INFO] [simple_service_client-2]: process has finished cleanly [pid 31884]
```

## Services Summary

A **ROS Service** provides a certain functionality of your robot. A ROS Service is composed of 2 parts:

- **Service Server**: This is what PROVIDES the functionality. Whatever you want your Service to do, you have to place it in the Service Server.
- **Service Client**: This is what CALLS the functionality provided by the Service Server. That is, it CALLS the Service Server.

ROS Services use a special service message, which is composed of 2 parts:

- **Request**: The request is the part of the message that is used to CALL the Service. Therefore, it is sent by the Service Client to the Service Server.
- **Response**: The response is the part of the message that is returned by the Service Server to the Service Client, once the Service has finished.

**ROS Services are synchronous**. This means that whenever you CALL a Service Server, you have to wait until the Service has finished (and returns a response) before you can do other stuff with your robot.

# Chapter 4: Actions

| Command | Description |
|---|---|
| ros2 action list | Output a list of action names |
| ros2 action info -t | Print information about an action including type |
| ros2 action send_goal | Send an action goal |
| ros2 interface show /<action_type> | Shows structure of action |

## Simple Action Client

**action_client.py**

```
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from t3_action_msg.action import Move

class MyActionClient(Node):
    def __init__(self):
        super().__init__('action_client')
        self._action_client = ActionClient(self, Move, 'turtlebot3_as')

    def send_goal(self, secs):
        goal_msg = Move.Goal()
        goal_msg.secs = secs
        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(goal_msg,
feedback_callback=self.feedback_callback)
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            return
        self.get_logger().info('Goal accepted :)')
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: {0}'.format(result.status))
        rclpy.shutdown()
    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback
        self.get_logger().info('Received feedback: {0}'.format(feedback.feedback))


def main(args=None):
    rclpy.init(args=args)
    action_client = MyActionClient()
    future = action_client.send_goal(5)
    rclpy.spin(action_client)


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

**setup.py**

```
from setuptools import setup
import os
from glob import glob

package_name = 'action_client_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']),
        (os.path.join('share', package_name), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'action_client = action_client_pkg.action_client:main'
        ],
    },
)
```

**Running the script above:**

**Create Package and files**

```
      cd ~/ros2_ws/src
      ros2 pkg create action_client_pkg --build-type ament_python --dependencies rclpy
      rclpy.action t3_action_msg
      (above is 1 line, not 2!)
      touch action_client.py
      chmod +x action_client.py (copy and paste code above into action_client.py)
      cd ~/ros2_ws
      colcon build --packages-select action_client_pkg
      source ~/ros2_ws/install/setup.bash
```

**Running**

```
      ros2 run action_client_pkg action_client
```

**Output**

```
     [INFO] [1642135979.103202364] [my_action_client]: Goal accepted :)
[INFO] [1642135979.104456839] [action_client]: Received feedback: Movint to the left
left left...
[INFO] [1642135980.075346736] [action_client]: Received feedback: Movint to the left
left left...
[INFO] [1642135981.075628695] [action_client]: Received feedback: Movint to the left
left left...
[INFO] [1642135982.075227936] [action_client]: Received feedback: Movint to the left
left left...
[INFO] [1642135983.075254763] [action_client]: Received feedback: Movint to the left
left left...
[INFO] [1642135984.075622401] [action_client]: Result: Finished action server. Robot
moved during 5 seconds
```

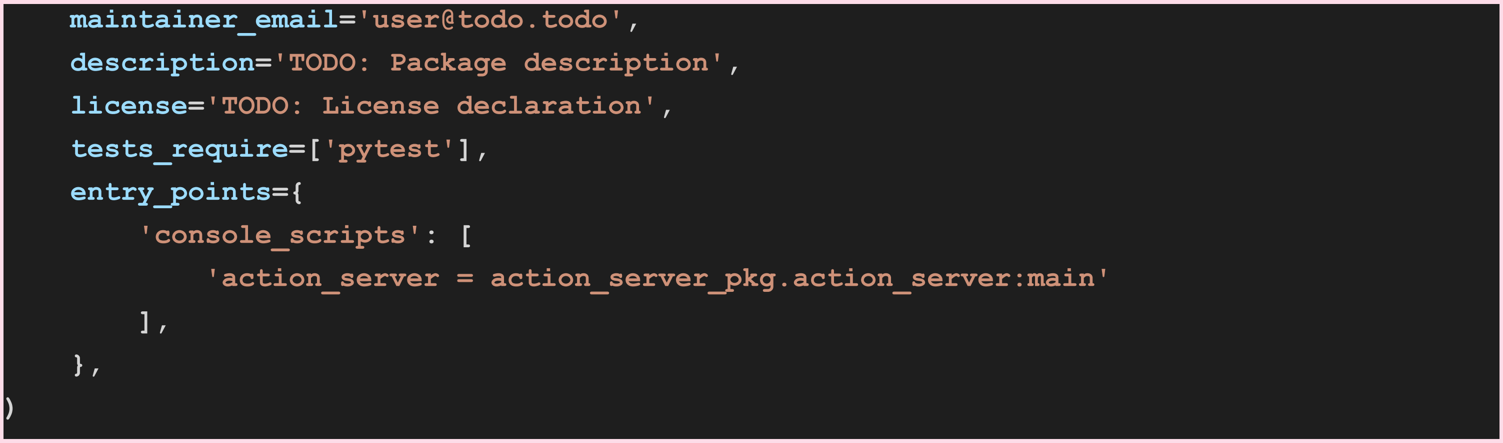
## Simple Action Server

**action_server.py**

```
import rclpy
from rclpy.action import ActionServer
from rclpy.node import Node
from t3_action_msg.action import Move
from geometry_msgs.msg import Twist
import time


class MyActionServer(Node):
    def __init__(self):
        super().__init__('action_server')
        self._action_server = ActionServer(self, Move,
'turtlebot3_as',self.execute_callback)
        self.cmd = Twist()
        self.publisher_ = self.create_publisher(Twist, 'cmd_vel', 10)


    def execute_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')
        feedback_msg = Move.Feedback()
        feedback_msg.feedback = "Moving to the left left left..."
        for i in range(1, goal_handle.request.secs):
            self.get_logger().info('Feedback: '.format(feedback_msg.feedback))
            goal_handle.publish_feedback(feedback_msg)
            self.cmd.linear.x = 0.3
            self.cmd.angular.z =0.3
            self.publisher_.publish(self.cmd)
            time.sleep(1)

        goal_handle.succeed()
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = 0.0
        self.publisher_.publish(self.cmd)
        feedback_msg.feedback = "Finished action server. Robot moved during 5 seconds"
        result = Move.Result()
        result.status = feedback_msg.feedback
        return result

def main(args=None):
    rclpy.init(args=args)
    my_action_server = MyActionServer()
    rclpy.spin(my_action_server)


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

**setup.py**

```
from setuptools import setup
import os
from glob import glob


package_name = 'action_server_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']),
        (os.path.join('share', package_name), glob('launch/*.launch.py'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
```



```
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'action_server = action_server_pkg.action_server:main'
        ],
    },
)
```

**Running the script above:**

**Create Package and files**

```
      cd ~/ros2_ws/src
      ros2 pkg create action_server_pkg --build-type ament_python --dependencies rclpy
      rclpy.action t3_action_msg
      (above is 1 line, not 2!)
      touch action_server.py
      chmod +x action_server.py (copy and paste code above into action_server.py)
      cd ~/ros2_ws
      colcon build --packages-select action_server_pkg
      source ~/ros2_ws/install/setup.bash
```

**Running**

**Terminal 1:** `ros2 run action_server_pkg action_server`

**Terminal 2:** `ros2 run action_client_pkg action_client`

**Output**

**Terminal 1:**

```
    [INFO] [1642136849.463852445] [action_server]: Executing goal...
[INFO] [1642136849.464673665] [action_server]: Feedback:
[INFO] [1642136850.466904531] [action_server]: Feedback:
[INFO] [1642136851.468581656] [action_server]: Feedback:
[INFO] [1642136852.470568331] [action_server]: Feedback:
```

**Terminal 2:**

```
    [INFO] [1642136849.492653101] [my_action_client]: Goal accepted :)
[INFO] [1642136849.494232660] [action_client]: Received feedback: Moving to the
left left left...
[INFO] [1642136850.468283205] [action_client]: Received feedback: Moving to the
left left left...
[INFO] [1642136851.469938741] [action_client]: Received feedback: Moving to the
left left left...
[INFO] [1642136853.474947356] [action_client]: Result: Finished action server.
Robot moved during 5 seconds
```

## Actions Summary

**Actions are like asynchronous calls to services**
Actions are very similar to services. When you call an action, you are calling a functionality that another node is providing. Just the same as with services. The difference is that when your node calls a service, it must wait until the service finishes. **When your node calls an action, it doesn't necessarily have to wait for the action to complete.**

**Hence, an action is an asynchronous call to another node's functionality**.

- The node that provides the functionality has to contain an **action server**. The *action server* allows other nodes to call that action functionality.
- The node that calls to the functionality has to contain an **action client**. The *action client* allows a node to connect to the *action server* of another node.

**Calling an action server means sending a message to it**. In the same way as with *topics* and *services*, it all works by passing messages around.

- The message of a topic is composed of a single part: the information the topic provides.
- The message of a service has two parts: the request and the response.
- **The message of an action server is divided into three parts: the goal, the result, and the feedback.**

An action message has three parts:

- the goal
- the result
- the feedback

So, whenever an action server is called, the sequence of steps are as follows:

1. When an **action client** calls an **action server** from a node, what actually happens is that the **action client** sends to the **action server** the goal requested through the /ardrone_action_server/goal topic.

2. When the **action server** starts to execute the goal, it sends to the **action client** the feedback through the /ardrone_action_server/feedback topic.

3. Finally, when the **action server** has finished the goal, it sends to the **action client** the result through the /ardrone_action_server/result topic.

# Chapter 5: Topics - Services - Actions

To understand what services are and when to use them, you have to compare them with topics and actions.

Imagine you have your own personal BB-8 robot. It has a laser sensor, a face-recognition system, and a navigation system. The laser will use a **Topic** to publish all of the laser readings at 20hz. We use a topic because we need to have that information available all the time for other ROS systems, such as the navigation system.

The Face-recognition system will provide a **Service**. Your ROS program will call that service and **WAIT** until it gives you the name of the person BB-8 has in front of it.

The navigation system will provide an **Action**. Your ROS program will call the action to move the robot somewhere, and **WHILE** it's performing that task, your program will perform other tasks, such as complain about how tiring C-3PO is. And that action will give you **Feedback** (for example: distance left to the desired coordinates) along the process of moving to the coordinates.

So... What's the difference between a **Service** and an **Action**?

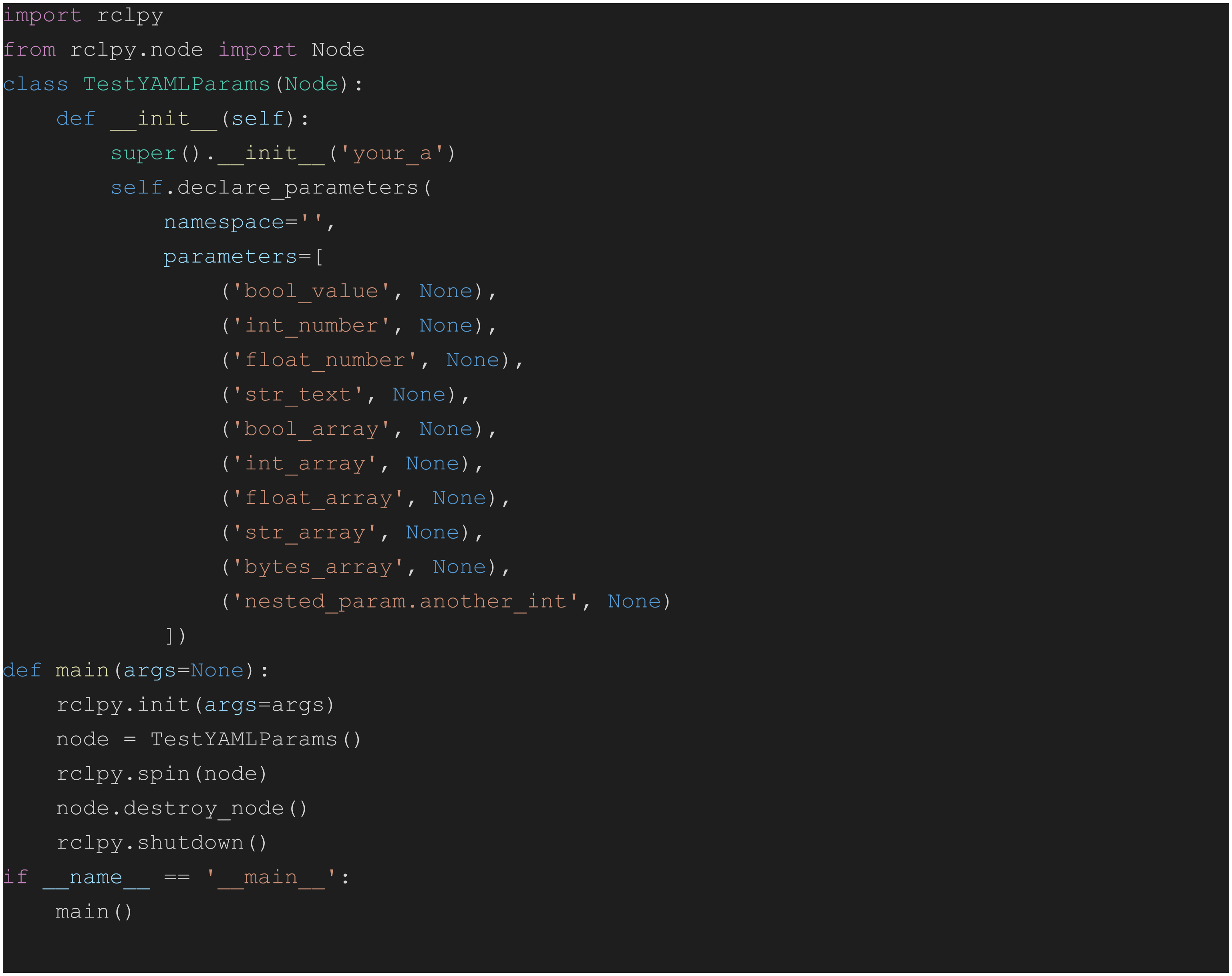
Services are **Synchronous**. When your ROS program calls a service, your program can't continue until it receives a result from the service.
Actions are **Asynchronous**. It's like launching a new thread. When your ROS program calls an action, your program can perform other tasks while the action is being performed in another thread.

**Conclusion: Use services when your program can't continue until it receives the result from the service.**

# Chapter 6: Parameters

| Command | Description |
|---|---|
| ros2 param list | To see the parameters belonging to available nodes |
| ros2 param get <node_name> <parameter_name> | To display the type and current value of a parameter |
| ros2 param set <node_name> <parameter_name> <value> | To change a parameter’s value at runtime |
| ros2 param dump <node_name> | To save all of a node’s current parameter values into a file to save for later |
| ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>.yaml | To load parameter values from a yaml file |

# Load YAML Parameters from a Node



```
import rclpy
from rclpy.node import Node
class TestYAMLParams(Node):
    def __init__(self):
        super().__init__('your_a')
        self.declare_parameters(
            namespace='',
            parameters=[
                ('bool_value', None),
                ('int_number', None),
                ('float_number', None),
                ('str_text', None),
                ('bool_array', None),
                ('int_array', None),
                ('float_array', None),
                ('str_array', None),
                ('bytes_array', None),
                ('nested_param.another_int', None)
            ])
def main(args=None):
    rclpy.init(args=args)
    node = TestYAMLParams()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
if __name__ == '__main__':
    main()
```

## Set Parameters in a Node

```
import rclpy
import rclpy.node
from rclpy.exceptions import ParameterNotDeclaredException
from rcl_interfaces.msg import ParameterType

class MinimalParam(rclpy.node.Node):
    def __init__(self):
        super().__init__('minimal_param_node')
        timer_period = 2  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

        self.declare_parameter('my_parameter', 'world')

    def timer_callback(self):
        my_param = self.get_parameter('my_parameter').get_parameter_value().string_value

        self.get_logger().info('Hello %s!' % my_param)

        my_new_param = rclpy.parameter.Parameter(
            'my_parameter',
            rclpy.Parameter.Type.STRING,
            'world'
        )
        all_new_parameters = [my_new_param]
        self.set_parameters(all_new_parameters)

def main():
    rclpy.init()
    node = MinimalParam()
    rclpy.spin(node)

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

## Create YAML File

**params.yaml**

```
<name_of_node>:
  ros__parameters:
    bool_value: True
    int_number: 5
    float_number: 3.14
    str_text: "Hello Universe"
    bool_array: [True, False, True]
    int_array: [10, 11, 12, 13]
    float_array: [7.5, 400.4]
    str_array: ['Nice', 'more', 'params']
    bytes_array: [0x01, 0xF1, 0xA2]
```

# Chapter 7: Launch Files

## Single Node Launch File Scheme

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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='<name_of_package>',
            executable='<name_of_python_file>',
            output='screen'),])
```

## Multi Node Launch File Scheme

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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='<name_of_package>',
            executable='<name_of_python_file>',
            output='screen'),
        Node(
            package='<name_of_package>',
            executable='<name_of_python_file>',
            output='screen'),])
```

## Load YAML File

**params_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    ld = LaunchDescription()
    config = os.path.join(
        get_package_share_directory('ros2_tutorials'),
        'config',
        'params.yaml'
        )

    node=Node(
        package = '<name_of_package>',
        executable = '<name_of_python_file>',
        name = '<name_of_node>',
        parameters = [config]
    )
```

## Setting Fixed Parameters

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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='<name_of_package>',
            executable='<name_of_python_file>',
            name='<name_of_node>',
            output='screen',
            emulate_tty=True,
            parameters=[
                {'parameter_1': 'test'},
                {'parameter_2': 1}
            ]
        )
    ])
```

## Call another Launch file

**another_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, LogInfo
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node


def generate_launch_description():
    some_package = 'counter_package'
    some_launch = 'counter_package_launch_file.launch.py'
    ld = LaunchDescription()
    counter_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(
                get_package_share_directory(some_package),
                some_launch)
        )
    )
    ld.add_action(counter_launch)
    return ld
```

## Call another Launch file based on yaml input

**yaml_launch.launch.py**

```python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, GroupAction
from launch.substitutions import LaunchConfiguration
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node
from launch_ros.actions import PushRosNamespace
import yaml

file_parameter_input_path = str(os.path.dirname(__file__) + '/car_config.yaml')

def update_parameters(file_parameter_input_path):
        with open(file_parameter_input_path, "r") as file:
            car_inputs = yaml.load(file, Loader=yaml.FullLoader)
            return car_inputs

def generate_launch_description():
    car_inputs_dict = update_parameters(file_parameter_input_path)
    some_package = car_inputs_dict['some_package']
    some_launch = car_inputs_dict['some_launch']
    ld = LaunchDescription()
    counter_launch = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                os.path.join(
                    get_package_share_directory(some_package),
                    some_launch)
            )
        )
    ld.add_action(counter_launch)
    return ld
```

## Call multiple Launch files based on yaml input

**multiple_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, GroupAction
from launch.substitutions import LaunchConfiguration
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node
from launch_ros.actions import PushRosNamespace
import yaml

car_parameter_input_path = str(os.path.dirname(__file__) + '/car_config.yaml')
packages_info_path = str(os.path.dirname(__file__) + '/rand_config.yaml')

def update_parameters(car_parameter_input_path):
        with open(car_parameter_input_path, "r") as file:
            car_inputs = yaml.load(file, Loader=yaml.FullLoader)
            my_car_inputs = {}
            for key in car_inputs:
                value = car_inputs[key]
                if value==1:
                    my_car_inputs[key] = value
            return my_car_inputs

def update_packages(packages_info_path):
    with open(packages_info_path, "r") as file:
            packages_dict = yaml.load(file, Loader=yaml.FullLoader)
            car_inputs_dict = update_parameters(car_parameter_input_path)
            my_packages = {}
            for key in packages_dict:
                value = packages_dict[key]
                if key in car_inputs_dict:
                    my_packages[key] = value
            return my_packages

my_packages = update_packages(packages_info_path)
print(my_packages)

def generate_a_launch_description(some_package, some_launch):
    ld = LaunchDescription()
    _launch = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                os.path.join(
                    get_package_share_directory(some_package),
                    some_launch)
            )
        )
    ld.add_action(_launch)
    return ld

def generate_launch_description():
    my_packages_dict = update_packages(packages_info_path)
    for key in my_packages_dict:
        pkg_name = my_packages_dict[key][0]
        launch_name = my_packages_dict[key][1]
        ld = generate_a_launch_description(pkg_name, launch_name)
    return ld
```

## Call ROS1 Launch File

```
import subprocess

bashCommand = "roslaunch my_package_r1 bash_launch.launch"
output = subprocess.run(['bash','-c', bashCommand])
# print("The exit code was: %d" % output.returncode)
```

## Change topic names (runtime)

**another_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, LogInfo
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node
import subprocess


def generate_launch_description():
    return LaunchDescription([
        Node(
            package='counter_package',
            executable='counter_publisher',
            output='screen',
            remappings=[('old_topic_name','new_topic_name')]
            ),
        Node(
            package='counter_package',
            executable='counter_subscriber',
            output='screen',
            remappings=[('old_topic_name','new_topic_name')]
            ),
    ])
```

# Add Launch arguments (ex. Changing topic name)

**another_launch.launch.py**

```python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, LogInfo
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir, LaunchConfiguration
from launch_ros.actions import Node
import subprocess



def generate_launch_description():
    value= LaunchConfiguration('topic_name', default='counter')
    return LaunchDescription([
        Node(
            package='counter_package',
            executable='counter_publisher',
            output='screen',
            remappings=[('counter',value)]
            ),
        Node(
            package='counter_package',
            executable='counter_subscriber',
            output='screen',
            remappings=[('counter',value)]
            ),
    ])
```

## Call another Launch file that has arguments (ex. Changing topic name)

**another_launch.launch.py**

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, LogInfo, DeclareLaunchArgument
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node


def generate_launch_description():
    some_launch = '/change_name_launch.launch.py'
    topic_name = "counter_test"

    return LaunchDescription([
    DeclareLaunchArgument(
            'topic_name',
            default_value = 'counter',
            description = 'Argument for child launch file'
        ),

    IncludeLaunchDescription(
            PythonLaunchDescriptionSource([ThisLaunchFileDir(), some_launch]),
            launch_arguments = {'topic_name': topic_name}.items()
        ),
    ])
```

# Chapter 8: Setup files

## Single Executable Scheme

```python
from setuptools import setup
import os
from glob import glob

package_name = '<name_of_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/*.launch.py')),
        (os.path.join('share', package_name), glob('config/*.yaml'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='somebody very awesome',
    maintainer_email='user@user.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            '<name_of_python_file> = <name_of_package>.<name_of_python_file>:main'
        ],
    },
)
```

## Multi Executable Scheme

```
from setuptools import setup
import os
from glob import glob

package_name = '<name_of_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/*.launch.py')),
        (os.path.join('share', package_name), glob('config/*.yaml'))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='somebody very awesome',
    maintainer_email='user@user.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            '<name_of_python_file> = <name_of_package>.<name_of_python_file>:main',
            '<name_of_python_file> = <name_of_package>.<name_of_python_file>:main'
        ],
    },
)
```

# Setep.py

**setup.py**

```
from setuptools import setup
import os
from glob import glob

package_name = 'counter_package'
submodule_name = 'counter_submodule'
submodule = str(package_name +'/'+ submodule_name)


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/*.launch.py')),
        (os.path.join('share', package_name, submodule_name), glob(submodule_name+
'/*.py'))

    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='somebody very awesome',
    maintainer_email='user@user.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'counter_publisher = counter_package.counter_publisher:main',
        ],
    },
)
```

**Running the script above:**
**Create Package and files**

```
cd ~/ros2_ws/src/counter_package/counter_package
mkdir counter_submodule
cd counter_submodule
touch counter_python.py
chmod +x counter_python.py
**Then update setup.py file
**Then compile package
cd ~/ros2_ws
colcon build --packages-select subscriber_pkg
source ~/ros2_ws/install/setup.bash
```

**Running**

```
ros2 launch counter_package counter_package_launch_file.launch.py
```

# (TO DO) Chapter 9: **Navigation**

## Basic Concepts

## Map Creation

## Robot Localization and Mapping - SLAM

## Path Planning

## Summary

# (TO DO) Chapter 10: **Motion Planning**

## Basic Concepts

## Grid and Sampling Methods

## Virtual Potential Fields

## Lidar

## Odom

Blank Dont Delete

| Blank Dont Delete |
|---|
| |