

ROS 기본 프로그래밍

ROBOTIS

Open Source Team

Yoonseok Pyo

자~ 이번에는

Topic: 퍼블리셔, 서비스크라이버

Service: 서비스 서버, 서비스 클라이언트

작성성에 들어갑니다.

Index

I. 메시지, 토픽, 서비스, 매개변수

II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

IV. 매개변수 사용법

V. roslaunch 사용법

Index

I. 메시지, 토픽, 서비스, 매개변수

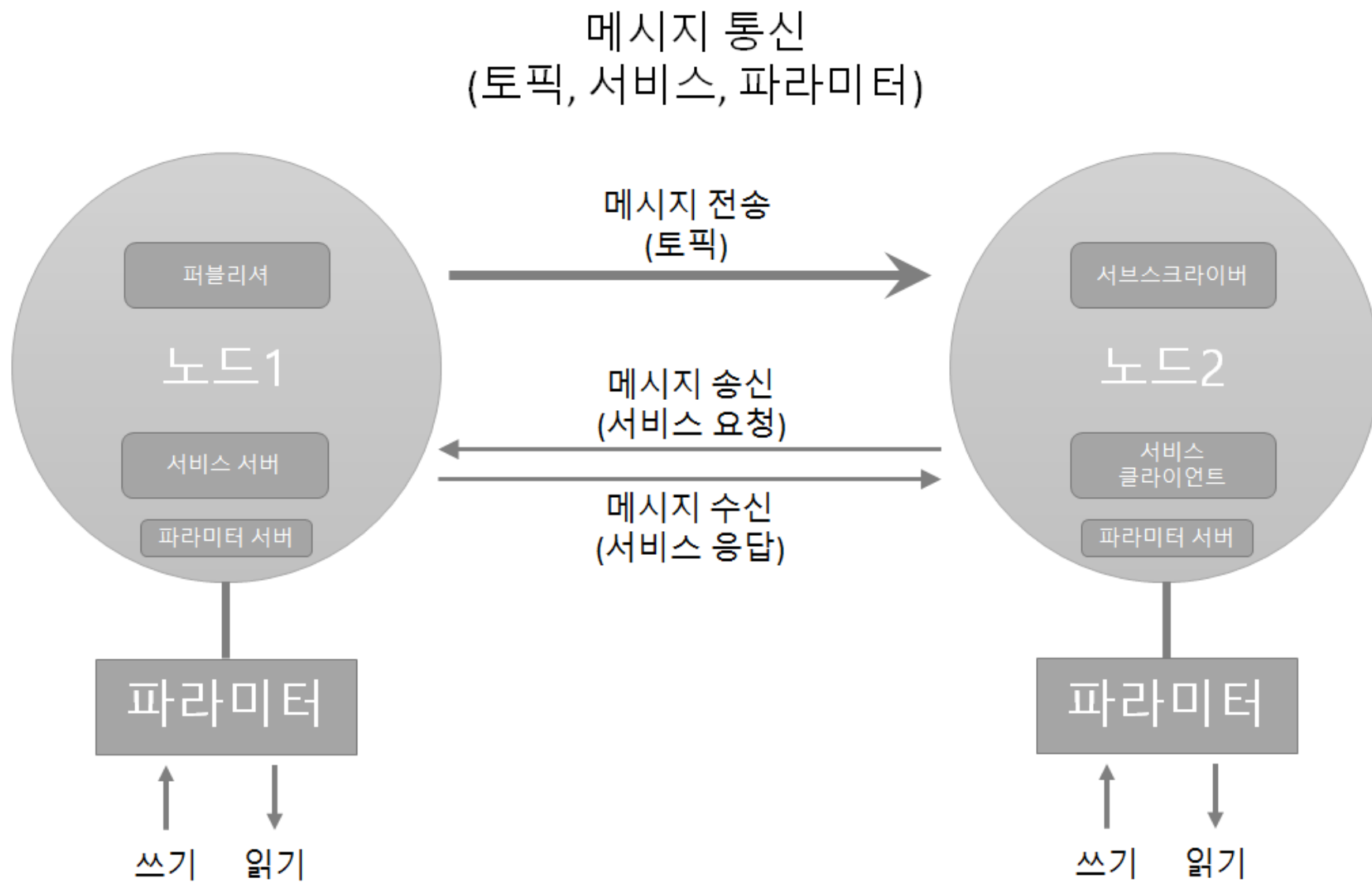
II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

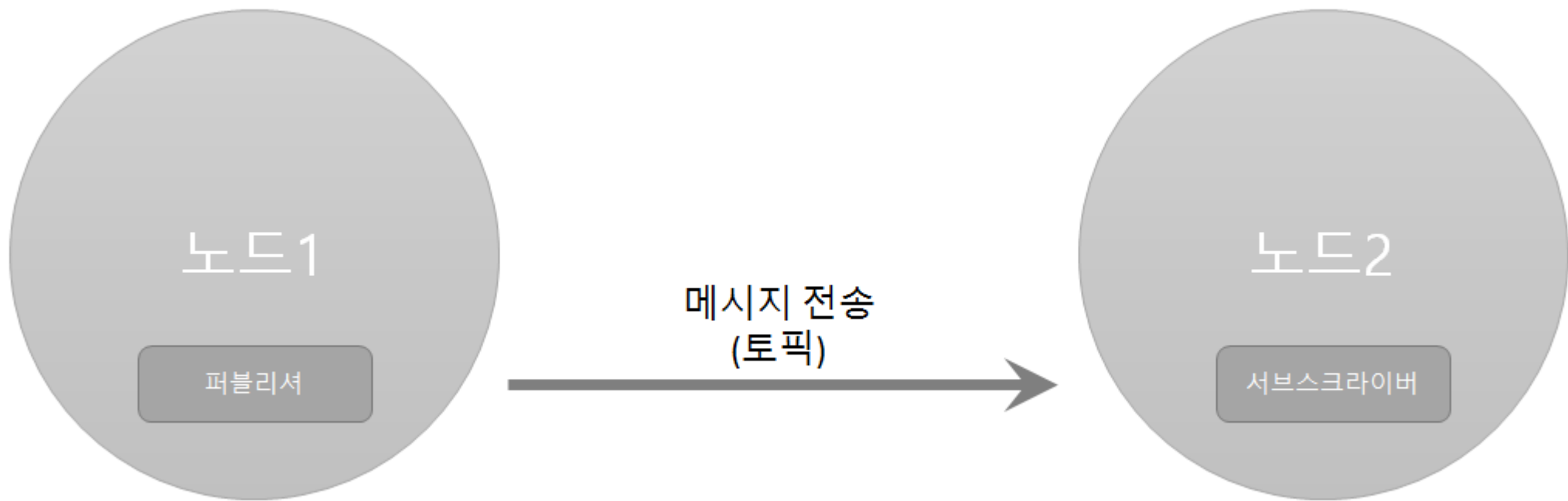
IV. 매개변수 사용법

V. roslaunch 사용법

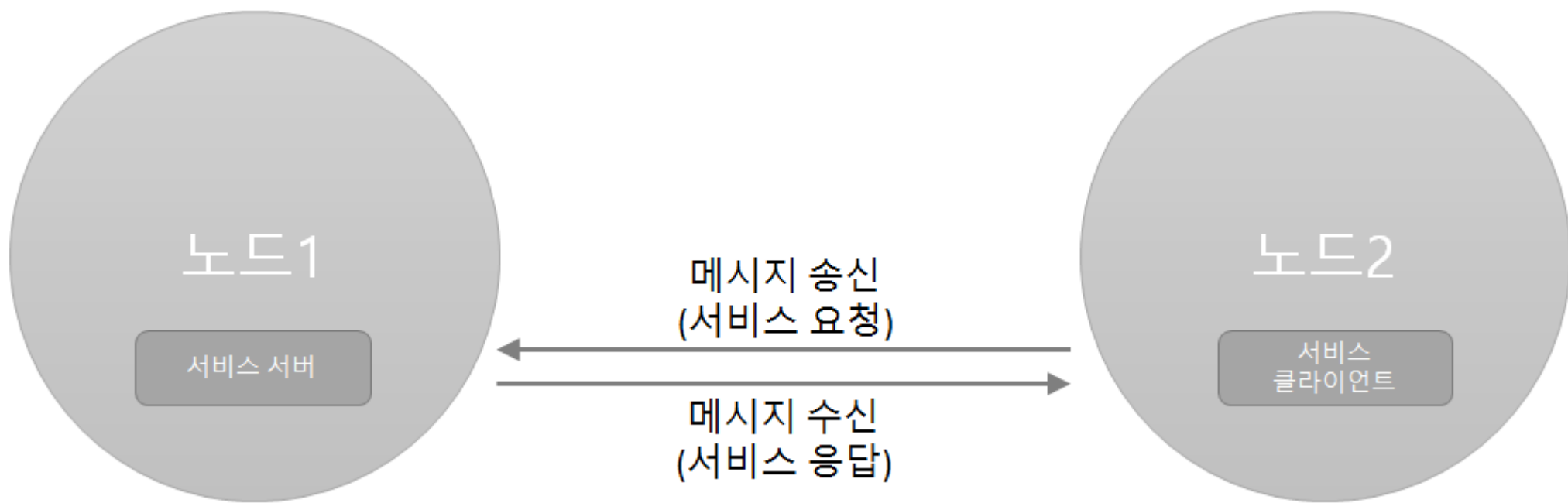
ROS 메시지 통신



토픽(Topic)



서비스(Service)



매개변수(Parameter)



Index

I. 메시지, 토픽, 서비스, 매개변수

II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

IV. 매개변수 사용법

V. roslaunch 사용법

Topic / Publisher / Subscriber

- ROS에서는 단방향 통신일때 'Topic' 이라는 메시지 통신을 사용한다. 이때 송신 측을 'Publisher', 수신 측을 'Subscriber'라고 부른다.

1) 패키지 생성

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg oroca_ros_tutorials std_msgs roscpp
```

```
$ cd ~/catkin_ws/src/oroca_ros_tutorials  
$ ls  
include      → 헤더 파일 폴더  
src          → 소스 코드 폴더  
CMakeLists.txt → 빌드 설정 파일  
package.xml  → 패키지 설정 파일
```

Topic / Publisher / Subscriber

2) 패키지 설정 파일(package.xml) 수정

- ROS의 필수 설정 파일 중 하나인 package.xml은 패키지 정보를 담은 XML 파일로서 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다.

```
$ gedit package.xml
```

```
<?xml version="1.0"?>
<package>
  <name>oroکا_ros_tutorials</name>
  <version>0.1.0</version>
  <description>The oroکا_ros_tutorials package</description>

  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="website">http://oroکا.org</url>
  <url type="repository">https://github.com/oroکا/oroکا_ros_tutorials.git</url>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>

  <license>BSD</license>
```

Topic / Publisher / Subscriber

```
<buildtool_depend>catkin</buildtool_depend>
```

```
<build_depend>roscpp</build_depend>
```

```
<build_depend>std_msgs</build_depend>
```

```
<build_depend>message_generation</build_depend>
```

```
<run_depend>roscpp</run_depend>
```

```
<run_depend>std_msgs</run_depend>
```

```
<run_depend>message_runtime</run_depend>
```

```
<export></export>
```

```
</package>
```

Topic / Publisher / Subscriber

3) 빌드 설정 파일(CMakeLists.txt) 수정

```
$ gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)
project(oroce_ros_tutorials)
## 캐킨 빌드를 할 때 요구되는 구성요소 패키지이다.
## 의존성 패키지로 roscpp, std_msgs, message_generation이며 이 패키지들이 존재하지 않으면 빌드 도중에 에러가 난다.
find_package(catkin REQUIRED COMPONENTS roscpp std_msgs message_generation)
## 메시지 선언: msgTutorial.msg
add_message_files(FILES msgTutorial.msg)
## 의존하는 메시지를 설정하는 옵션이다.
## std_msgs가 설치되어 있지 않다면 빌드 도중에 에러가 난다.
generate_messages(DEPENDENCIES std_msgs)
## 캐킨 패키지 옵션으로 라이브러리, 캐킨 빌드 의존성, 시스템 의존 패키지를 기술한다.
catkin_package(
  #INCLUDE_DIRS include
  LIBRARIES oroce_ros_tutorials
  CATKIN_DEPENDS roscpp std_msgs
  DEPENDS system_lib
)
```

Topic / Publisher / Subscriber

인클루드 디렉터리를 설정한다.

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

ros_tutorial_msg_publisher 노드에 대한 빌드 옵션이다.

실행 파일, 타겟 링크 라이브러리, 추가 의존성 등을 설정한다.

```
add_executable(ros_tutorial_msg_publisher src/ros_tutorial_msg_publisher.cpp)
```

```
target_link_libraries(ros_tutorial_msg_publisher ${catkin_LIBRARIES})
```

```
add_dependencies(ros_tutorial_msg_publisher oroca_ros_tutorials_generate_messages_cpp)
```

ros_tutorial_msg_subscriber 노드에 대한 빌드 옵션이다.

```
add_executable(ros_tutorial_msg_subscriber src/ros_tutorial_msg_subscriber.cpp)
```

```
target_link_libraries(ros_tutorial_msg_subscriber ${catkin_LIBRARIES})
```

```
add_dependencies(ros_tutorial_msg_subscriber oroca_ros_tutorials_generate_messages_cpp)
```

Topic / Publisher / Subscriber

4) 메시지 파일 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 옵션을 넣었다.

```
add_message_files(FILES msgTutorial.msg)
```

- 노드에서 사용할 메시지인 msgTutorial.msg를 빌드할 때 포함하라는 이야기

\$ roscd oroca_ros_tutorials	→ 패키지 폴더로 이동
\$ mkdir msg	→ 패키지에 msg라는 메시지 폴더를 신규 작성
\$ cd msg	→ 작성한 msg 폴더로 이동
\$ gedit msgTutorial.msg	→ msgTutorial.msg 파일 신규 작성 및 내용 수정

- int32 (메시지 형식), data (메시지 이름)
- 메시지 타입은 int32 이외에도 bool, int8, int16, float32, string, time, duration 등의 메시지 기본 타입과 ROS 에서 많이 사용되는 메시지를 모아놓은 common_msgs 등도 있다. 여기서는 간단한 예제를 만들어 보기 위한 것으로 int32를 이용하였다. (부록C 및 <http://wiki.ros.org/msg> 를 참고 할 것!)

```
int32 data
```

Topic / Publisher / Subscriber

5) 퍼블리셔 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(ros_tutorial_msg_publisher src/ros_tutorial_msg_publisher.cpp)
```

- src 폴더의 ros_tutorial_msg_publisher.cpp라는 파일을 빌드하여 ros_tutorial_msg_publisher라는 실행 파일을 만들라는 이야기

```
$ roscd oroca_ros_tutorials/src      → 패키지의 소스 폴더인 src 폴더로 이동  
$ gedit ros_tutorial_msg_publisher.cpp → 소스 파일 신규 작성 및 내용 수정
```

```
#include "ros/ros.h"           // ROS 기본 헤더파일  
#include "oroca_ros_tutorials/msgTutorial.h" // msgTutorial 메시지 파일 헤더 (빌드후 자동 생성됨)  
  
int main(int argc, char **argv) // 노드 메인 함수  
{  
    ros::init(argc, argv, "ros_tutorial_msg_publisher"); // 노드명 초기화  
    ros::NodeHandle nh; // ROS 시스템과 통신을 위한 노드 핸들 선언
```


Topic / Publisher / Subscriber

```
// 퍼블리셔 선언, oroca_ros_tutorials 패키지의 msgTutorial 메시지 파일을 이용한
// 퍼블리셔 ros_tutorial_pub 를 작성한다. 토픽명은 " ros_tutorial_msg " 이며,
// 퍼블리셔의 큐(queue) 사이즈를 100개로 설정한다는 것이다
ros::Publisher ros_tutorial_pub = nh.advertise<oroca_ros_tutorials::msgTutorial>("ros_tutorial_msg", 100);
ros::Rate loop_rate(10); // 루프 주기를 설정한다. "10" 이라는 것은 10Hz를 말하는 것으로 0.1초 간격으로 반복된다
int count = 0;           // 메시지에 사용될 변수 선언

while (ros::ok())
{
    oroca_ros_tutorials::msgTutorial msg; // msgTutorial 메시지 파일 형식으로 msg 라는 메시지를 선언
    msg.data = count; // count 라는 변수를 이용하여 메시지 값을 정한다
    ROS_INFO("send msg = %d", count); // ROS_INFO 라는 ROS 함수를 이용하여 count 변수를 표시한다
    ros_tutorial_pub.publish(msg); // 메시지를 발행한다. 약 0.1초 간격으로 발행된다
    loop_rate.sleep(); // 위에서 정한 루프 주기에 따라 슬립에 들어간다
    ++count; // count 변수 1씩 증가
}
return 0;
}
```

Topic / Publisher / Subscriber

6) 서브스크라이버 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(ros_tutorial_msg_subscriber src/ros_tutorial_msg_subscriber.cpp)
```

- 즉, ros_tutorial_msg_subscriber.cpp라는 파일을 빌드하여
ros_tutorial_msg_subscriber라는 실행 파일을 만들라는 이야기

```
$ roscd oroca_ros_tutorials/src      → 패키지의 소스 폴더인 src 폴더로 이동  
$ gedit ros_tutorial_msg_subscriber.cpp → 소스 파일 신규 작성 및 내용 수정
```

```
#include "ros/ros.h"                // ROS 기본 헤더파일  
#include "oroca_ros_tutorials/msgTutorial.h" // msgTutorial 메시지 파일 헤더 (빌드 후 자동 생성됨)  
  
// 메시지 콜백함수로써, ros_tutorial_sub 구독자에 해당되는 메시지를 수신하였을 때 동작하는 함수이다  
// 입력 메시지는 oroca_ros_tutorial 패키지의 msgTutorial 메시지를 받도록 되어있다  
void msgCallback(const oroca_ros_tutorials::msgTutorial::ConstPtr& msg)  
{  
    ROS_INFO("recieve msg: %d", msg->data); // 수신된 메시지를 표시하는 함수  
}
```

Topic / Publisher / Subscriber

```
int main(int argc, char **argv)                // 노드 메인 함수
{
    ros::init(argc, argv, "ros_tutorial_msg_subscriber"); // 노드명 초기화

    ros::NodeHandle nh;                          // ROS 시스템과 통신을 위한 노드 핸들 선언

    // 서브스크라이버 선언, oroca_ros_tutorials 패키지의 msgTutorial 메시지 파일을 이용한
    // 서브스크라이버 ros_tutorial_sub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며,
    // 서브스크라이버의 큐(queue) 사이즈를 100개로 설정한다는 것이다
    ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);

    // 콜백함수 호출을 위한 함수로써, 메시지가 수신되기를 대기, 수신되었을 경우 콜백함수를 실행한다
    ros::spin();

    return 0;
}
```

Topic / Publisher / Subscriber

7) ROS 노드 빌드

- 다음 명령어로 oroca_ros_tutorials 패키지의 메시지 파일, 퍼블리셔 노드, 서브스 크라이버 노드를 빌드하자.

```
$ cd ~/catkin_ws → catkin 폴더로 이동  
$ catkin_make → catkin 빌드 실행
```

- [참고] 파일시스템

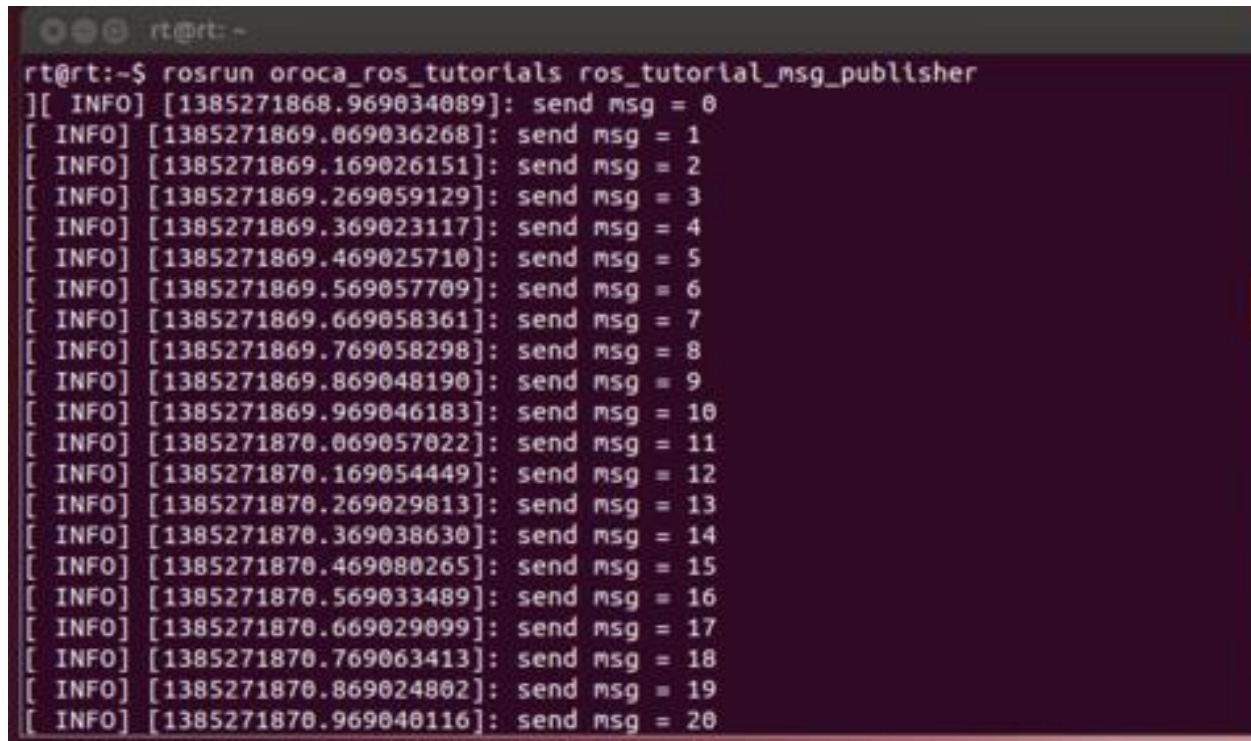
- oroca_ros_tutorials 패키지의 **소스 코드 파일**: ~/catkin_ws/src/oroca_ros_tutorials/src
- oroca_ros_tutorials 패키지의 **메시지 파일**: ~/catkin_ws/src/oroca_ros_tutorials/msg
- 빌드된 결과물은 /catkin_ws의 /build와 /devel 폴더에 각각 생성
 - /build 폴더에는 캐킨 빌드에서 사용된 **설정** 내용이 저장
 - /devel/lib/oroca_ros_tutorials 폴더에는 **실행 파일**이 저장
 - /devel/include/oroca_ros_tutorials 폴더에는 메시지 파일로부터 자동 생성된 **메시지 헤더 파일**이 저장

Topic / Publisher / Subscriber

8) 퍼블리셔 실행 [참고로 노드 실행에 앞서 roscore를 실행하는 것을 잊지 말자.]

- ROS 노드 실행 명령어인 rosrn을 이용하여, oroca_ros_tutorials 패키지의 ros_tutorial_msg_publisher 노드를 구동하라는 명령어

```
$ rosrn oroca_ros_tutorials ros_tutorial_msg_publisher
```



```
rt@rt:~$ rosrn oroca_ros_tutorials ros_tutorial_msg_publisher
[ INFO] [1385271868.969034089]: send msg = 0
[ INFO] [1385271869.069036268]: send msg = 1
[ INFO] [1385271869.169026151]: send msg = 2
[ INFO] [1385271869.269059129]: send msg = 3
[ INFO] [1385271869.369023117]: send msg = 4
[ INFO] [1385271869.469025710]: send msg = 5
[ INFO] [1385271869.569057709]: send msg = 6
[ INFO] [1385271869.669058361]: send msg = 7
[ INFO] [1385271869.769058298]: send msg = 8
[ INFO] [1385271869.869048190]: send msg = 9
[ INFO] [1385271869.969046183]: send msg = 10
[ INFO] [1385271870.069057022]: send msg = 11
[ INFO] [1385271870.169054449]: send msg = 12
[ INFO] [1385271870.269029813]: send msg = 13
[ INFO] [1385271870.369038630]: send msg = 14
[ INFO] [1385271870.469080265]: send msg = 15
[ INFO] [1385271870.569033489]: send msg = 16
[ INFO] [1385271870.669029099]: send msg = 17
[ INFO] [1385271870.769063413]: send msg = 18
[ INFO] [1385271870.869024802]: send msg = 19
[ INFO] [1385271870.969040116]: send msg = 20
```

Topic, Publisher, Subscriber

- [참고] rostopic

- rostopic 명령어를 이용하여 현재 ROS 네트워크에서 사용 중인 토픽 목록, 주기, 데이터 대역폭, 내용 확인 등이 가능하다.

```
$ rostopic list
/ros_tutorial_msg
/rosout
/rosout_agg
```

```
$ rostopic echo /ros_tutorial_msg
```

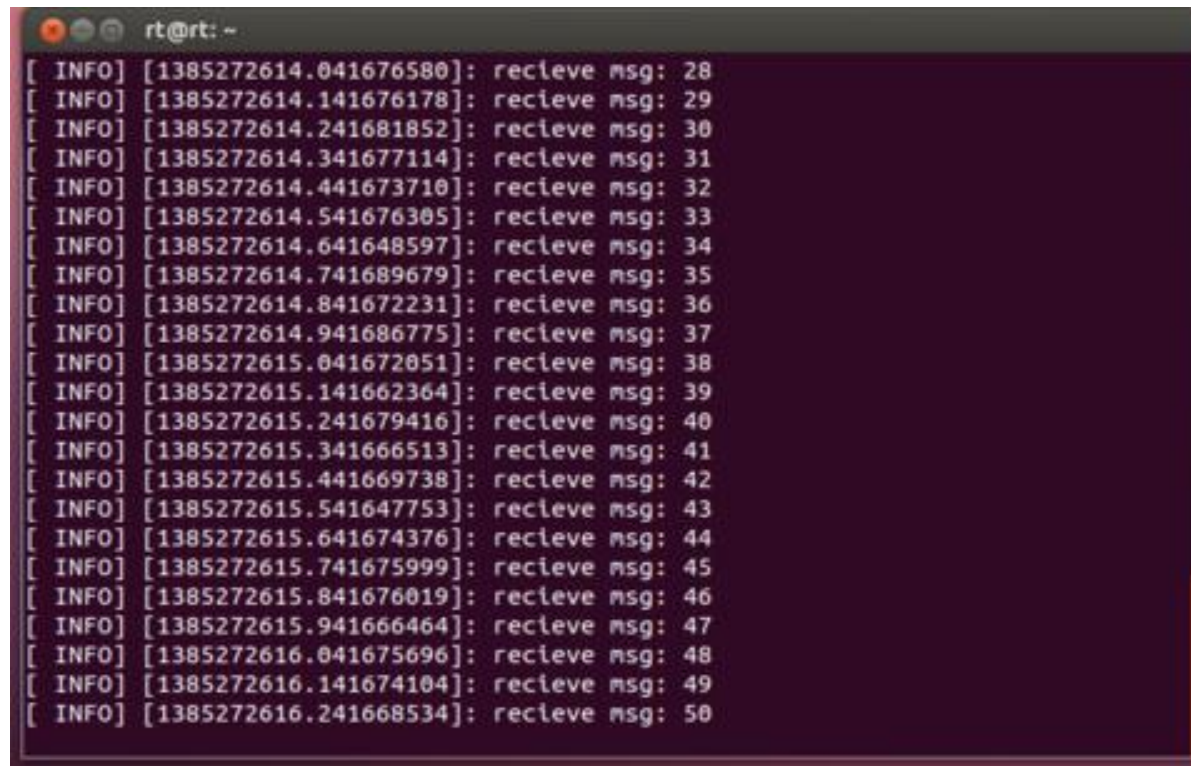
```
rt@rt: ~/catkin_ws/src/oroca_ros_tutorials/src
^Crt@rt:~/catkin_ws/src/oroca_ros_tutorials/src$ rostopic echo /ros_tutorial_msg
data: 2
---
data: 3
---
data: 4
---
data: 5
---
data: 6
---
data: 7
---
data: 8
---
data: 9
---
data: 10
---
data: 11
---
data: 12
---
```

Topic / Publisher / Subscriber

9) 서브스크라이버 실행

- ROS 노드 실행 명령어인 `roslaunch`를 이용하여, `oroca_ros_tutorials` 패키지의 `ros_tutorial_msg_subscriber` 노드를 구동하라는 명령어

```
$ roslaunch oroca_ros_tutorials ros_tutorial_msg_subscriber
```

A terminal window with a dark background and light-colored text. The window title is 'rt@rt: ~'. It displays a series of 23 lines of log output, each starting with '[INFO]' followed by a timestamp in brackets and the text 'recieve msg: ' followed by a number from 28 to 50. The text 'recieve' is misspelled as 'recieve' instead of 'receive'.

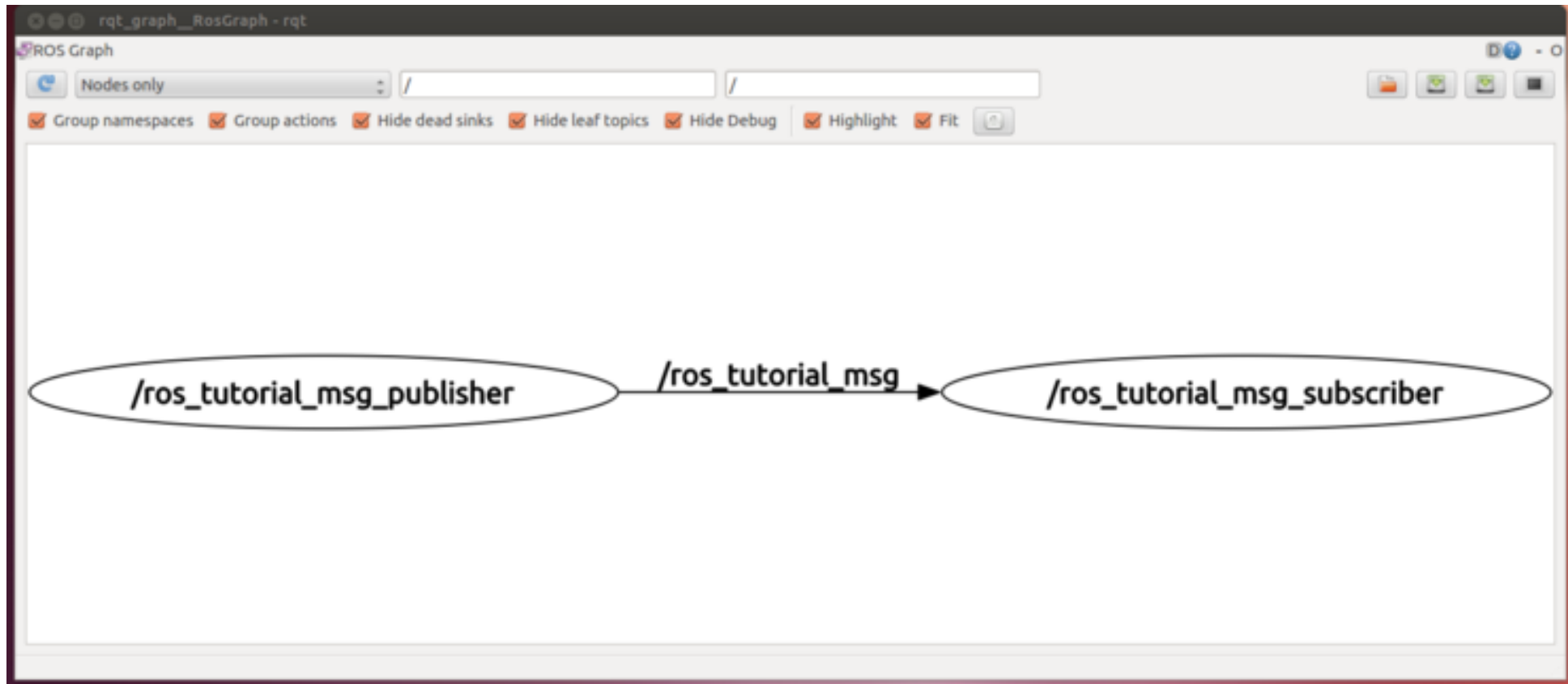
```
rt@rt: ~  
[ INFO] [1385272614.041676580]: recieve msg: 28  
[ INFO] [1385272614.141676178]: recieve msg: 29  
[ INFO] [1385272614.241681852]: recieve msg: 30  
[ INFO] [1385272614.341677114]: recieve msg: 31  
[ INFO] [1385272614.441673710]: recieve msg: 32  
[ INFO] [1385272614.541676305]: recieve msg: 33  
[ INFO] [1385272614.641648597]: recieve msg: 34  
[ INFO] [1385272614.741689679]: recieve msg: 35  
[ INFO] [1385272614.841672231]: recieve msg: 36  
[ INFO] [1385272614.941686775]: recieve msg: 37  
[ INFO] [1385272615.041672051]: recieve msg: 38  
[ INFO] [1385272615.141662364]: recieve msg: 39  
[ INFO] [1385272615.241679416]: recieve msg: 40  
[ INFO] [1385272615.341666513]: recieve msg: 41  
[ INFO] [1385272615.441669738]: recieve msg: 42  
[ INFO] [1385272615.541647753]: recieve msg: 43  
[ INFO] [1385272615.641674376]: recieve msg: 44  
[ INFO] [1385272615.741675999]: recieve msg: 45  
[ INFO] [1385272615.841676019]: recieve msg: 46  
[ INFO] [1385272615.941666464]: recieve msg: 47  
[ INFO] [1385272616.041675696]: recieve msg: 48  
[ INFO] [1385272616.141674104]: recieve msg: 49  
[ INFO] [1385272616.241668534]: recieve msg: 50
```

Topic / Publisher / Subscriber

10) 실행된 노드들의 통신 상태 확인

```
$ rqt_graph
```

```
$ rqt [플러그인(Plugins)] → [인트로스펙션(Introspection)] → [노드 그래프(Node Graph)]
```



Index

I. 메시지, 토픽, 서비스, 매개변수

II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

IV. 매개변수 사용법

V. roslaunch 사용법

Service / Service server / Service client

- 서비스는 요청(request)이 있을 때만 응답(response)하는 서비스 서버(service server)와 요청하고 응답받는 서비스 클라이언트(service client)로 나뉜다. 서비스는 토픽과는 달리 일회성 메시지 통신이다. 따라서 서비스의 요청과 응답이 완료되면 연결된 두 노드는 접속이 끊긴다.
- 이러한 서비스는 로봇에 특정 동작을 수행하도록 요청할 때에 명령어로서 많이 사용된다. 혹은 특정 조건에 따라 이벤트를 발생해야 할 노드에 사용된다. 또한, 일회성 통신 방식이라서 네트워크에 부하가 적기 때문에 토픽을 대체하는 수단으로도 사용되는 등 매우 유용한 통신수단이다.
- 이번 강의에서는 간단한 서비스 파일을 작성해보고, 서비스 서버(server) 노드와 서비스 클라이언트(client) 노드를 작성하고 실행하는 것을 목적으로 한다.
- 참고로 앞 절에서 이미 oroca_ros_tutorials라는 패키지를 생성하였으므로 패키지 생성은 생략하며, 패키지 정보를 담은 package.xml 파일도 수정할 부분이 없으므로 생략한다.

Service / Service server / Service client

1) 빌드 설정 파일(CMakeLists.txt) 수정

- 이전에 작성한 oroca_ros_tutorials 패키지에 새로운 서비스 서버 노드, 서비스 클라이언트 노드, 서비스 파일(*.srv)을 추가하기 위하여 다음 예제와 같이 oroca_ros_tutorials 패키지로 이동한 후에 CMakeLists.txt 파일을 수정하도록 하자.

```
$ roscd oroca_ros_tutorials  
$ gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)  
project(oroca_ros_tutorials)
```

```
## 캐킨 빌드를 할 때 요구되는 구성요소 패키지이다.
```

```
## 의존성 패키지로 roscpp, std_msgs, message_generation이며 이 패키지들이 존재하지 않으면 빌드  
도중에 에러가 난다.
```

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

Service / Service server / Service client

```
## 메시지 선언: msgTutorial.msg
## 서비스 선언: srvTutorial.srv
add_message_files(FILES msgTutorial.msg)
add_service_files(FILES srvTutorial.srv)

## 의존하는 메시지를 설정하는 옵션이다.
## std_msgs가 설치되어 있지 않다면 빌드 도중에 에러가 난다.
generate_messages(DEPENDENCIES std_msgs)

## 캐킨 패키지 옵션으로 라이브러리, 캐킨 빌드 의존성, 시스템 의존 패키지를 기술한다.
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES oroca_ros_tutorials
  CATKIN_DEPENDS roscpp std_msgs
  DEPENDS system_lib
)
```

Service / Service server / Service client

```
## 인클루드 디렉터리를 설정한다.
include_directories(include ${catkin_INCLUDE_DIRS})
## ros_tutorial_msg_publisher 노드에 대한 빌드 옵션이다.
## 실행 파일, 타겟 링크 라이브러리, 추가 의존성 등을 설정한다.
add_executable(ros_tutorial_msg_publisher src/ros_tutorial_msg_publisher.cpp)
target_link_libraries(ros_tutorial_msg_publisher ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_msg_publisher oroca_ros_tutorials_generate_messages_cpp)
## ros_tutorial_msg_subscriber 노드에 대한 빌드 옵션이다.
add_executable(ros_tutorial_msg_subscriber src/ros_tutorial_msg_subscriber.cpp)
target_link_libraries(ros_tutorial_msg_subscriber ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_msg_subscriber oroca_ros_tutorials_generate_messages_cpp)
# ros_tutorial_srv_server 노드에 대한 빌드 옵션이다.
add_executable(ros_tutorial_srv_server src/ros_tutorial_srv_server.cpp)
target_link_libraries(ros_tutorial_srv_server ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_srv_server oroca_ros_tutorials_generate_messages_cpp)
# ros_tutorial_srv_client 노드에 대한 빌드 옵션이다.
add_executable(ros_tutorial_srv_client src/ros_tutorial_srv_client.cpp)
target_link_libraries(ros_tutorial_srv_client ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_srv_client oroca_ros_tutorials_generate_messages_cpp)
```

Service / Service server / Service client

2) 서비스 파일 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 옵션을 넣었다.

```
add_service_files(FILES srvTutorial.srv)
```

- 노드에서 사용할 메시지인 srvTutorial.srv를 빌드할 때 포함하라는 이야기

```
$ roscd oroca_ros_tutorials      → 패키지 폴더로 이동
$ mkdir srv                     → 패키지에 srv라는 서비스 폴더를 신규 작성
$ cd srv                        → 작성한 srv 폴더로 이동
$ gedit srvTutorial.srv         → srvTutorial.srv 파일 신규 작성 및 내용 수정
```

- int64 (메시지 형식), a, b (서비스 요청: request), result (서비스 응답: response),
'---' (요청과 응답을 구분하는 구분자)

```
int64 a
int64 b
---
int64 result
```

Service / Service server / Service client

3) 서비스 서버 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(ros_tutorial_srv_server src/ros_tutorial_srv_server.cpp)
```

- src 폴더의 ros_tutorial_srv_server.cpp라는 파일을 빌드하여 ros_tutorial_srv_server라는 실행 파일을 만들라는 이야기

```
$ roscd oroca_ros_tutorials/src          → 패키지의 소스 폴더인 src 폴더로 이동  
$ gedit ros_tutorial_srv_server.cpp      → 소스 파일 신규 작성 및 내용 수정
```

```
#include "ros/ros.h"           // ROS 기본 헤더 파일  
#include "oroca_ros_tutorials/srvTutorial.h" // srvTutorial 서비스 파일 헤더
```

Service / Service server / Service client

```
// 서비스 요청이 있을 경우, 아래의 처리를 수행한다.  
// 서비스 요청은 req, 서비스 응답은 res로 설정하였다.  
bool calculation(orooca_ros_tutorials::srvTutorial::Request &req,  
                  orooca_ros_tutorials::srvTutorial::Response &res)  
{  
    // 서비스 요청시 받은 a와 b 값을 더하여 서비스 응답 값에 저장한다.  
    res.result = req.a + req.b;  
  
    // 서비스 요청에 사용된 a, b 값의 표시 및 서비스 응답에 해당되는 result 값을 출력한다.  
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);  
    ROS_INFO("sending back response: [%ld]", (long int)res.result);  
    return true;  
}
```


Service / Service server / Service client

```
int main(int argc, char **argv)                // 노드 메인 함수
{
    ros::init(argc, argv, "ros_tutorial_srv_server"); // 노드명 초기화
    ros::NodeHandle nh;                          // 노드 핸들 선언

    // 서비스 서버 선언,
    // oroca_ros_tutorials 패키지의 srvTutorial 서비스 파일을 이용한
    // 서비스 서버 ros_tutorial_service_server를 작성한다.
    // 서비스명은 ros_tutorial_srv이며 서비스 요청이 있을 때,
    // calculation라는 함수를 실행하라는 설정이다.
    ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv", calculation);
    ROS_INFO("ready srv server!");

    ros::spin();                                // 서비스 요청을 대기한다
    return 0;
}
```

Service / Service server / Service client

4) 서비스 클라이언트 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(ros_tutorial_srv_client src/ros_tutorial_srv_client.cpp)
```

- src 폴더의 ros_tutorial_srv_client.cpp라는 파일을 빌드하여 ros_tutorial_srv_client라는 실행 파일을 만들라는 이야기

```
$ roscd oroca_ros_tutorials/src
```

→ 패키지의 소스 폴더인 src 폴더로 이동

```
$ gedit ros_tutorial_srv_client.cpp
```

→ 소스 파일 신규 작성 및 내용 수정

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

// ROS 기본 헤더 파일

```
#include "oroca_ros_tutorials/srvTutorial.h"
```

// srvTutorial 서비스 파일 헤더

```
#include <cstdlib>
```

// atoi 함수 사용을 위한 라이브러리

Service / Service server / Service client

// 노드 메인 함수

```
int main(int argc, char **argv)
```

```
{
```

// 노드명 초기화

```
ros::init(argc, argv, "ros_tutorial_srv_client");
```

// 입력값 오류 처리

```
if (argc != 3)
```

```
{
```

```
    ROS_INFO("cmd : rosrunc ros_tutorial ros_tutorial_service_client arg0 arg1");
```

```
    ROS_INFO("arg0: double number, arg1: double number");
```

```
    return 1;
```

```
}
```

// ROS 시스템과 통신을 위한 노드 핸들 선언

```
ros::NodeHandle nh;
```

// 서비스 클라이언트 선언, oroca_ros_tutorials 패키지의 srvTutorial 서비스 파일을 이용한

// 서비스 클라이언트 ros_tutorial_service_client를 작성한다.

// 서비스명은 "ros_tutorial_srv"이다

```
ros::ServiceClient ros_tutorial_service_client = nh.serviceClient<oroca_ros_tutorials::srvTutorial>("ros_tutorial_srv");
```

Service / Service server / Service client

```
// srv라는 이름으로 srvTutorial 서비스 파일을 이용하는 서비스 파일을 선언한다.
roca_ros_tutorials::srvTutorial srv;
// 서비스 요청 값으로 노드가 실행될 때 입력으로 사용된 매개변수를 각각의 a, b에 저장한다.
srv.request.a = atoll(argv[1]);
srv.request.b = atoll(argv[2]);
// 서비스를 요청하고, 요청이 받아들여졌을 경우, 응답 값을 표시한다.
if (ros_tutorial_service_client.call(srv))
{
    ROS_INFO("send srv, srv.Request.a and b: %ld, %ld", (long int)srv.request.a, (long int)srv.request.b);
    ROS_INFO("receive srv, srv.Response.result: %ld", (long int)srv.response.result);
}
else
{
    ROS_ERROR("Failed to call service ros_tutorial_srv");
    return 1;
}
return 0;
}
```

Service / Service server / Service client

5) ROS 노드 빌드

- 다음 명령어로 oroca_ros_tutorials 패키지의 서비스 파일, 서비스 서버 노드와 클라이언트 노드를 빌드한다.

```
$ cd ~/catkin_ws && catkin_make → catkin 폴더로 이동 후 catkin 빌드 실행
```

• [참고] 파일시스템

- oroca_ros_tutorials 패키지의 **소스 코드 파일**: ~/catkin_ws/src/oroca_ros_tutorials/src
- oroca_ros_tutorials 패키지의 **메시지 파일**: ~/catkin_ws/src/oroca_ros_tutorials/msg
- 빌드된 결과물은 /catkin_ws의 /build와 /devel 폴더에 각각 생성
 - /build 폴더에는 캐킨 빌드에서 사용된 **설정** 내용이 저장
 - /devel/lib/oroca_ros_tutorials 폴더에는 **실행 파일**이 저장
 - /devel/include/oroca_ros_tutorials 폴더에는 메시지 파일로부터 자동 생성된 **메시지 헤더 파일**이 저장

Service / Service server / Service client

6) 서비스 서버 실행

[참고로 노드 실행에 앞서 roscore를 실행하는 것을 잊지 말자.]

- 서비스 서버는 서비스 요청이 있기 전까지 아무런 처리를 하지 않고 기다리도록 프로그래밍하였다. 그러므로 다음 명령어를 실행하면 서비스 서버는 서비스 요청을 기다린다.

```
$ rosrn oroaa_ros_tutorials ros_tutorial_srv_server  
[INFO] [1423118250.6704810023]: ready srv server!
```

Service / Service server / Service client

7) 서비스 클라이언트 실행

- 서비스 서버를 실행했으면 이어서 다음 명령어로 서비스 클라이언트를 실행한다.

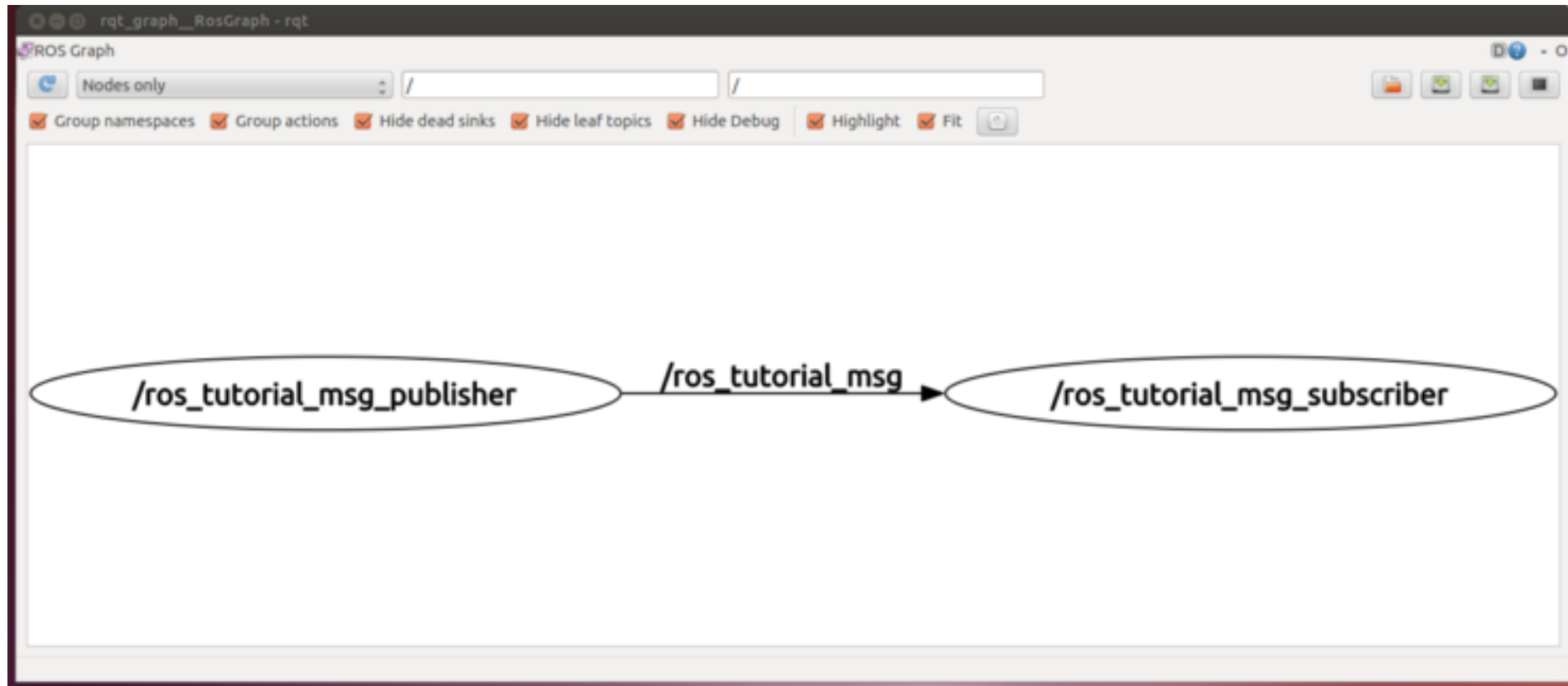
```
$ roslaunch oroca_ros_tutorials ros_tutorial_srv_client 2 3  
[INFO] [1423118255.156345643]: send srv, srv.Request.a and b: 2, 3  
[INFO] [1423118255.156345021]: receive srv, srv.Response.result: 5
```

- 서비스 클라이언트를 실행하면서 입력해준 실행 매개변수 2와 3을 서비스 요청 값으로 전송하도록 프로그래밍하였다.
- 2와 3은 각각 a, b 값으로 서비스를 요청하게 되고, 결과값으로 둘의 합인 5를 응답값으로 전송받았다.
- 여기서는 단순히 실행 매개변수로 이를 이용하였으나, 실제 활용에서는 명령어로 대체해도 되고, 계산되어야 할 값, 트리거용 변수 등을 서비스 요청 값으로 사용할 수도 있다.

Service / Service server / Service client

[참고] rqt_graph

- 서비스는 아래 그림의 토픽과는 달리 일회성이므로 rqt_graph 등에서 확인할 수 없다.



Service / Service server / Service client

[참고] rosservice call 명령어 사용 방법

- 서비스 요청은 서비스 클라이언트 노드를 실행하는 방법도 있지만 "rosservice call"이라는 명령어나 rqt의 ServiceCaller를 이용하는 방법도 있다.
- 그 중 우선, rosservice call를 사용하는 방법에 대해서 알아보자.

```
$ rosservice call /ros_tutorial_srv 3 4  
result: 7
```

```
$ rosservice call /ros_tutorial_srv 5 15  
result: 20
```

Service / Service server / Service client

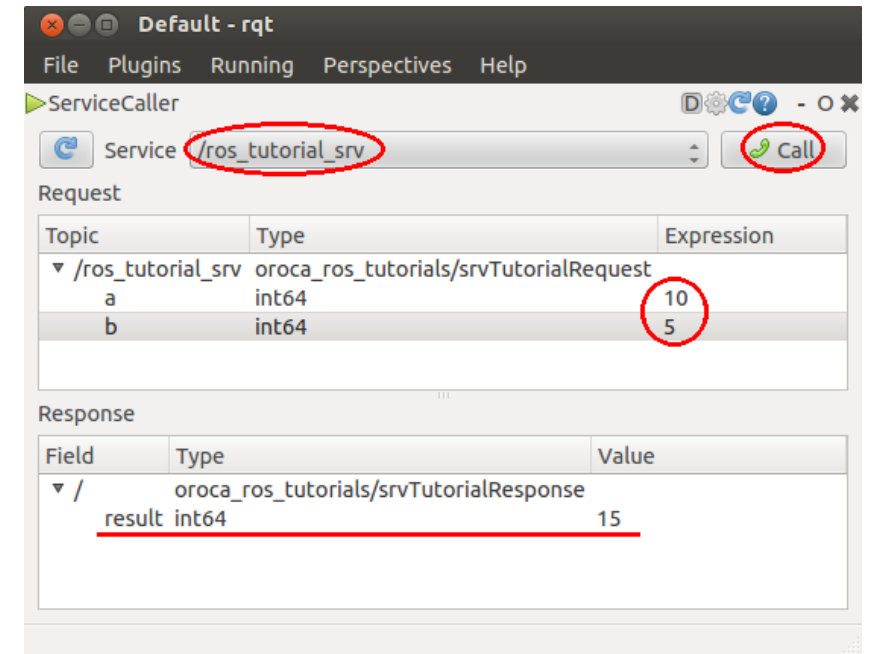
[참고] GUI 도구인 Service Caller 사용 방법

- ROS의 GUI 도구인 rqt를 실행하자.

\$ rqt

- rqt 프로그램의 메뉴에서 [플러그인(Plugins)] → [서비스(Service)] → [Service Caller]를 선택하면 다음과 같은 화면이 나온다.

- (1) servic에 /ros_tutorial_srv 입력
- (2) a = 10, b = 5 입력
- (3) Call 버튼을 누른다.
- (4) Result에 15라는 값이 표시된다.



한 가지 더!!!

- 하나의 노드는 복수의 퍼블리셔, 서브스크라이버, 서비스 서버, 서비스 클라이언트 역할도 할 수 있다!
- 마음껏 요리해 보세요~ ^^

```
ros::NodeHandle nh;  
  
ros::Publisher ros_tutorial_pub = nh.advertise<oroca_ros_tutorials::msgTutorial>("ros_tutorial_msg", 100);  
ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);  
ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv", calculation);  
ros::ServiceClient ros_tutorial_service_client = nh.serviceClient<oroca_ros_tutorials::srvTutorial>("ros_tutorial_srv");
```

Index

I. 메시지, 토픽, 서비스, 매개변수

II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

IV. 매개변수 사용법

V. roslaunch 사용법

Parameter

1) 매개변수를 활용한 노드 작성

- 이전 강의에서는 작성한 서비스 서버와 클라이언트 노드에서 `ros_tutorial_srv_server.cpp` 소스를 수정하여 서비스 요청으로 입력된 `a`와 `b`를 단순히 덧셈하는 것이 아니라, 사칙연산을 할 수 있도록 매개변수를 활용해 볼 것이다.
- 다음 순서대로 `ros_tutorial_srv_server.cpp` 소스를 수정해보자.

```
$ roscd oroca_ros_tutorials/src          → 패키지의 소스 코드 폴더인 src 폴더로 이동
$ gedit ros_tutorial_srv_server.cpp      → 소스 파일 내용 수정
```

```
#include "ros/ros.h"           // ROS 기본 헤더파일
#include "oroca_ros_tutorials/srvTutorial.h" // srvTutorial 서비스 파일 헤더 (빌드 후 자동 생성됨)

#define PLUS                    1 // 덧셈
#define MINUS                   2 // 빼기
#define MULTIPLICATION          3 // 곱하기
#define DIVISION                4 // 나누기

int g_operator = PLUS;
```

Parameter

```
// 서비스 요청이 있을 경우, 아래의 처리를 수행한다  
// 서비스 요청은 req, 서비스 응답은 res로 설정하였다
```

```
bool calculation(orooca_ros_tutorials::srvTutorial::Request &req,  
                 orooca_ros_tutorials::srvTutorial::Response &res)  
{  
    // 서비스 요청시 받은 a와 b 값을 파라미터 값에 따라 연산자를 달리한다.  
    // 계산한 후 서비스 응답 값에 저장한다  
    switch(g_operator){  
        case PLUS:  
            res.result = req.a + req.b; break;  
        case MINUS:  
            res.result = req.a - req.b; break;  
        case MULTIPLICATION:  
            res.result = req.a * req.b; break;
```

Parameter

```
case DIVISION:
    if(req.b == 0){
        res.result = 0; break;
    }
    else{
        res.result = req.a / req.b; break;
    }
default:
    res.result = req.a + req.b; break;
}
```

// 서비스 요청에 사용된 a, b값의 표시 및 서비스 응답에 해당되는 result 값을 출력한다

```
ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
ROS_INFO("sending back response: [%ld]", (long int)res.result);
```

```
return true;
}
```

Parameter

```
int main(int argc, char **argv)           // 노드 메인 함수
{
    ros::init(argc, argv, "ros_tutorial_srv_server"); // 노드명 초기화

    ros::NodeHandle nh;                     // ROS 시스템과 통신을 위한 노드 핸들 선언

    nh.setParam("calculation_method", PLUS); // 매개변수 초기설정

    // 서비스 서버 선언, oroca_ros_tutorials 패키지의 srvTutorial 서비스를 이용한
    // 서비스 서버 ros_tutorial_service_server 를 작성한다. 서비스명은 "ros_tutorial_srv" 이며,
    // 서비스 요청이 있을경우, calculation 라는 함수를 실행하라는 설정이다
    ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv", calculation);

    ROS_INFO("ready srv server!");
```


Parameter

```
ros::Rate r(10); // 10 hz

while (1)
{
    nh.getParam("calculation_method", g_operator); // 연산자를 매개변수로부터 받은 값으로 변경한다
    ros::spinOnce();                               // 콜백함수 처리루틴
    r.sleep();                                       // 루틴 반복을 위한 sleep 처리
}

return 0;
}
```

- Parameter 관련하여 설정(setParam) 및 읽기(getParam) 사용법에 주목 할 것!

[참고] 매개변수로 사용 가능 형태

- 매개변수는 integers, floats, boolean, string, dictionaries, list 등으로 설정할 수 있다.
- 간단히 예를 들자면, 1은 integer, 1.0은 floats, "Internet of Things"은 string, true는 boolean, [1,2,3]은 integers의 list, a: b, c: d는 dictionary이다.

Parameter

2) 노드 빌드 및 실행

```
$ cd ~/catkin_ws && catkin_make
```

```
$ rosrun oroca_ros_tutorials ros_tutorial_srv_server  
[INFO] [1423118250.6704810023]: ready srv server!
```

3) 매개변수 목록 보기

- "rosparam list" 명령어로 현재 ROS 네트워크에 사용된 매개변수의 목록을 확인할 수 있다. /calculation_method가 우리가 사용한 매개변수이다.

```
$ rosparam list  
/calculation_method  
/roscdistro  
/rosversion  
/run_id
```

Parameter

4) 매개변수 사용 예

- 다음 명령어대로 매개변수를 설정하여, 매번 같은 서비스 요청을 하여 서비스 처리가 달라짐을 확인해보자.
- ROS에서 매개변수는 노드 외부로부터 노드의 흐름이나 설정, 처리 등을 바꿀 수 있다. 매우 유용한 기능이므로 지금 당장 쓰지 않더라도 꼭 알아두도록 하자.

```
$ rosservice call /ros_tutorial_srv 10 5  
result: 15
```

→ 사칙연산의 변수 a, b 입력
→ 디폴트 사칙연산인 덧셈 결과값

```
$ rosparam set /calculation_method 2
```

→ 뺄셈

```
$ rosservice call /ros_tutorial_srv 10 5  
result: 5
```

```
$ rosparam set /calculation_method 3
```

→ 곱셈

```
$ rosservice call /ros_tutorial_srv 10 5  
result: 50
```

```
$ rosparam set /calculation_method 4
```

→ 나눗셈

```
$ rosservice call /ros_tutorial_srv 10 5  
result: 2
```

Index

I. 메시지, 토픽, 서비스, 매개변수

II. 퍼블리셔와 서브스크라이버 노드 작성 및 실행

III. 서비스 서버와 서비스 클라이언트 노드 작성 및 실행

IV. 매개변수 사용법

V. roslaunch 사용법

roslaunch 사용법

- `roslaunch`이 하나의 노드를 실행하는 명령어이다.
- `roslaunch`는 하나 이상의 정해진 노드를 실행시킬 수 있다.
- 그 밖의 기능으로 노드를 실행할 때 패키지의 매개변수나 노드 이름 변경, 노드 네임스페이스 설정, `ROS_ROOT` 및 `ROS_PACKAGE_PATH` 설정, 환경 변수 변경 등의 옵션을 붙일 수 있는 ROS 명령어이다.
- `roslaunch`는 `'*.launch'`라는 파일을 사용하여 실행 노드를 설정하는데 이는 XML 기반이며, 태그별 옵션을 제공한다.
- 실행 명령어는 "`roslaunch [패키지명] [roslaunch 파일]`"이다.

roslaunch 사용법

1) roslaunch의 활용

- 이전에 작성한 `ros_tutorial_msg_publisher`와 `ros_tutorial_msg_subscriber` 노드의 이름을 바꾸어서 실행해 보자. 이름만 바꾸면 의미가 없으니, 퍼블리쉬 노드와 서브스크라이버 노드를 각각 두 개씩 구동하여 서로 별도의 메시지 통신을 해보자.
- 우선, *.launch 파일을 작성하자. roslaunch에 사용되는 파일은 *.launch라는 파일명을 가지며 해당 패키지 폴더에 launch라는 폴더를 생성하고 그 폴더 안에 넣어야 한다. 다음 명령어대로 폴더를 생성하고 union.launch라는 새 파일을 생성해보자.

```
$ roscd oroca_ros_tutorials  
$ mkdir launch  
$ cd launch  
$ gedit union.launch
```

roslaunch 사용법

```
<launch>
  <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_publisher" name="msg_publisher1"/>
  <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_subscriber" name="msg_subscriber1"/>
  <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_publisher" name="msg_publisher2"/>
  <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_subscriber" name="msg_subscriber2"/>
</launch>
```

- **<launch>** 태그 안에는 roslaunch 명령어로 노드를 실행할 때 필요한 태그들이 기술된다. **<node>**는 roslaunch로 실행할 노드를 기술하게 된다. 옵션으로는 pkg, type, name이 있다.
 - **pkg** 패키지의 이름
 - **type** 실제 실행할 노드의 이름(노드명)
 - **name** 위 type에 해당하는 노드가 실행될 때 붙여지는 이름(실행명), 일반적으로 type과 같게 설정하지만 필요에 따라 실행할 때 이름을 변경하도록 설정할 수 있다.

roslaunch 사용법

- roslaunch 파일을 작성하였으면 다음처럼 union.launch를 실행하자.

```
$ roslaunch oroca_ros_tutorials union.launch --screen
```

- [참고] roslaunch 사용시에 상태 출력 방법
 - roslaunch 명령어로여럿의 노드가 실행될 때는 실행되는 노드들의 출력(info, error 등)이 터미널 스크린에 표시되지 않아 디버깅하기 어렵게 된다. 이때에 --screen 옵션을 추가해주면 해당 터미널에 실행되는 모든 노드들의 출력들이 터미널 스크린에 표시된다.

roslaunch 사용법

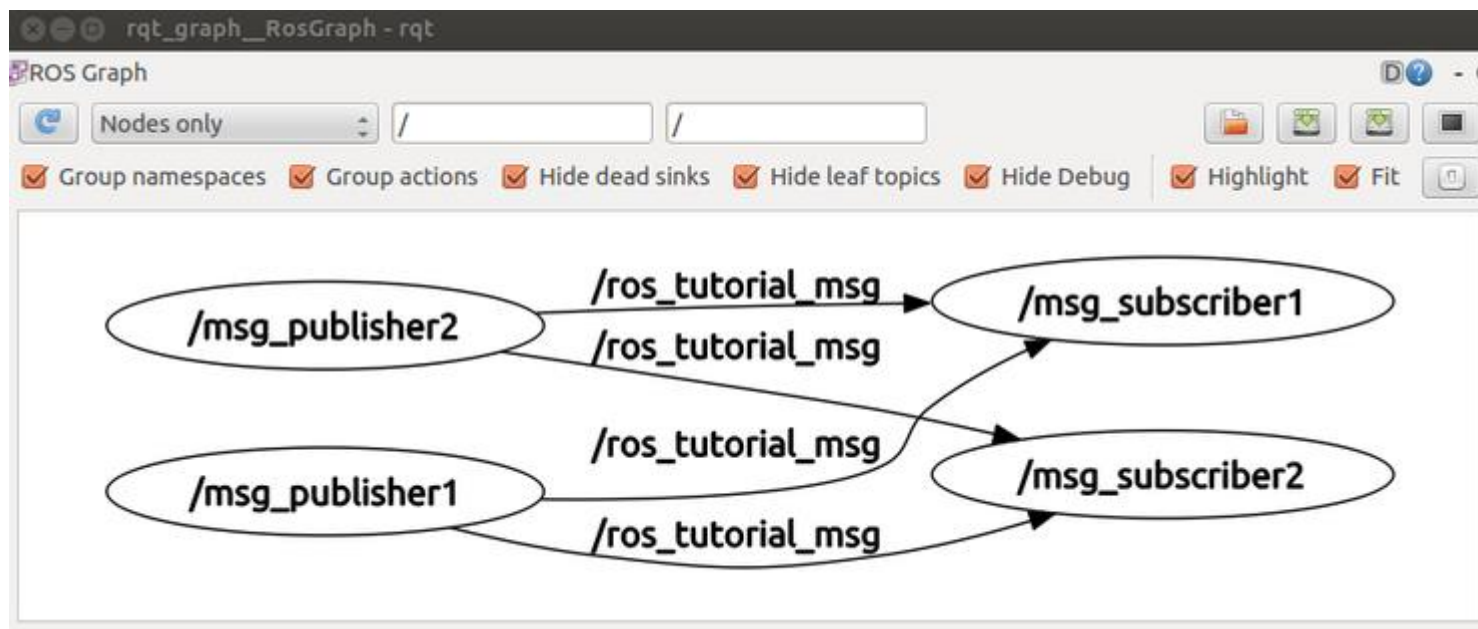
- 실행 결과?

```
$ roslaunch list  
/msg_publisher1  
/msg_publisher2  
/msg_subscriber1  
/msg_subscriber2  
/rosout
```

- 결과적으로 ros_tutorial_msg_publisher 노드가 msg_publisher1와 msg_publisher2로 이름이 바뀌어 두 개의 노드가 실행되었다.
- ros_tutorial_msg_subscriber 노드도 msg_subscriber1와 msg_subscriber2로 이름이 바뀌어 실행되었다.

roslaunch 사용법

- 문제는 "퍼블리쉬 노드와 서브스크라이버 노드를 각각 두 개씩 구동하여 서로 별도의 메시지 통신하게 한다"는 처음 의도와는 다르게 rqt_graph를 통해 보면 서로의 메시지를 모두 서브스크라이브하고 있다는 것이다.
- 이는 단순히 실행되는 노드의 이름만을 변경해주었을 뿐 사용되는 메시지의 이름을 바꿔주지 않았기 때문이다.
- 이 문제를 다른 roslaunch 네임스페이스 태그를 사용하여 해결해보자.



roslaunch 사용법

- union.launch을 수정하자.

```
$ roscd oroca_ros_tutorials/launch
```

```
$ gedit union.launch
```

```
<launch>
  <group ns="ns1">
    <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_publisher" name="msg_publisher"/>
    <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_subscriber" name="msg_subscriber"/>
  </group>
  <group ns="ns2">
    <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_publisher" name="msg_publisher"/>
    <node pkg="oroca_ros_tutorials" type="ros_tutorial_msg_subscriber" name="msg_subscriber"/>
  </group>
</launch>
```

roslaunch 사용법

- 변경 후의 실행된 노드들의 모습



roslaunch 사용법

2) launch 태그

<launch>	roslaunch 구문의 시작과 끝을 가리킨다.
<node>	노드 실행에 대한 태그이다. 패키지, 노드명, 실행명을 변경할 수 있다.
<machine>	노드를 실행하는 PC의 이름, address, ros-root, ros-package-path 등을 설정할 수 있다.
<include>	다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch파일처럼 실행할 수 있다.
<remap>	노드 이름, 토픽 이름 등의 노드에서 사용 중인 ROS 변수의 이름을 변경할 수 있다.
<env>	경로, IP 등의 환경 변수를 설정한다. (거의 안쓰임)
<param>	매개변수 이름, 타입, 값 등을 설정한다
<rosparam>	rosparam 명령어 처럼 load, dump, delete 등 매개변수 정보의 확인 및 수정한다.
<group>	실행되는 노드를 그룹화할 때 사용한다.
<test>	노드를 테스트할 때 사용한다. <node>와 비슷하지만 테스트에 사용할 수 있는 옵션들이 추가되어 있다.
<arg>	launch 파일 내에 변수를 정의할 수 있어서 아래와 같이 실행할 때 매개변수를 변경 시킬 수도 있다.

```
<launch>
  <arg name="update_period" default="10" />
  <param name="timing" value="$(arg update_period)"/>
</launch>
```

```
roslaunch my_package my_package.launch update_period:=30
```

이번 강좌의 리포지토리

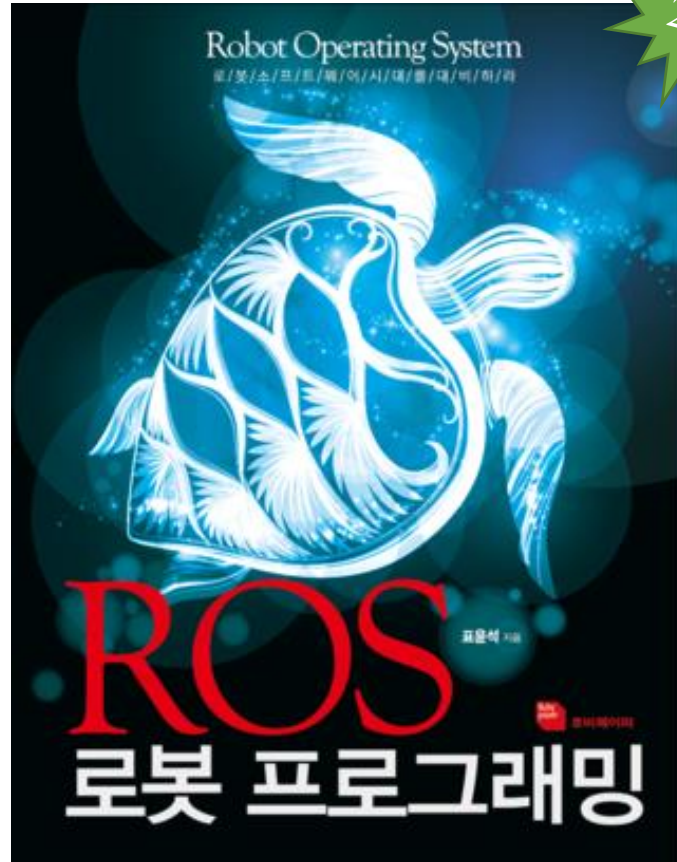
- Example 리포지토리
 - https://github.com/oroca/oroca_ros_tutorials
- 웹에서 바로 보거나 다운로드도 가능하지만 바로 적용해 보고 싶다면 다음 명령어로 리포지토리를 복사하여 빌드하면 된다.
- 참고로 본 강의의 최종판 소스코드가 업로드 되어 있기에 강의 진행을 위해 도중에 수정된 소스코드는 최종 버전만 확인 할 수 있다.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/oroca/oroca_ros_tutorials.git
$ cd ~/catkin_ws
$ catkin_make
```

이제 ROS 초보랑은 Bye Bye~!

**질문
대환영!**

여기서! 광고 하나 나가요~



국내 유일! 최초! ROS 책
비 영어권 최고의 책
인세 전액 기부

여기서! 광고 둘 나가요~



- 오로카
- www.oroqa.org
- 오픈 로보틱스 지향
- 풀뿌리 로봇공학의 저변 활성화
- 공개 강좌, 세미나, 프로젝트 진행

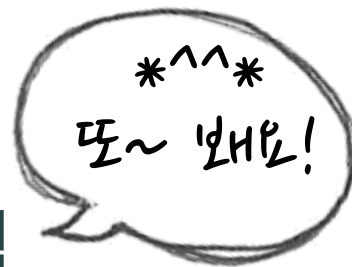
- 로봇공학을 위한 열린 모임 (KOS-ROBOT)
- www.facebook.com/groups/KoreanRobotics
- 로봇공학 통합 커뮤니티 지향
- 일반인과 전문가가 어울러지는 한마당
- 로봇공학 소식 공유
- 연구자 간의 협력

혼자 하기에 답답하시다고요?
커뮤니티에서 함께 해요~

끝.

표윤석

Yoonseok Pyo
pyo@robotis.com
www.robotpilot.net



www.facebook.com/yoonseok.pyo

Thanks for your attention!

표윤석

Yoonseok Pyo
pyo@robotis.com
www.robotpilot.net

www.facebook.com/yoonseok.pyo