Robot Operating System (ROS)

Lab 3: Custom Messages and ROS Services

Haitham El-Hussieny, PhD

December 5, 2022

Department of Mechatronics and Robotics Engineering
Egypt-Japan University of Science and Technology (E-JUST)
Alexandria, Egypt.

Outline

Create Your Own Custom ROS Message.

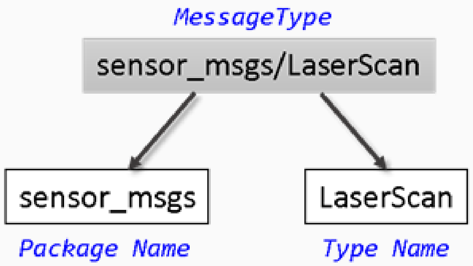
CREATE YOUR OWN CUSTOM ROS MESSAGE.

ROS Message Structure:

package_name/message_type

Examples:

- std_msgs/String
- geometry_msgs/Twist



ROS alllows to create your own messages with different fields.

Create Your Own Custom ROS Message.

The message type has many types and fields:

```
$ rosmsg show geometry_msgs/Twist

geometry_msgs/Vector3 linear
    float64  x
    float64  y
    float64  z

geometry_msgs/Vector3 angular
    float64  x
    float64  y
    float64  z
```

Create Your Own Custom ROS Message.

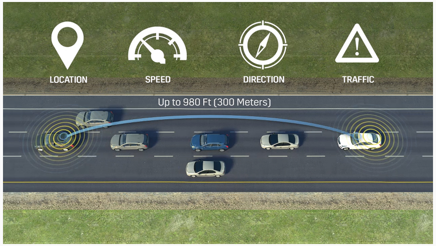
The message type has many types and fields:

```
$ rosmsg show geometry_msgs/Twist

geometry_msgs/Vector3 linear
    float64  x
    float64  y
    float64  z

geometry_msgs/Vector3 angular
    float64  x
    float64  y
    float64  z
```

```
$ rosmsg show geometry_msgs/Transform

geometry_msgs/Vector3 translation
    float64  x
    float64  y
    float64  z

geometry_msgs/Quaternion rotation
    float64  x
    float64  y
    float64  z
    float64  w
```

Create Your Own Custom ROS Message.

Imagine you need to create your own custom message. For examples:

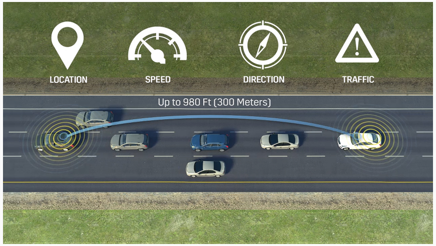


For V2V:

- Id
- location
- speed
- direction
- traffic

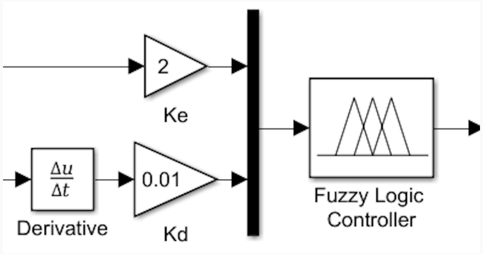
Create Your Own Custom ROS Message.

Imagine you need to create your own custom message. For examples:



For V2V:

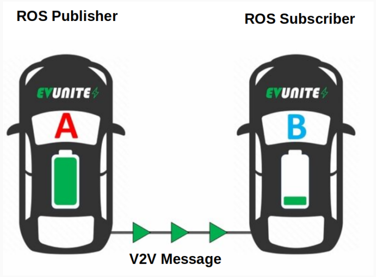
- Id
- location
- speed
- direction
- traffic



For Fuzzy controller:

- time
- error
- change of error

Create Your Own Custom ROS Message.

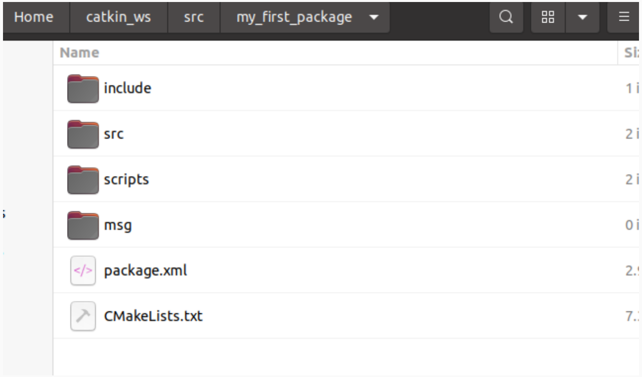


Our custom ROS message has the fields:

- int32 **id**
- string **name**
- float32 **battery_level**
- geometry_msgs/Pose2D **car_pose**
- geometry_msgs/Twist **car_speed**

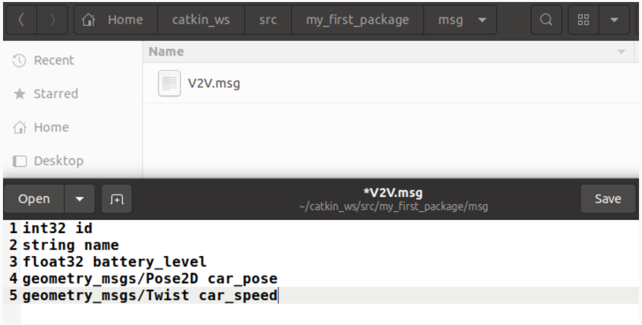
Create Your Own Custom ROS Message.

1. Create a msg folder in your package



Create Your Own Custom ROS Message.

2. Create a file with *.msg extension and add types and fields.



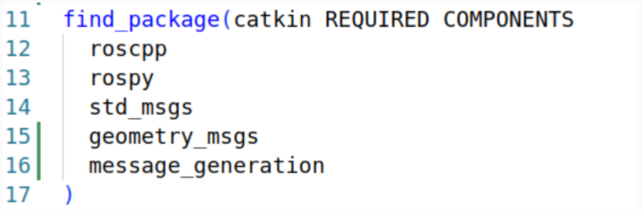
```
int32 id
string name
float32 battery_level
geometry_msgs/Pose2D car_pose
geometry_msgs/Twist car_speed
```

in terminal:

```
$ gedit ~/catkin_ws/src/my_first_package/msg/V2V.msg
```

Create Your Own Custom ROS Message.

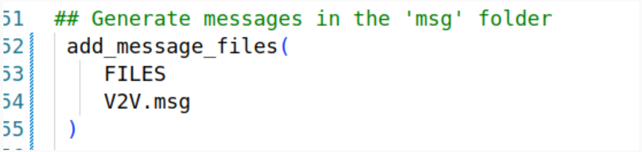
3. Update dependencies in CMakeLists.txt



```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  geometry_msgs
  message_generation
)
```

Create Your Own Custom ROS Message.

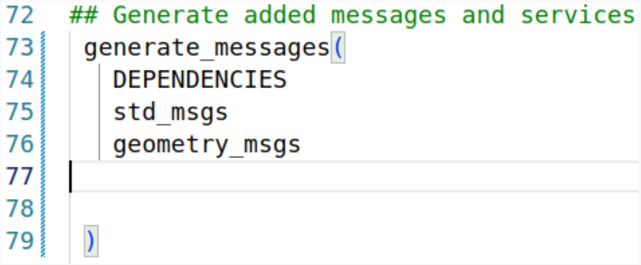
3. Update dependencies in CMakeLists.txt



```
## Generate messages in the 'msg' folder
add_message_files(
   FILES
   V2V.msg
)
```

Create Your Own Custom ROS Message.

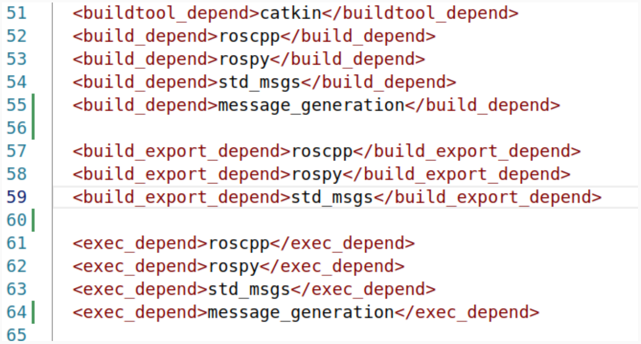
3. Update dependencies in CMakeLists.txt



```
## Generate added messages and services
generate_messages(
  DEPENDENCIES
  std_msgs
  geometry_msgs


)
```

Create Your Own Custom ROS Message.

4. Update dependencies in package.xml



```xml
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>

<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>

<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>message_generation</exec_depend>
```

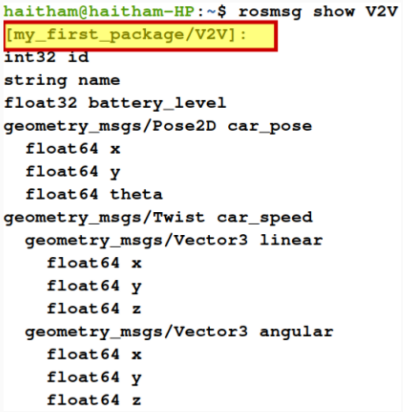
Create Your Own Custom ROS Message.

5. Catkin_make

```
~/catkin_ws$ catkin_make
```

6. Show your message

```
$ rosmsg show V2V
```



```
haitham@haitham-HP:~$ rosmsg show V2V
[my_first_package/V2V]:
int32 id
string name
float32 battery_level
geometry_msgs/Pose2D car_pose
  float64 x
  float64 y
  float64 theta
geometry_msgs/Twist car_speed
  geometry_msgs/Vector3 linear
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
```

Create Your Own Custom ROS Message.

To use your custom message in any other nodes:

talker.py

```
from my_first_package.msg import V2V

pub = rospy.Publisher('v2v_topic', V2V, queue_size=10)

my_car_info = V2V() #define your message name
my_car_info.battery_level = 0.95
my_car_info.id = 1223
my_car_info.car_pose.x = 5
my_car_info.car_speed.linear.x = 30
...
...
pub.publish(my_car_info)
```

Don't forget to add the message package 'my_first_package' as dependencies in CMakeLists.txt and package.xml of **the other package** that will use your custom message.

ROS Services.

ROS Services.

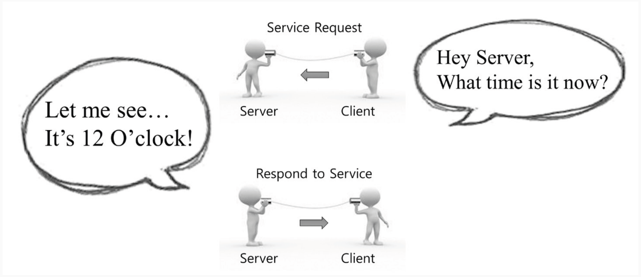
ROS Services consist of:

- ROS Server.
- ROS Client.

When to use ROS services?

When ask a robot to do a task and wait until finish. Examples:

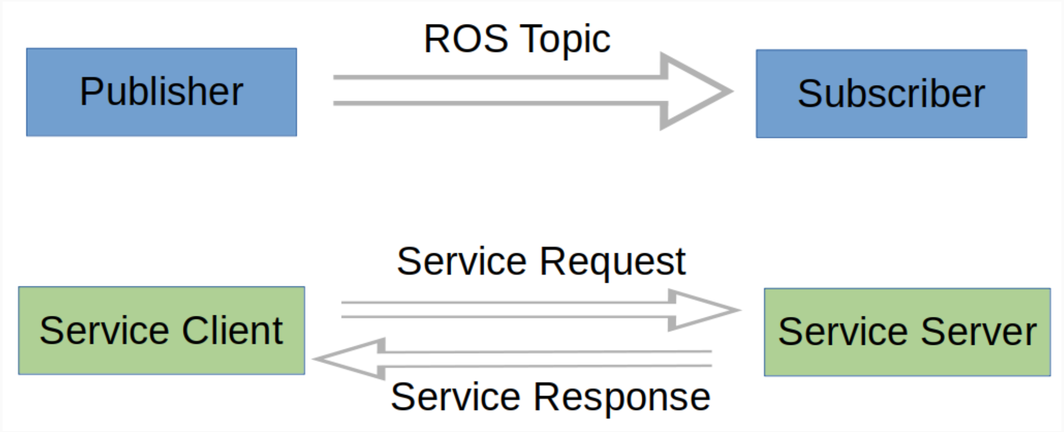
- Find a path from A to B.
- Open the door.
- etc.



A client sends a **request** and the server responds with a **response** (Synchronous communication).

ROS Services.

Communication over topics vs. communication over services.



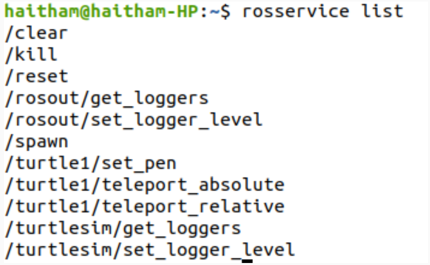
ROS Services. (Turtlesim ROS Services:)

run turtlesim_node

```
$ roscore
$ rosrun turtlesim turtlesim_node
```

show the services list

```
$ rosservice list
```

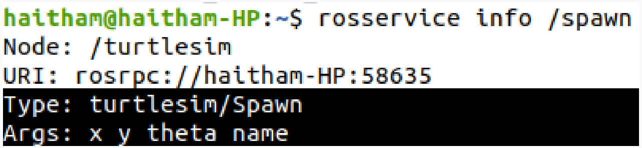


```
haitham@haitham-HP:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

ROS Services. (Turtlesim ROS Services:)

get info about spawn service

```
$ rosservice info /spawn
```

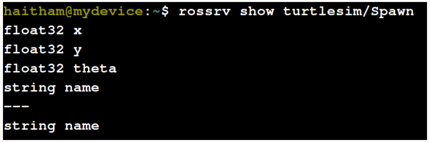


```
haitham@haitham-HP:~$ rosservice info /spawn
Node: /turtlesim
URI: rosrpc://haitham-HP:58635
Type: turtlesim/Spawn
Args: x y theta name
```

This service creates a new turtle inside the simulator. The type of the request/response message is turtlesim/Spawn with arguments: x, y, theta and name.

get info

```
$ rossrv show turtlesim/Spawn
```



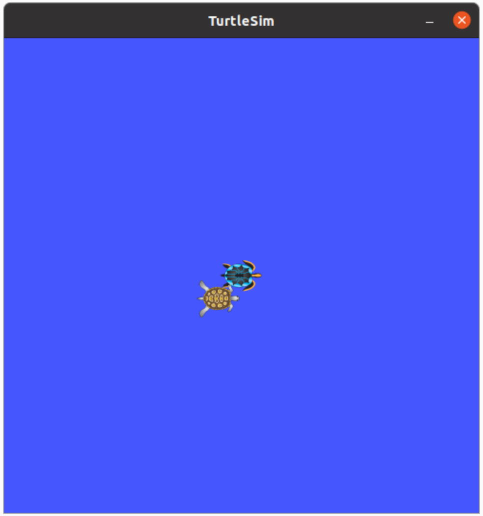
```
haitham@mydevice:~$ rossrv show turtlesim/Spawn
float32 x
float32 y
float32 theta
string name
---
string name
```

ROS Services.

call the spawn service

```
$ rosservice call /spwan 5 5 0 turtle2


    name: "turtle2"
```

The client asks to insert a new turtle at location 5,5 and angle 0 with the name 'turtle2'. The server replies with name of the new turtle.

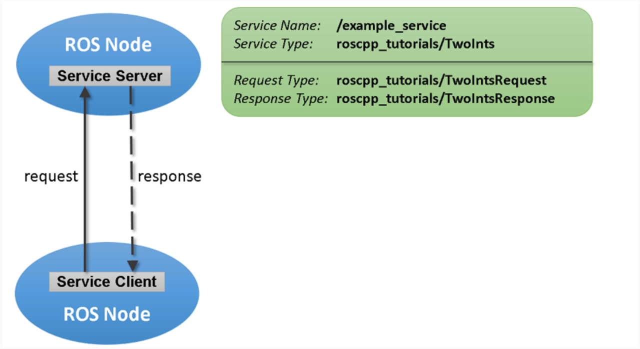


Create Custom ROS Services.

Create Custom ROS Services.

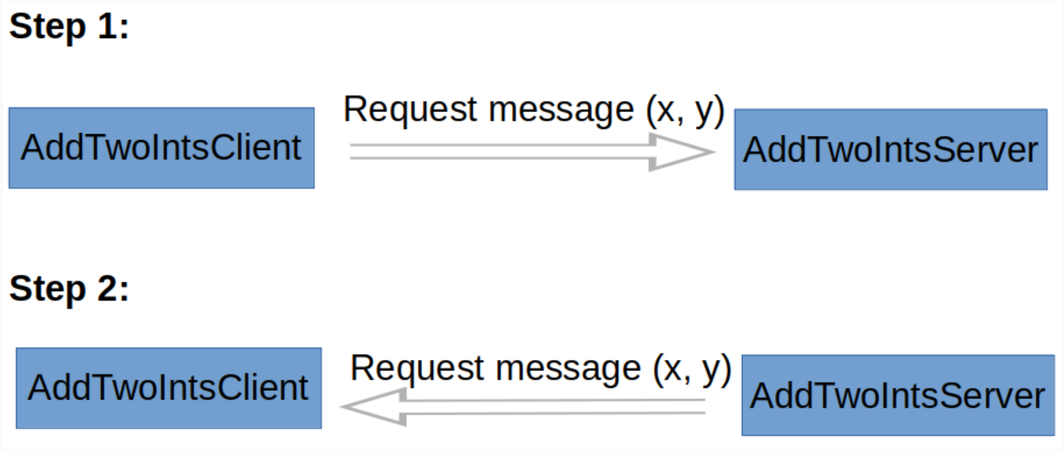
To create a client/server ROS service:

1. Define the service message (service file).
2. Create the ROS server node.
3. Create the ROS client node.
4. Execute the service server.
5. Call the server by the client.



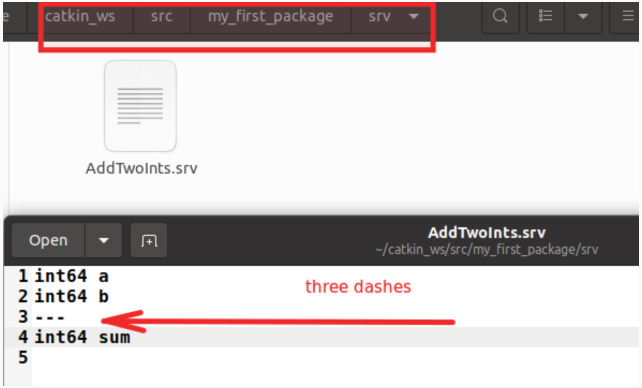
Service Name:	/example_service
Service Type:	roscpp_tutorials/TwoInts
Request Type:	roscpp_tutorials/TwoIntsRequest
Response Type:	roscpp_tutorials/TwoIntsResponse

Create Custom ROS Services.



STEP 1: DEFINE THE SERVICE MESSAGE (SERVICE FILE).

In your ROS package folder create a **srv** folder with AddTwoInts.srv file.



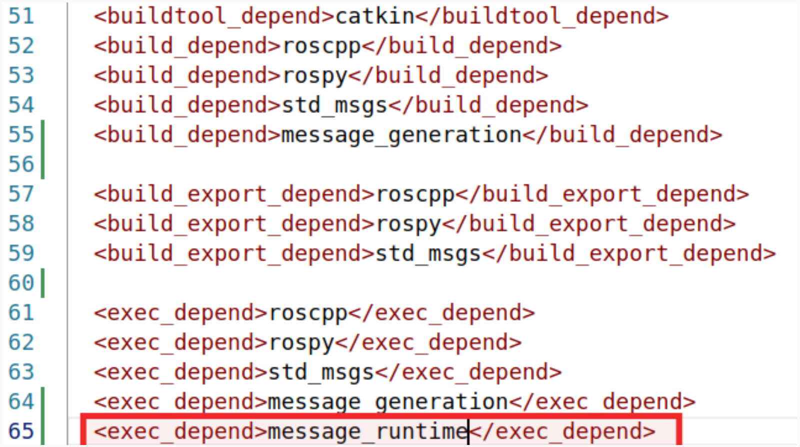
```
int64 a
int64 b
---
int64 sum
```

in terminal

```
$ gedit ~/catkin_ws/src/my_first_package/srv/AddTwoInts.srv
```

Step 1: Define the service message (service file).

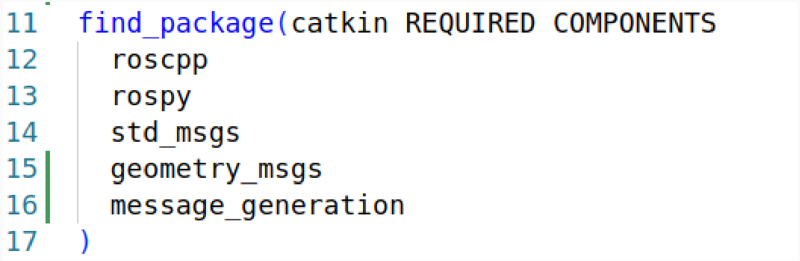
Update the Dependencies in **package.xml** file



```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>

<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>

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

Step 1: Define the service message (service file).

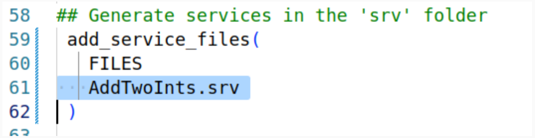
Make sure the Dependencies in **CMakeLists.txt** file are defined.



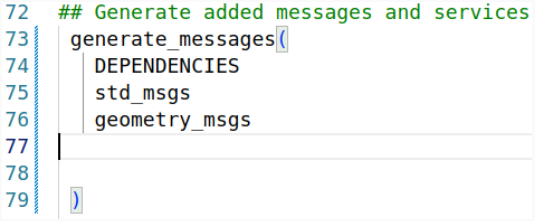
```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  geometry_msgs
  message_generation
)
```

Step 1: Define the service message (service file).

Update the Dependencies in **CMakeLists.txt** file and add the srv file.



```
## Generate services in the 'srv' folder
add_service_files(
  FILES
  AddTwoInts.srv
)
```



```
## Generate added messages and services
generate_messages(
  DEPENDENCIES
  std_msgs
  geometry_msgs


)
```

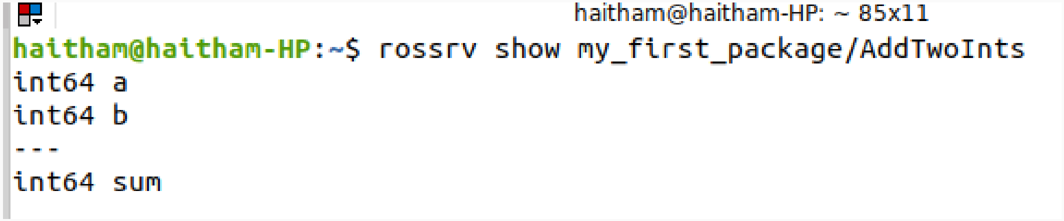
catkin make your workspace

```
$ ~/catkin_ws/catkin_make
```

Step 1: Define the service message (service file).

Verify the service is created

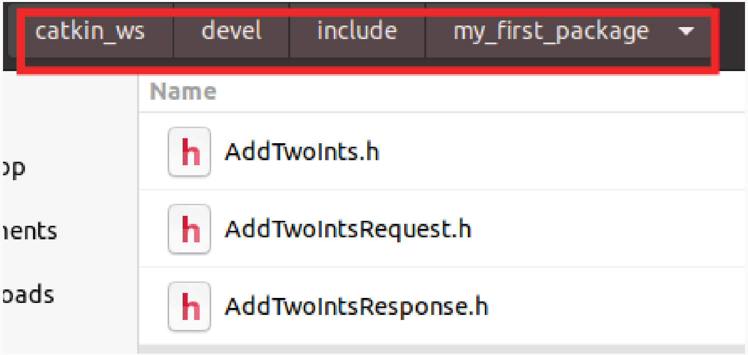
```
$ rossrv show my_first_package/AddTwoInts
```



```
haitham@haitham-HP: ~ 85x11
haitham@haitham-HP:~$ rossrv show my_first_package/AddTwoInts
int64 a
int64 b
---
int64 sum
```

Step 1: Define the service message (service file).

Verify the auto-generated service header files are created



AddTwoInts.h

AddTwoIntsRequest.h

AddTwoIntsResponse.h

Step 2: Create a ROS server node in Python.

my_first_package/scripts/add_server.py

```
from my_first_package.srv import AddTwoInts
from my_first_package.srv import AddTwoIntsRequest
from my_first_package.srv import AddTwoIntsResponse
import time
import rospy

def handle_add_two_ints(req):
    print("Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b)))
    time.sleep(5) # 5 seconds delay
    sum_response = AddTwoIntsResponse(req.a + req.b)
    return sum_response

def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print("Ready to add two ints.")
    rospy.spin()

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

Step 2: Create a ROS server node in Python.

Explanation of the code:

1. import the service definitions

```
from my_first_package.srv import AddTwoInts
from my_first_package.srv import AddTwoIntsRequest
from my_first_package.srv import AddTwoIntsResponse
import time
import rospy
```

Step 2: Create a ROS server node in Python.

Explanation of the code:

2. create the server node

```
def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print("Ready to add two ints.")
    rospy.spin()
```

- **init_node**: create the server node with a name.
- **rospy.Service()**: defines the service name, request type, handler of a request.

Step 2: Create a ROS server node in Python.

Explanation of the code:

3. create the request handler

```
def handle_add_two_ints(req):
    print("Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b)))
    time.sleep(5) # 5 seconds delay
    sum_response = AddTwoIntsResponse(req.a + req.b)
    return sum_response
```

- access the request variables: a and b.
- create a variable of type **AddTwoIntsResponse**.
- return the response.

Step 3: Create a ROS client node in Python.

my_first_package/scripts/add_client.py

```python
#!/usr/bin/env python

import sys
import rospy
from my_first_package.srv import AddTwoInts
from my_first_package.srv import AddTwoIntsRequest
from my_first_package.srv import AddTwoIntsResponse

def add_two_ints_client(x, y):
    rospy.wait_for_service('add_two_ints')
    try:
        add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        sys.exit(1)
    print("Requesting %s+%s"%(x, y))
    print("%s + %s = %s"%(x, y, add_two_ints_client(x, y)))
```

Step 3: Create a ROS client node in Python.

Explanation of the code:

1. input the two values from the terminal

```
if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        sys.exit(1)
    print("Requesting %s+%s"%(x, y))
    print("%s + %s = %s"%(x, y, add_two_ints_client(x, y)))
```

Step 3: Create a ROS client node in Python.

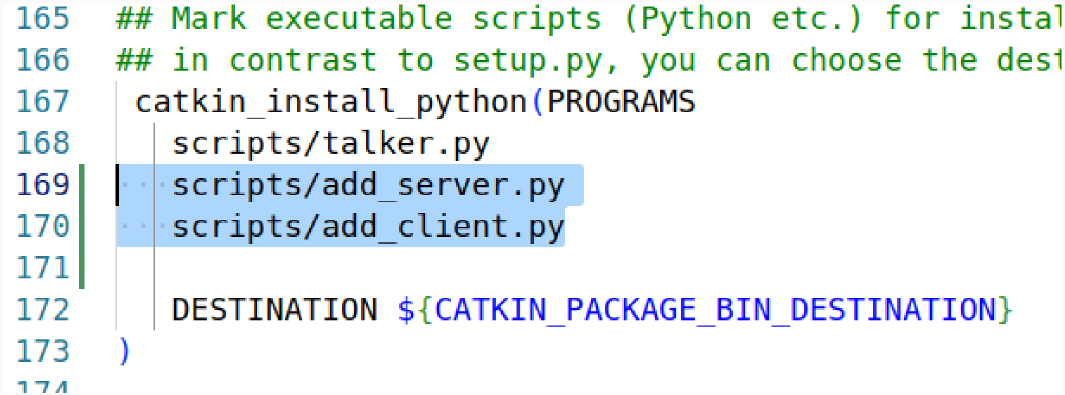
Explanation of the code:

2. call the server

```
def add_two_ints_client(x, y):
    rospy.wait_for_service('add_two_ints')
    try:
        add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)
```

- **wait_for_service()** blocks until the service is available.
- **rospy.ServiceProxy()** creates a handle for calling the service named add_two_ints.

Modify the CMakeLists.txt file.



```
## Mark executable scripts (Python etc.) for instal
## in contrast to setup.py, you can choose the desi
catkin_install_python(PROGRAMS
    scripts/talker.py
    scripts/add_server.py
    scripts/add_client.py

    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Step 4: Run the server and client services.

1. catkin make your workspace

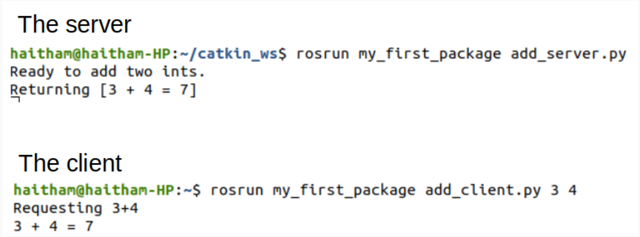
```
$ ~/catkin_ws/catkin_make
```

2. change *.py files to executable

```
$ chmod a+x add_server.py
$ chmod a+x add_client.py
```

3. run the nodes

```
$ rosrun my_first_package add_server.py
$ rosrun my_first_package add_client.py
```



The server

```
haitham@haitham-HP:~/catkin_ws$ rosrun my_first_package add_server.py
Ready to add two ints.
Returning [3 + 4 = 7]
```

The client

```
haitham@haitham-HP:~$ rosrun my_first_package add_client.py 3 4
Requesting 3+4
3 + 4 = 7
```

Assignment: A Service for Controlling Pioneer P3DX in VREP,

Step 5: Create a ROS server node in C++. (Optional)

my_first_package/src/add_server.cpp

```
#include "ros/ros.h"
#include "my_first_package/AddTwoInts.h"

bool add(my_first_package::AddTwoInts::Request  &req, beginner_tutorials::AddTwoInts::Response &res)
{
    res.sum = req.a + req.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: [%ld]", (long int)res.sum);
    return true;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_server");
    ros::NodeHandle n;

    ros::ServiceServer service = n.advertiseService("add_two_ints", add);
    ROS_INFO("Ready to add two ints.");
    ros::spin();

    return 0;
}
```

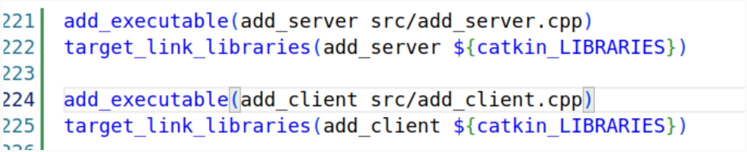
STEP 6: CREATE A ROS CLIENT NODE IN C++. (OPTIONAL)

my_first_package/src/add_client.cpp

```
#include "ros/ros.h"
#include "my_first_package/AddTwoInts.h"
#include <cstdlib>
int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_client");
    if (argc != 3)
    {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }
    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<my_first_package::AddTwoInts>("add_two_ints");
    beginner_tutorials::AddTwoInts srv;
    srv.request.a = atoll(argv[1]);
    srv.request.b = atoll(argv[2]);
    if (client.call(srv))
    {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else
    {ROS_ERROR("Failed to call service add_two_ints");return 1;}

    return 0;
}
```

Step 7: Modify the CMakeLists.txt file.

Add the two source files to be executable.



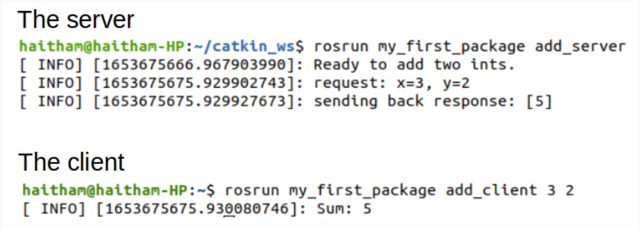
```
add_executable(add_server src/add_server.cpp)
target_link_libraries(add_server ${catkin_LIBRARIES})

add_executable(add_client src/add_client.cpp)
target_link_libraries(add_client ${catkin_LIBRARIES})
```

catkin make

```
$ ~/catkin_ws/catkin_make
```

run the nodes

```
$ rosrun my_first_package
add_server
$ rosrun my_first_package
add_client
```



The server

```
haitham@haitham-HP:~/catkin_ws$ rosrun my_first_package add_server
[ INFO] [1653675666.967903990]: Ready to add two ints.
[ INFO] [1653675675.929902743]: request: x=3, y=2
[ INFO] [1653675675.929927673]: sending back response: [5]
```

The client

```
haitham@haitham-HP:~$ rosrun my_first_package add_client 3 2
[ INFO] [1653675675.930080746]: Sum: 5
```

End of Lecture

haitham.elhussieny@ejust.edu.eg