Dr. Eng. Essa Alghannam

ROS Custom Services

✓ Example 1:

~/catkin_ws/src/

- **my_python_pkg/**
  - ❖ srv/
    - ✓ AddTwoInts.srv
  - ❖ src/
    - ✓ server.py
    - ✓ client.py
  - ❖ include
  - ❖ CMakeLists.txt
  - ❖ package.xml

✓ 1. Create the Service Definition (AddTwoInts.srv)

my_python_pkg/srv/AddTwoInts.srv

```
$ roscd my_python_pkg
$ mkdir srv
$ cd srv
$ gedit AddTwoInts.srv
```

```
# Request fields

int64 a

int64 b
```

```
---
# Response fields
int64 sum
```

✓ 2. Create the Service Server (server.py)

server.py

In the src directory of my_python_pkg

```
$ roscd my_python_pkg/src
$ gedit server.py
```

```
#!/usr/bin/env python
import rospy
from my_python_pkg.srv import *
# from my_python_pkg.srv import AddTwoInts, AddTwoIntsResponse


def handle_add_two_ints(req):
    response =AddTwoIntsResponse()
    print(f"Received request: {req.a} + {req.b}")
    response.sum= req.a + req.b
    print(f"Returning sum: {response.sum}")

    # Return the response object
    return response

if __name__ == "__main__":
    # Initialize the ROS node
```

Commented [EA5]: Shebang line to make the script executable

Commented [EA6]: Imports the service definition (AddTwoInts) and the response message class (AddTwoIntsResponse) generated from the .srv file.

```
rospy.init_node('add_two_ints_server')
# Advertise the service
# Arguments: service_name, service_type, handler_function
rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)

print("Ready to add two integers service.")

# Keep the node running
rospy.spin()
```

Commented [EA7]: The name this service will be known by on the ROS graph.

Commented [EA8]: The service type (defined by the .srv file).

Commented [EA9]: The Python function to call when a request is received.

Commented [EA10]: Enters a loop that keeps the node alive and processes callbacks (like service requests) until the node is shut down.

```
chmod +x server.py
```

✓ 3. Create the Service Client (client.py)

client.py

in the src directory of my_python_pkg

```
$ roscd my_python_pkg/src
$ gedit client.py
```

```
#!/usr/bin/env python
import rospy
import sys # Used for accessing command line arguments

from my_python_pkg.srv import *
def add_two_ints_client(x, y):
    # Wait for the service to become available
```

```
    rospy.wait_for_service('add_two_ints')


    try:
        # Arguments: service_name, service_type
        addtwoints = rospy.ServiceProxy('add_two_ints', AddTwoInts)


        # Call the handle/proxy of the service “addtwoints ” with the request
arguments
        # The arguments match the field names in the .srv file (a and b)
        resp = addtwoints (x, y)


        # Process the response
        return resp.sum


    except rospy.ServiceException as e:
        # Handle potential errors:
        print(f"Service call failed: {e}")
        return None # Or raise the exception


if __name__ == '__main__':
    rospy.init_node('add_two_ints_client', anonymous=True)


    # Get arguments from the command line
    if len(sys.argv) == 3:
```

Commented [EA11]: It pauses the script until the service named add_two_ints is available on the ROS network.

Create a handle/proxy to the service :Commented [EA12] Creates a client object (a "proxy") that we can use to call the service. we provide the service name and the service type.

Commented [EA13]: This line *calls* the service. We have to pass the arguments x and y, which correspond to the request fields a and b in .srv file

Commented [EA14]: the sum field from the response object

Commented [EA15]: Catches errors that might occur if the service call fails (e.g., the server crashed, or the service wasn't found even after waiting)

```
    a = int(sys.argv[1])
    b = int(sys.argv[2])
else:
    print("Usage: add_two_ints_client.py a b")
    sys.exit(1) # Exit with an error code

print(f"Requesting {a} + {b}")
result = add_two_ints_client(a, b)

if result is not None:
    print(f"Result: {a} + {b} = {result}")
```

```
chmod +x client.py
```

✓ 4. Update CMakeLists.txt and package.xml

In package.xml:

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

In CMakeLists.txt:

a) Find the find_package call and add message_generation:

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation # Add this line
)
```

b) **Add .srv file to the add_service_files section:**

```
# Add here srv files, each file must be listed separately
add_service_files(
  FILES
  AddTwoInts.srv # Add this line for new service file
)
```

c) **Make sure generate_messages is called:**

```
# Generate services, actions and messages
generate_messages(
  DEPENDENCIES
  std_msgs # Add dependencies if the .srv uses other message types
)
```

d) **Ensure scripts are installed (optional but good practice):**

```
catkin_install_python(PROGRAMS
  src/server.py
  src/client.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

✓ 5. Build and Run

a) Source workspace to make the new executables available: source devel/setup.bash

b) Open a terminal in ROS workspace and build the package using: catkin_make

c) Open two terminals and run the server and client:

- Terminal 1: roscore
- **Terminal 2(Server):**

  rosrun **my_python_pkg** server.py

output:

```
^Cessa@essa:~/mycatkin_ws$ rosrun my_python_pkg server.py
Ready to add two integers service.
Received request: 5 + 7
Returning sum: 12
Received request: 5 + 8
Returning sum: 13
```

- **Terminal 3 (Client):**

  rosrun **my_python_pkg** client.py 5 7

```
essa@essa:~/mycatkin_ws$ rosrun my_python_pkg client.py 5 7
Requesting 5 + 7
Result: 5 + 7 = 12
essa@essa:~/mycatkin_ws$ rosrun my_python_pkg client.py 5 8
Requesting 5 + 8
Result: 5 + 8 = 13
essa@essa:~/mycatkin_ws$
```

---

rosservice show AddTwoInts

rosservice call /AddTwoInts 1 5

rosrun my_python_pkg client.py 1 6