ROS: Services & Actions

Services

- Remote function calls
- One node calls a function that executes in another node
- **Client:** The node that calls the service. The service is accessed through a local proxy
- **Server:** The node that executes the service. Contains a callback for the service request.

Defining a Service

- A service definition is similar to a message definition.
- It contains 2 parts: The message type of the **request** and the message type of the **response.**
- A service definition is saved in the ***srv*** folder in the package as a ***.srv*** file.

Example: Service definition

Example from Morgan Quigley

Definition file *(**WordCount.srv**)* for a service that counts the number of words in a request string:

```
string words
---
uint32 count
```

The first line is the request type
The ***' ---'*** separates the request type and the response type
The last line is the response type

Required Modifications in CMakeLists.txt

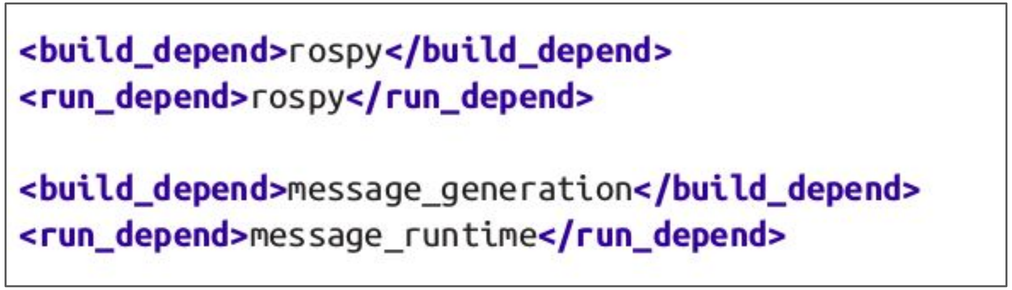
```
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   message_generation   # Add message_generation here, after the other packages
)
```

```
add_service_files(
  FILES
  WordCount.srv
)
```

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

Required Modifications in Package.xml

Add the following build-time dependencies and runtime dependencies if they are not added already.



```
<build_depend>rospy</build_depend>
<run_depend>rospy</run_depend>

<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

Compiling the service

From the workspace, run `catkin_make`.

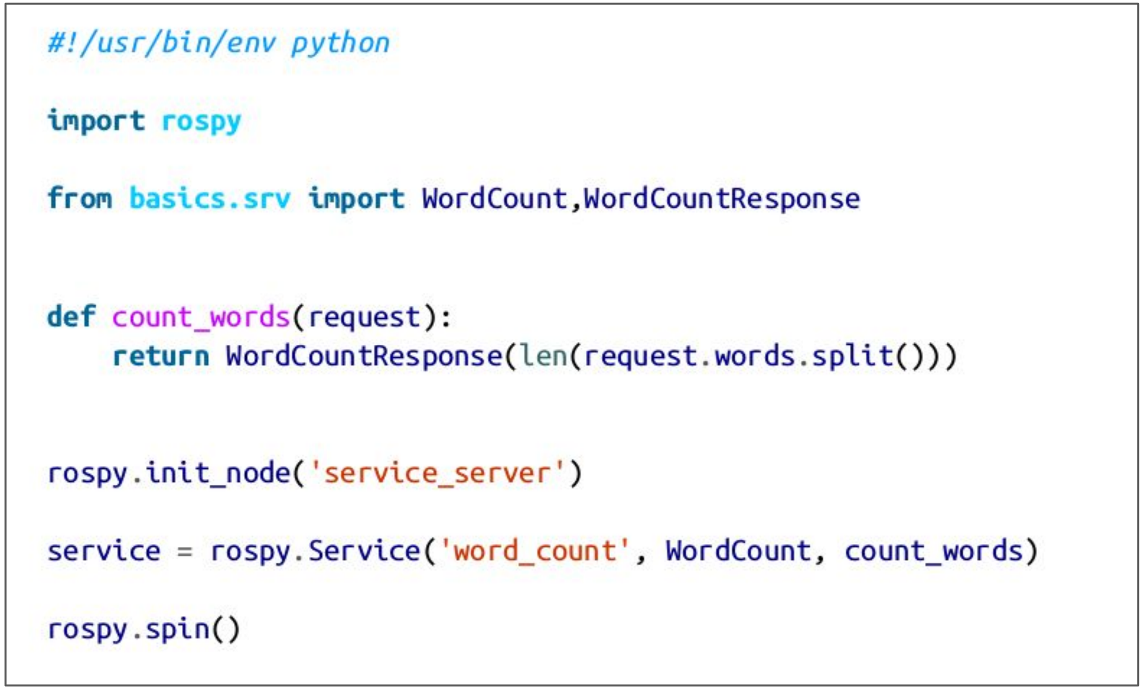
This will generate a library *<package_name>*.srv containing three class definitions

1. WordCount
2. WordCountRequest
3. WordCountResponse

Check if the service has been compiled by running `rossrv show WordCount`

Implementing a Service

Example from Morgan Quigley Page-55 (4.3 service_server.py)



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

import rospy

from basics.srv import WordCount,WordCountResponse


def count_words(request):
    return WordCountResponse(len(request.words.split()))


rospy.init_node('service_server')

service = rospy.Service('word_count', WordCount, count_words)

rospy.spin()
```

Writing a Service Client

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

import rospy

from basics.srv import WordCount

import sys


rospy.init_node('service_client')

rospy.wait_for_service('word_count')

word_counter = rospy.ServiceProxy('word_count', WordCount)

words = ' '.join(sys.argv[1:])

word_count = word_counter(words)

print words, '->', word_count.count
```

Using the Service

Services can be called from command line, as well as from other nodes

1. Check if the service is compiled using `rosservice show <service_name>`

2. Run the service using rosrun or as a python script

3. Check if the service has been advertised using `rosservice list`

4. In a new terminal, run the client or run the following command:
   `rosservice call <service_name> <arguments>`

Examples of Services

- A node that resets parameters of a sensor
- A node that takes a high resolution image using the onboard camera
- A node that plans a new path to a goalpoint when called

?

When would you use a service and when would you use a normal function?

ROS Actions

- Similar to services, but **long-running** and **goal-oriented**
- Consist of a Request, a **Result** and a **Goal.**
- Use **feedback** to provide updates on the progress towards the goal
- Allow the goals to be cancelled
- Can run parallely while the rest of the code executes

Defining an Action

Action definitions are saved in the ***actions*** folder alongside src, msg, srv, etc.

Example: Definition of a 'Timer' action
Save the following as ***Timer.action*** in the actions folder

```
duration time_to_wait
---
duration time_elapsed
uint32 updates_sent
---
duration time_elspsed
duration time_remaining
```

Goal

Result

Feedback

Changes in CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  # other packages are already listed here
  actionlib_msgs
)
```

```
add_action_files(
  DIRECTORY action
  FILES Timer.action
)
```

```
generate_messages(
  DEPENDENCIES
  actionlib_msgs
  std_msgs
)
```

```
catkin_package(
  CATKIN_DEPENDS
  actionlib_msgs
)
```

Once the changes are done, run `catkin_make`

Post-compilation

Catkin uses the information in the action file to produce 7 new message types

1. TimerAction.msg
2. TimerActionGoal.msg
3. TimerActionResult.msg
4. TimerActionFeedback.msg
5. TimerGoal.msg
6. TimerResult.msg
7. TimerFeedback.msg

Implementing an Action Server

Example from Morgan Quigley Ex 5-2 (Page 64)

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

import time
import actionlib
from basics.msg import TimerAction, TimerGoal, TimerResult

def do_timer(goal):
    start_time = time.time()
    time.sleep(goal.time_to_wait.to_sec())
    result = TimerResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = 0
    server.set_succeeded(result)

rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```

Implementing an Action Client

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

import actionlib
from basics.msg import TimerAction, TimerGoal, TimerResult

rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()
goal = TimerGoal()
goal.time_to_wait = rospy.Duration.from_sec(5.0)
client.send_goal(goal)
client.wait_for_result()
print('Time elapsed: %f'%(client.get_result().time_elapsed.to_sec()))
```

Examples of Actions

- Telling a robot to go to a particular goal point
- Scan a given area until a particular April tag is found
- Timers
- Docking tasks
- Exploration tasks

Reference Material on Actions

1. Morgan Quigley Chapter 5
   (Read and Implement all Codes)

2. ROS Wiki Tutorials:
   http://wiki.ros.org/actionlib/Tutorials

Task -

(Will be provided by Friday)