Writing a simple service and client (Python)

Goal: Create and run service and client nodes using Python.

Tutorial level: Beginner

Time: 20 minutes

Contents

Background

When nodes communicate using services, the node that sends a request for data is called the client node, and the one that responds to the request is the service node. The structure of the request and response is determined by a `.srv` file.

The example used here is a simple integer addition system; one node requests the sum of two integers, and the other responds with the result.

Prerequisites

In previous tutorials, you learned how to create a workspace and create a package.

Tasks

1 Create a package

Open a new terminal and source your ROS 2 installation so that `ros2` commands will work.

Navigate into the `ros2_ws` directory created in a previous tutorial.

Recall that packages should be created in the `src` directory, not the root of the workspace. Navigate into `ros2_ws/src` and create a new package:

```
ros2 pkg create --build-type ament_python --license Apache-2.0 py_srvcli --dependencies rclpy
example_interfaces
```

Your terminal will return a message verifying the creation of your package `py_srvcli` and all its necessary files and folders.

The `--dependencies` argument will automatically add the necessary dependency lines to `package.xml`. `example_interfaces` is the package that includes the .srv file you will need to structure your requests and responses:

```
int64 a
int64 b
---
int64 sum
```

The first two lines are the parameters of the request, and below the dashes is the response.

1.1 Update `package.xml`

Because you used the `--dependencies` option during package creation, you don't have to manually add dependencies to `package.xml`.

As always, though, make sure to add the description, maintainer email and name, and license information to `package.xml`.

```
<description>Python client server tutorial</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

1.2 Update `setup.py`

Add the same information to the `setup.py` file for the `maintainer`, `maintainer_email`, `description` and `license` fields:

```
maintainer='Your Name',
maintainer_email='you@email.com',
description='Python client server tutorial',
license='Apache License 2.0',
```

2 Write the service node

Inside the `ros2_ws/src/py_srvcli/py_srvcli` directory, create a new file called `service_member_function.py` and paste the following code within:

```
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node


class MinimalService(Node):

    def __init__(self):
        super().__init__('minimal_service')
        self.srv = self.create_service(AddTwoInts, 'add_two_ints', self.add_two_ints_callback)

    def add_two_ints_callback(self, request, response):
        response.sum = request.a + request.b
        self.get_logger().info('Incoming request\na: %d b: %d' % (request.a, request.b))

        return response


def main():
    rclpy.init()

    minimal_service = MinimalService()

    rclpy.spin(minimal_service)

    rclpy.shutdown()


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

2.1 Examine the code

The first `import` statement imports the `AddTwoInts` service type from the `example_interfaces` package. The following `import` statement imports the ROS 2 Python client library, and specifically the `Node` class.

```
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node
```

The `MinimalService` class constructor initializes the node with the name `minimal_service`. Then, it creates a service and defines the type, name, and callback.

```
def __init__(self):
    super().__init__('minimal_service')
    self.srv = self.create_service(AddTwoInts, 'add_two_ints', self.add_two_ints_callback)
```

The definition of the service callback receives the request data, sums it, and returns the sum as a response.

```
def add_two_ints_callback(self, request, response):
    response.sum = request.a + request.b
    self.get_logger().info('Incoming request\na: %d b: %d' % (request.a, request.b))

    return response
```

Finally, the main class initializes the ROS 2 Python client library, instantiates the `MinimalService` class to create the service node and spins the node to handle callbacks.

2.2 Add an entry point

To allow the `ros2 run` command to run your node, you must add the entry point to `setup.py` (located in the `ros2_ws/src/py_srvcli` directory).

Add the following line between the `'console_scripts':` brackets:

```
'service = py_srvcli.service_member_function:main',
```

3 Write the client node

Inside the `ros2_ws/src/py_srvcli/py_srvcli` directory, create a new file called `client_member_function.py` and paste the following code within:

```
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node


class MinimalClientAsync(Node):

    def __init__(self):
        super().__init__('minimal_client_async')
        self.cli = self.create_client(AddTwoInts, 'add_two_ints')
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')
        self.req = AddTwoInts.Request()

    def send_request(self, a, b):
        self.req.a = a
        self.req.b = b
        self.future = self.cli.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()


def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    response = minimal_client.send_request(int(sys.argv[1]), int(sys.argv[2]))
    minimal_client.get_logger().info(
        'Result of add_two_ints: for %d + %d = %d' %
        (int(sys.argv[1]), int(sys.argv[2]), response.sum))

    minimal_client.destroy_node()
    rclpy.shutdown()


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

3.1 Examine the code

As with the service code, we first `import` the necessary libraries.

```
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node
```

The `MinimalClientAsync` class constructor initializes the node with the name `minimal_client_async`. The constructor definition creates a client with the same type and name as the service node. The type and name must match for the client and service to be able to communicate. The `while` loop in the constructor checks if a service matching the type and name of the client is available once a second. Finally it creates a new `AddTwoInts` request object.

```
def __init__(self):
    super().__init__('minimal_client_async')
    self.cli = self.create_client(AddTwoInts, 'add_two_ints')
    while not self.cli.wait_for_service(timeout_sec=1.0):
        self.get_logger().info('service not available, waiting again...')
    self.req = AddTwoInts.Request()
```

Below the constructor is the `send_request` method, which will send the request and spin until it receives the response or fails.

```
def send_request(self, a, b):
    self.req.a = a
    self.req.b = b
    self.future = self.cli.call_async(self.req)
    rclpy.spin_until_future_complete(self, self.future)
    return self.future.result()
```

Finally we have the `main` method, which constructs a `MinimalClientAsync` object, sends the request using the passed-in command-line arguments, and logs the results.

```
def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    response = minimal_client.send_request(int(sys.argv[1]), int(sys.argv[2]))
    minimal_client.get_logger().info(
        'Result of add_two_ints: for %d + %d = %d' %
        (int(sys.argv[1]), int(sys.argv[2]), response.sum))

    minimal_client.destroy_node()
    rclpy.shutdown()
```

3.2 Add an entry point

Like the service node, you also have to add an entry point to be able to run the client node.

The `entry_points` field of your `setup.py` file should look like this:

```
entry_points={
    'console_scripts': [
        'service = py_srvcli.service_member_function:main',
        'client = py_srvcli.client_member_function:main',
    ],
},
```

4 Build and run

It's good practice to run `rosdep` in the root of your workspace (`ros2_ws`) to check for missing dependencies before building:

Linux | macOS | Windows

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

Navigate back to the root of your workspace, `ros2_ws`, and build your new package:

```
colcon build --packages-select py_srvcli
```

Open a new terminal, navigate to `ros2_ws`, and source the setup files:

Linux | macOS | Windows

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

Now run the service node:

```
ros2 run py_srvcli service
```

The node will wait for the client's request.

Open another terminal and source the setup files from inside `ros2_ws` again. Start the client node, followed by any two integers separated by a space:

```
ros2 run py_srvcli client 2 3
```

If you chose `2` and `3`, for example, the client would receive a response like this:

```
[INFO] [minimal_client_async]: Result of add_two_ints: for 2 + 3 = 5
```

Return to the terminal where your service node is running. You will see that it published log messages when it received the request:

```
[INFO] [minimal_service]: Incoming request
a: 2 b: 3
```

Enter `Ctrl+C` in the server terminal to stop the node from spinning.

Summary

You created two nodes to request and respond to data over a service. You added their dependencies and executables to the package configuration files so that you could build and run them, allowing you to see a service/client system at work.

Next steps

In the last few tutorials you've been utilizing interfaces to pass data across topics and services. Next, you'll learn how to create custom interfaces.

Related content

- There are several ways you could write a service and client in Python; check out the `minimal_client` and `minimal_service` packages in the ros2/examples repo.

- In this tutorial, you used the `call_async()` API in your client node to call the service. There is another service call API available for Python called synchronous calls. We do not recommend using synchronous calls, but if you'd like to learn more about them, read the guide to Synchronous vs. asynchronous clients.