# Week_7

- 챕터 1~4

## 1.1 코드 스타일 가이드

- **오픈소스 소프트웨어**는 커뮤니티의 협업을 기반으로 한 **공동의 결과물**이고, 이러한 협업을 할 때에는 **일관된 규칙**을 만들고 이를 준수해야 한다.

- 자동화 툴로 자가 검토를 진행하고
- 여러 리뷰를 받아 이러한 규칙을 준수해야 한다.

규칙은

- ROS2 developer guide
- ROS Enhancement Proposals(REPs)

와 같은 가이드를 따르고

- 코드 스타일은

ROS2 Code style을 따른다.

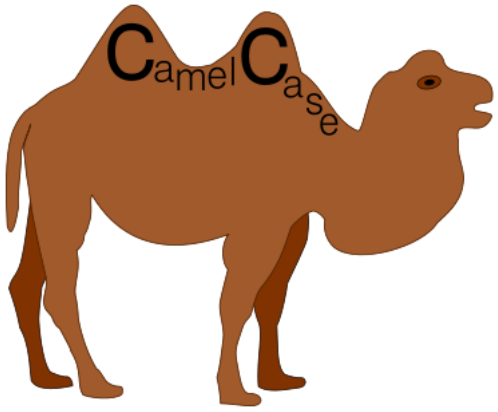
## 1.2 기본 이름 규칙

1. snake_case 이름 규칙을 사용하는 경우
    a. 파일 이름
    b. 변수명
    c. 함수명
    - 축약어는 사용하지 않는다.
    - 확장자명은 소문자로 표시한다.

- snake_case 이름 규칙은 중간에 언더라인을 붙히는 방식이다

(하이픈 [-]을 사용해도 snake-case라고 할 수 있다)

2. 타입 및 클래스명은 CamelCased 이름 규칙을 사용한다.




- CamelCased 규직은 단어가 합쳐진 부분의 처음 글자를 대문자로 표기하는 방법이다.

- **ROS인터페이스 파일**은 /msg /srv /action 폴더에 위치하고,

인터페이스 파일의 이름명은 CamelCased 규칙을 따른다.

  - *.msg / *.srv / *.action 파일은 *.h나 *.hpp로 변환하면 이는 헤더 파일이 되고, 이 헤더 파일에 구조체 및 타입이 정의되어 있어 이를 활용할 수 있게된다.

- 특정한 파일의 이름은 대소문자 규칙을 따르지 않는다.
  - package.xml
  - CMakeLists.txt
  - README.md
  - LICENSE
  - CHANGELOG.rst
  - .gitignore
  - .travix.yml
  - *.repos

## 1.3 C++ style

- ROS2 developer guide / ROS2 code style에서 다루는 C++코드 스타일은 **Google C++ Stule Guide**를 사용하고 있고, ROS의 특성에 따라서 **일부 수정**하여 사용하고 있다.

- 주요한 규칙
  - 기본 규칙
  - 라인 길이
    - 최대 100문자
  - 이름 규칙(Naming)
    - 소스 파일은 .cpp, 헤더 파일은 .hpp를 사용한다.
    - **전역 변수**는 접두어 (**g_**)를 사용한다
    - 클래스 멤버 변수는 마지막에 밑줄(_)을 붙힌다
  - 공백 문자 대 탭
    - 들여쓰기는 공백문자(Space)를 2개 사용한다

        - public, protected, private(접근 지정자)는 들여쓰기를 하지 않는다.
    - 괄호
        - if, else, do, while, for문에 괄호를 사용한다
    - 주석(Comments)
    - 린터(Linters)
        - 자동 오류 검출에는 ament_cpplint, ament_uncrustify를 사용하고
        - 정적 코드 분석에는 ament_cppcheck를 사용한다.
    - 기타
        - Boost라이브러리의 사용은 가능한 피한다
        - 포인터 구문은 char * c 처럼 사용한다

          (char* c, char *c 형태를 피한다)
        - 중첩 템플릿은 set<list<string>> 형태로 사용한다

등의 규칙이 있다.

## 1.4 Python Style

- 파이썬 코드 스타일은
    - PEPs (**P**ython **E**nhancement **P**roposals)의 **PEP8**을 준수한다.

1. 기본 규칙
    a. 파이썬 3.5 이상을 사용한다
2. 라인 길이
    a. 최대 100문자
3. 네이밍 규칙
    a. CamelCased, snake_case, ALL_CAPITALS만을 사용한다.
        i. CamelCase : 타입, 클래스
        ii. snake_case : 파일, 패키지, 인터페이스, 모듈, 변수, 함수, 메소드
        iii. ALL_CAPITALS : 상수
4. 공백 문자와 탭
    a. indent(들여쓰기)는 공백 문자(Space) 4개를 사용하고, 탭(Tap)문자는 사용하지 않는다.
    b. 문자 중간에 들여쓰는 방법 / 괄호와 문장 사용은 다음 예제에서 설명한다.

- **올바른** 사용

```
foo = function_name(var_one, var_two, var_three, var_four)

def long_long_long_long_function_name(
        var_one,
        var_two,
        var_three,
        var_four):
    print(var_one)
```

- **잘못된** 사용

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

```
foo = long_function_name(var_one, var_two,
  var_three, var_four)

def long_function_name(
  var_one, var_two, var_three,
  var_four):
  print(var_one)
```

---

5. 괄호(Brace)
    a. 괄호는  계산식/배열의 인덱스로 사용하고, 자료형이 List, Dictionary, Tupple이냐에 따라서 적절한 괄호를 사용한다.

```
list = [1, 2, 3, 4, 5]
dictionary = {'age' : 30, 'name' : '홍길동'}
tupple = (1, 2, 3, 4, 5)
```

6. 주석
    a. 문서 주석은 “””을 사용하고, (Docstring Conventions를 기술한) PEP 257을 준수한다
    b. 구현 주석은 #을 사용한다.

7. 린터(Linters)
    a. 파이썬 코드의 자동 오류 컴출에는 ament_flake8을 사용한다.

8. 기타
    a. 모든 문자는 “ 큰 따옴표가 아니라 ‘ 작은 따옴표를 사용하여 표현한다.

### 1.5 다른언어

- C언어는 C99 Standard를 준수하고, PEP-7을 참고한다
- Java언어는 Aribnb Javascript Style Guide를 참고한다.

# 2장 ROS 파이썬 프로그래밍 기초

### 2.2 패키지 생성

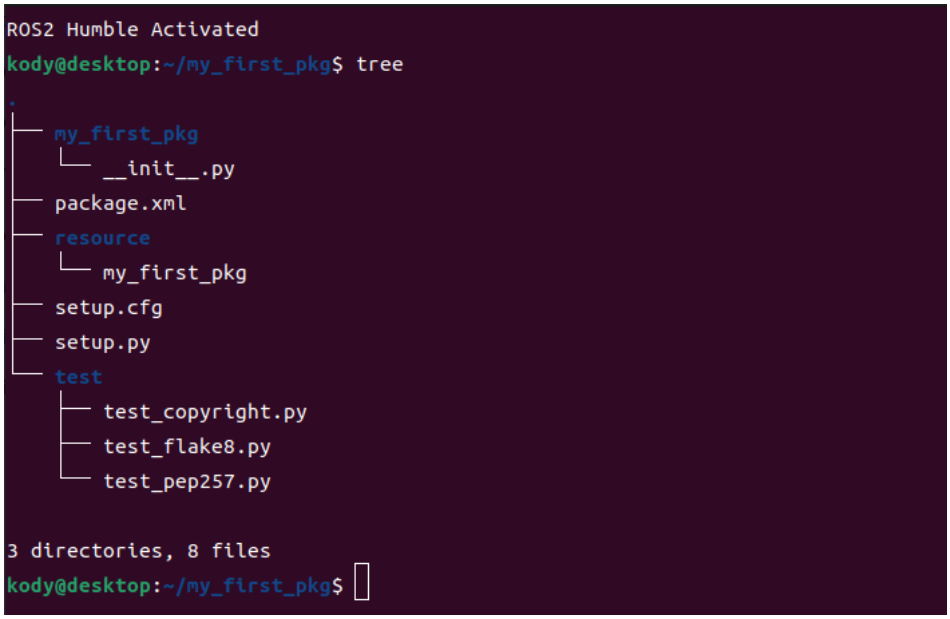
- ROS2 패키지 생성은 다음 구문에 따른다

```
$ ros2 pkg create [package name]
--build-type [build type] --dependencies [의존 패키지1] [의존 패키지n]
```

- ex

```
$ ros2 pkg create my_first_pkg --build-type ament_python --dependencies rclpy std_msgs
```

- 실행결과




## 2.3 패키지 설정

.

- helloworld_publisher.py

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String

class HelloworldPublisher(Node):
#파이썬에서 클래스() 형태를 사용할 때에
# 클래스 정의(class definition)의 경우 이는 인자(argument)를 전달해주는 것이 아닌 상속(represent)을 의미한다.

#함수(function)나 메소드(method)를 정의할 때에는 반대로, 인자(argument)를 전달하는 것을 의미한다.

    def __init__(self):
        super().__init__('helloworld_publisher')
        qos_profile = QoSProfile(depth=10)
        #여기에서의 depth는 publish/subscribe 메시지 대기열에 대한 queue사이즈를 의미한다.
        # queue는 버퍼와 같은 역할을 하며, 꽉 차게되면 가장 오래된 메시지부터 삭제된다.
        # depth 파라미터는 메시지 처리속도보다 < 메시지의 생성속도가 더 빠른 경우에 중요한 지표로 사용된다.


        self.helloworld_publisher = self.create_publisher(String, 'helloworld', qos_profile)
        # 메시지 타입이 "String"인  메시지를 helloworld토픽에 qos_profile을 적용하여 발행한다.
        self.timer_ = self.create_timer(1, self.publish_helloworld_msg)
        #뒤에서 정의하는 publish_helloworld_msg 함수를 1초마다 실행한다.

        self.count = 0

    def publish_helloworld_msg(self):
        msg = String()
        #msg를 String 타입으로 정의한다.
        msg.data = 'Hello World: {0}' % self.count
        #메시지의 필드에 Hello World: [count]를 넣는다.
        self.helloworld_publisher.publish(msg)
        # helloworld_publisher에서 정의한 msg를(String형태를) publish한다.
        self.get_logger().info('Publishing message: {0}'.format(msg.data))
        #get_logger() 메서드는 노드와 관련된 logger를 반환한다.
        #info() 메서드는 로그 메시지를 생성하는 메서드이다.
        #format() 메서드는 {0} placeholder에 msg.data를 넣는다.
        self.count += 1
        #publish_helloworld_msg 함수가 실행될 때마다 count를 1씩 증가시킨다.

    def main(args=None):
        #default args값을 None으로 설정한다.

        rclpy.init(args=args)
```

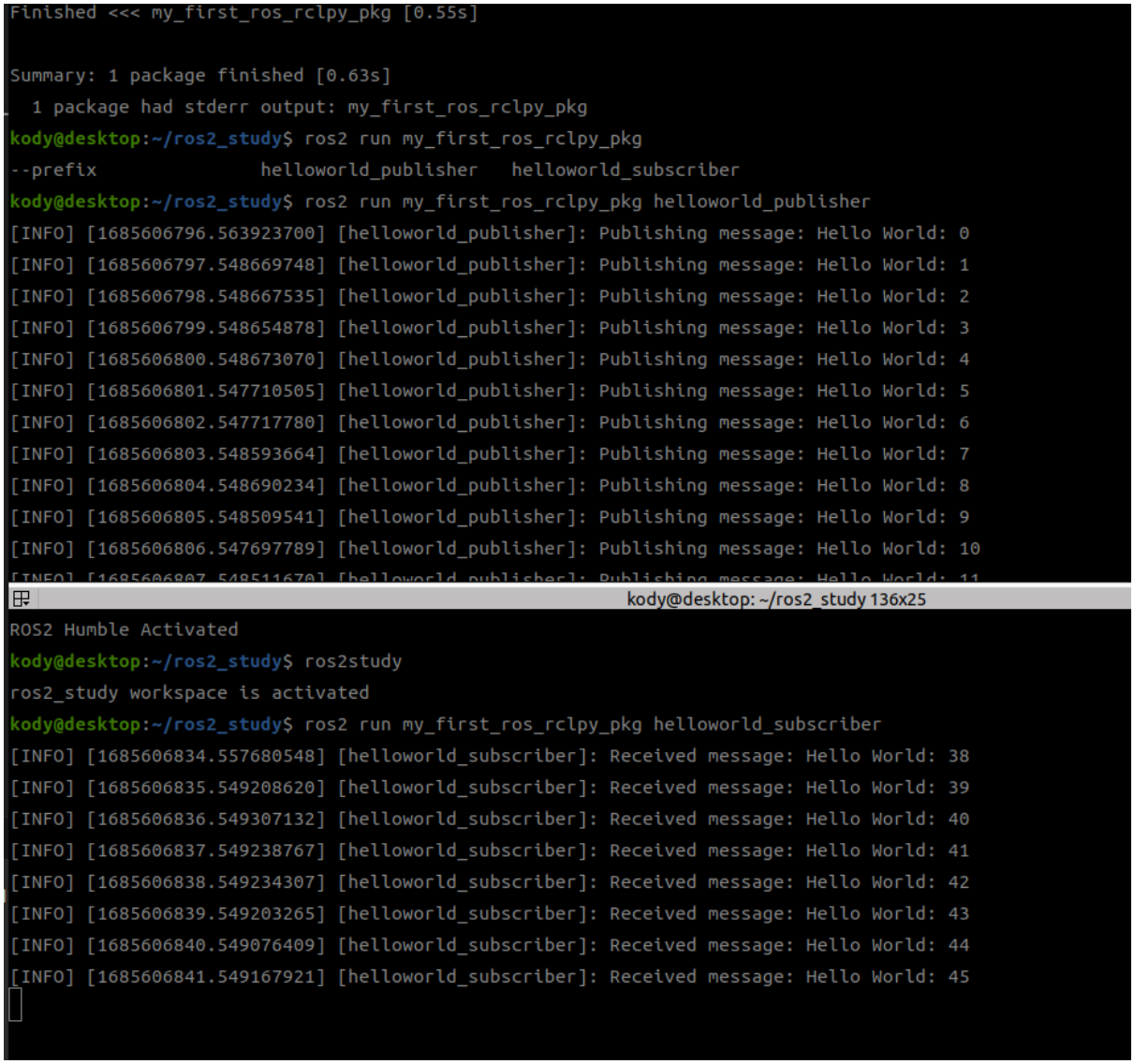
```
        node = HelloworldPublisher()
        try:
        #try 구문을 시도한다.
            rclpy.spin(node)

        except KeyboardInterrupt:
        #try에서 예외가 발생하면 KeyboardInterrupt를 발생시킨다.
            node.get_logger().info('Keyboard Interrupt (SIGINT)')
        finally:
        #try에서 예외가 발생하든 안하든 무조건 실행하는 구문이다.
            node.destroy_node()
            rclpy.shutdown()

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

## 2.5 서브스크라이버 노드 작성

- helloworld_subscriber.py

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String

class HelloworldSubscriber(Node):
# rclpy.node의 Node로부터 상속받는다.

    def __init__(self):


        super().__init__('helloworld_subscriber')
        # 부모 클래스(super())의 생성자를 호출한다. (__init__() 메서드로 생성자를 호출한다)
        # 부모 클래스로부터, __init__()으로 생성자를 생성하고
        # __init__()의 argument로 helloworld_subscriber를 전달하여 노드를 생성해 주었다.

        # 여기에서 부모 클래스(super)은 Node가 되기 때문에
        # 부모 클래스의 생성자 (__init__)는 노드가 되는 것이다.

        qos_profile = QoSProfile(depth=10)
        #버퍼 = 10개

        self.helloworld_subscriber = self.create_subscription(
            String, #토픽의 *메시지 타입*은 String 형태이다.
            'helloworld', #토픽의 *이름*은 helloworld이다.

            self.subscribe_topic_message,

            #(rclpy.python의 Node클래스가 생성하는)
            #create_subscription() 메소드의 세 번째 인자는 콜백 함수를 의미한다.

            qos_profile) # QoS는 qos_profile로 설정한다.


    def subscribe_topic_message(self, msg):
        #콜백 함수를 정의하는 부분이다.
        self.get_logger().info('Received message: {0}'.format(msg.data))
        #

def main(args=None):
    #main 함수의 argument를 None으로 설정한다.
    rclpy.init(args=args)
    #None인 main함수의 agrs값을 rclpy의 생성자 인수인 args에 대입한다.
    #rclpy의 생성자 인수를 None으로 만들어 주었다.
    node = HelloworldSubscriber()
    #인수(argument)에 아무것도 전달하지 않아 생성자(construcor)를 호출해 주었다.
    # constructor는 __init__를 호출하는 생성자이다.

    # HelloworldSubscriber의 클래스 정의 부분에서 consructor를 super의 'helloworld_subscriber'으로 정의해주고,
    # main함수 부분에서 HelloworldSubscriber()을 이용하여 생성자를 호출하여 node에 할당하여
    # 모듈식으로 노드를 생성해 주었다.

    try:
        rclpy.spin(node) #이 구문을 시도한다.
    except KeyboardInterrupt:
        node.get_logger().info('Keyboardf Interrupt (SIGINT)')
        #예외시 KeyboardInterrupt를 발생시킨다.
```

```
    finally:
        #예외와 관계없이 실행하는 부분이다.
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__' :
    #다른 모듈의 코드가 실행되지 않도록 하는 부분이다.
    main()
```




- 파이썬으로는 처음 해보기때문에 애먹었지만, 실행하는 것까지는 성공했다.

```
kody@desktop:~/ros2_study$ colcon build --symlink-install
Starting >>> my_first_ros_rclpy_pkg
--- stderr: my_first_ros_rclpy_pkg
/usr/lib/python3/dist-packages/setuptools/command/easy_install.py:158: EasyInstallDeprecationWarning: easy_install command is deprecat
  warnings.warn(
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use bui
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 1.16.0-unknown is an invalid version and
  warnings.warn(
```

```
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 1.1build1 is an invalid version and will
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 0.1.43ubuntu1 is an invalid version and
  warnings.warn(
---
Finished <<< my_first_ros_rclpy_pkg [0.55s]

Summary: 1 package finished [0.63s]
  1 package had stderr output: my_first_ros_rclpy_pkg
```

```
kody@desktop:~/ros2_study$ colcon build --symlink-install
Starting >>> my_first_ros_rclpy_pkg
--- stderr: my_first_ros_rclpy_pkg
/usr/lib/python3/dist-packages/setuptools/command/easy_install.py:158: EasyInstallDeprecationWarning: easy_install command is deprecated
. Use build and pip and other standards-based tools.
  warnings.warn(
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build
 and pip and other standards-based tools.
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 1.16.0-unknown is an invalid version and w
ill not be supported in a future release
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 1.1build1 is an invalid version and will n
ot be supported in a future release
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprecationWarning: 0.1.43ubuntu1 is an invalid version and wi
ll not be supported in a future release
  warnings.warn(
---
Finished <<< my_first_ros_rclpy_pkg [0.55s]

Summary: 1 package finished [0.63s]
  1 package had stderr output: my_first_ros_rclpy_pkg
```

- colcon build시의 에러는 남겨진 숙제로…

# 3장 ROS C++ 프로그래밍 기초

## 3.2 패키지 생성

- 형식

```
$ ros2 pkg create [패키지이름] --build-tpe [빌드타입]
--dependencies [의존 패키지1] [의존 패지지n]
```

- 실행

```
$ ros2 pkg create my_first_ros_rclcpp_pkg --build-type ament_cmake
--dependencies rclcpp std_msgs
```

- helloworld_publishser.cpp

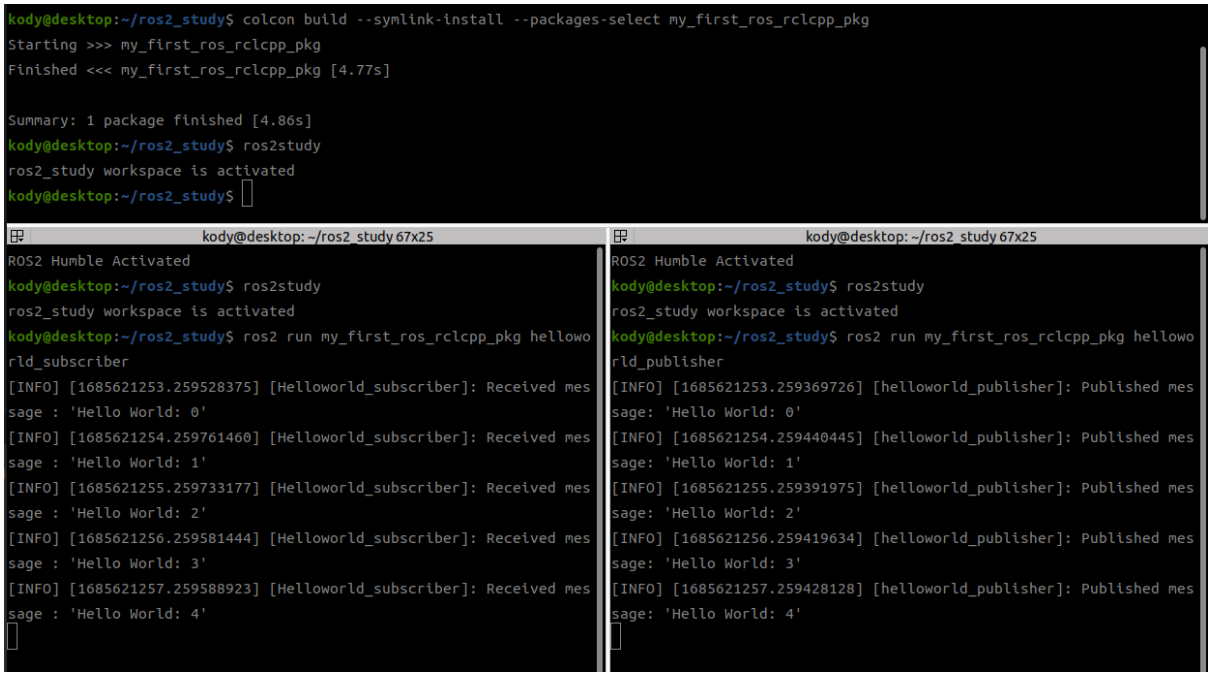
```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
```

```
using namespace std::chrono_literals;

class HelloworldPublisher : public rclcpp::Node
{
public:
    HelloworldPublisher()
    : Node("helloworld_publisher"), count_(0)
    {
        auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
        //qos_profile을 생성하고 버퍼 용량을 10으로 설정한다.
        helloworld_publisher_ = this->create_publisher<std_msgs::msg::String>(
            "helloworld",qos_profile);
        // 토픽 이름을 helloworld로 하고, qos_profile을 사용한다.
        timer_ = this->create_wall_timer(
            1s, std::bind(&HelloworldPublisher::publish_helloworld_msg, this));
        // create_wall_timer()메서드를 이용하여 timer_에 반환하는 구문이다.
        // 메서드의 첫 번째 인자는 지연시간, 두 번째 인자는 콜백 함수이다.
        // (period, callback,   group = nullptr)
    };
private:
    void publish_helloworld_msg()
    {
        auto msg = std_msgs::msg::String();
        msg.data = "Hello World: " + std::to_string(count_++);
        RCLCPP_INFO(this->get_logger(), "Published message: '%s'", msg.data.c_str());
        helloworld_publisher_-> publish(msg);

    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr helloworld_publisher_ ;
    size_t count_;
    // timer_, helloworld_publisher_ ,  count_를 정의하는 부분이다.
    // 실수로 publish_helloworld_msg() 메서드 안에 작성하면 public에서 인식하지 못한다.
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc,argv);
    auto node = std::make_shared<HelloworldPublisher>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

- helloworld_subscriber.cpp

```
#include <functional>
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using std::placeholders::_1;
//bind 함수에 사용할 placeholder를 정의한다.

class HelloworldSubscriber : public rclcpp::Node
{
public:
    HelloworldSubscriber()
    : Node("Helloworld_subscriber")
    {
        auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
        helloworld_subscriber_ = this->create_subscription<std_msgs::msg::String>(
            "helloworld",
            qos_profile,
            std::bind(&HelloworldSubscriber::subscribe_topic_message, this, _1));
        //create_subscription의 인자는 다음과 같다
        //1. 토픽 이름 (const std::string & topic_name)
        //2. QoS (const rclcpp::QoS & qos)
        //3. 콜백 함수(CallbackT && callback)
        //4. 구독 옵션(SubscriptionOptionsWithAllocator)

        //3번째 인자인 bind메소드는
    }
```

```
// create_subscription 메소드의 형식

// template<typename MessageT, typename CallbackT, typename AllocatorT, typename SubscriptionT>
// 탬플릿 부분이다

// std::shared_ptr<SubscriptionT> create_subscription(
//   const std::string & topic_name,
//   const rclcpp::QoS & qos,
//   CallbackT && callback,
//   const SubscriptionOptionsWithAllocator<AllocatorT> & options = (
//     SubscriptionOptionsWithAllocator<AllocatorT>()
//   ),
//   typename MessageMemoryStrategy<MessageT, AllocatorT>::SharedPtr
//   msg_mem_strat = MessageMemoryStrategy<MessageT, AllocatorT>::create_default()
// );


//bind 함수에서 바인드하는 값은 2개 [this,_1]이고,
//this는 클래스의 object(인스턴스)를 나타내고
//_1은 인자 (const std_msgs::msg::String::SharedPtr msg)을 나타낸다.

private:
    void subscribe_topic_message(const std_msgs::msg::String::SharedPtr msg) const
    {
        RCLCPP_INFO(this->get_logger(), "Received message : '%s'", msg->data.c_str());
    }
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr helloworld_subscriber_;
    //helloworld_subscriber_의 유형을 Subscription의 SharedPtr로 정의한다.
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc,argv);
    auto node = std::make_shared<HelloworldSubscriber>();
    // 말 그대로 shared pointer를 make하는 메소드이다.
    // HelloworldSubscriber에 속하는 생성자를 생성한 후에 변수 node에 반환한다.
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```




- 실행 결과

# 4장 ROS2 Tips

## 4.2 setup.bash vs local_setup.bash

### 4.2.1 underlay와 overlay

- ros를 binary로 설치하면 ROS는 /opt/ros/humble에 위치하고
- 소스코드를 받아 ROS를 설정하면 ~/ros2_humble/와 같은 경로를 사용하게 된다
  - 이러한 ROS 의 개발환경을 **underlay**라고 한다.

- 개발자의 워크스페이스 개발 환경을 ROS2에서는 **overlay**라고 한다.

→ **overlay** 환경은 **ROS 패키지에 의존**하기 때문에 **underlay** 환경에 종속적이게 된다.

### 4.2.2 setup.bash / local_setup.bash

- local_setup.bash는 **overlay**에 대한 환경을 설정하는 것으로, **underlay에 대한 환경은 포함하지 않는다.**
- setup.bash는 **underlay**에 대한 환경을 설정하는 것으로, **로컬의 개발환경**을 사용하고 싶으면 overlay환경을 추가로 설정해야 한다.

## 4.3 colcon_cd

- .bashrc파일에 다음을 추가하여 colcon_cd 명령을 사용할 수 있다.

```
source /usr/share/colcon_cd/function/colcon_cd.sh
export _colcon_cd_root=~/<워크스페이스 이름>
```

## 4.4 ROS_DOMAIN_ID / Namespace

- 같은 네트워크를 사용하고 있으면 노드에 대한 정보를 쉽게 공유할 수 있는데 독립적인 작업을 할 때에는 이 기능이 불편할 수 있고, 이를 방지하는 방법은 다음과 같다
  - **물리적으로 다른 네트워크를 사용**
  - **ROS_DOMAIN_ID를 이용**하여 DDS의 도메인을 변경
  - 노드 / 토픽/서비스/액션의 이름에 Namespace 추가

### 4.4.1 물리적으로 다른 네트워크

- 이 방법이 가장 편하다고 한다

### 4.4.2 ROS_DOMAIN_ID

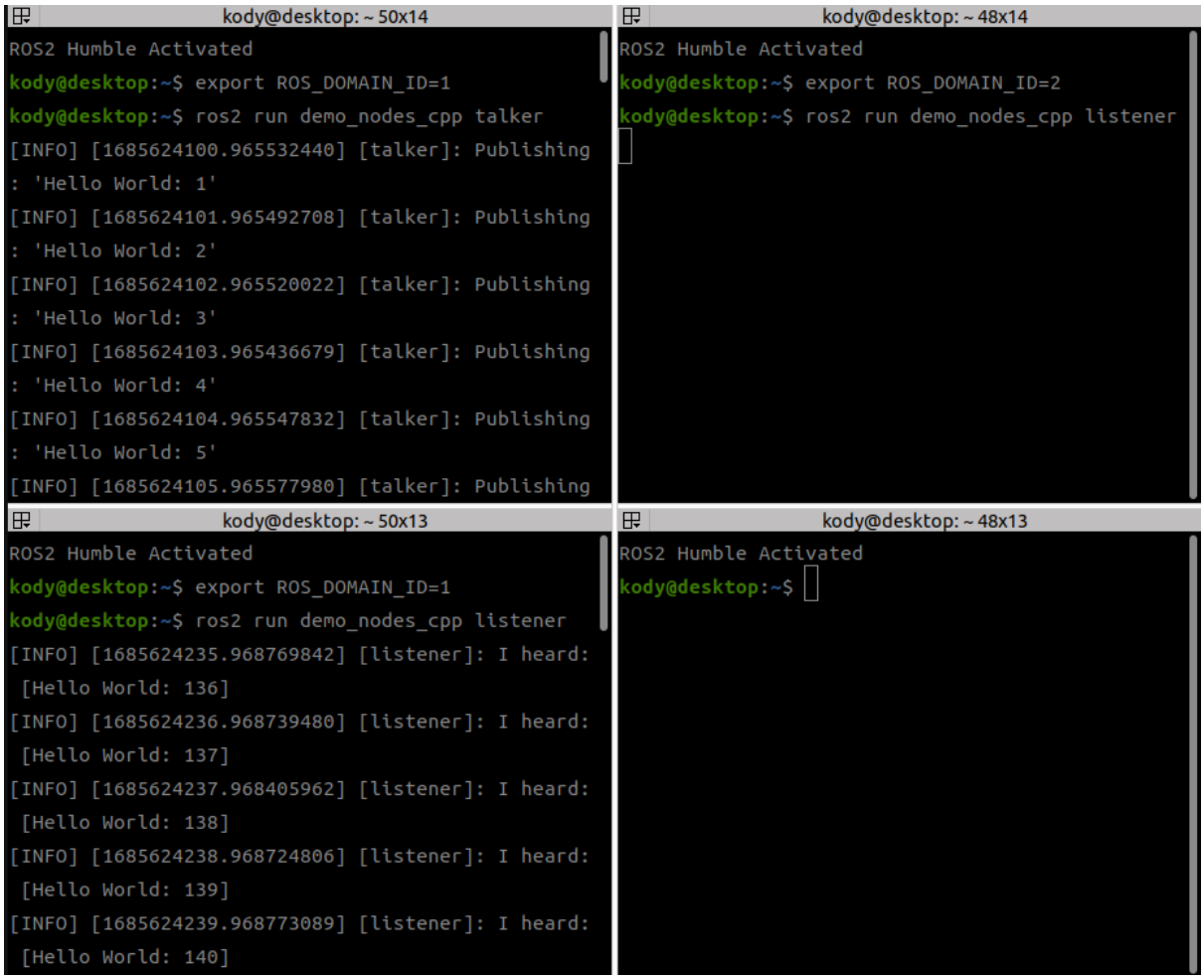
- ROS2에서는 DDS를 채택하여

  DDS Global Space공간에 있는 토픽들을 publish/subscribe할 수 있고

- ROS_DOMAIN_ID를 변경하면 DDS Global Space를 변경하여 데이터를 의도에 따라 취사선택할 수 있다.

- 도메인 아이디 설정 (export를 이용하여 환경변수를 설정함)

```
$ export ROS_DOMAIN_ID=<DOMAIN NUMBER>
```

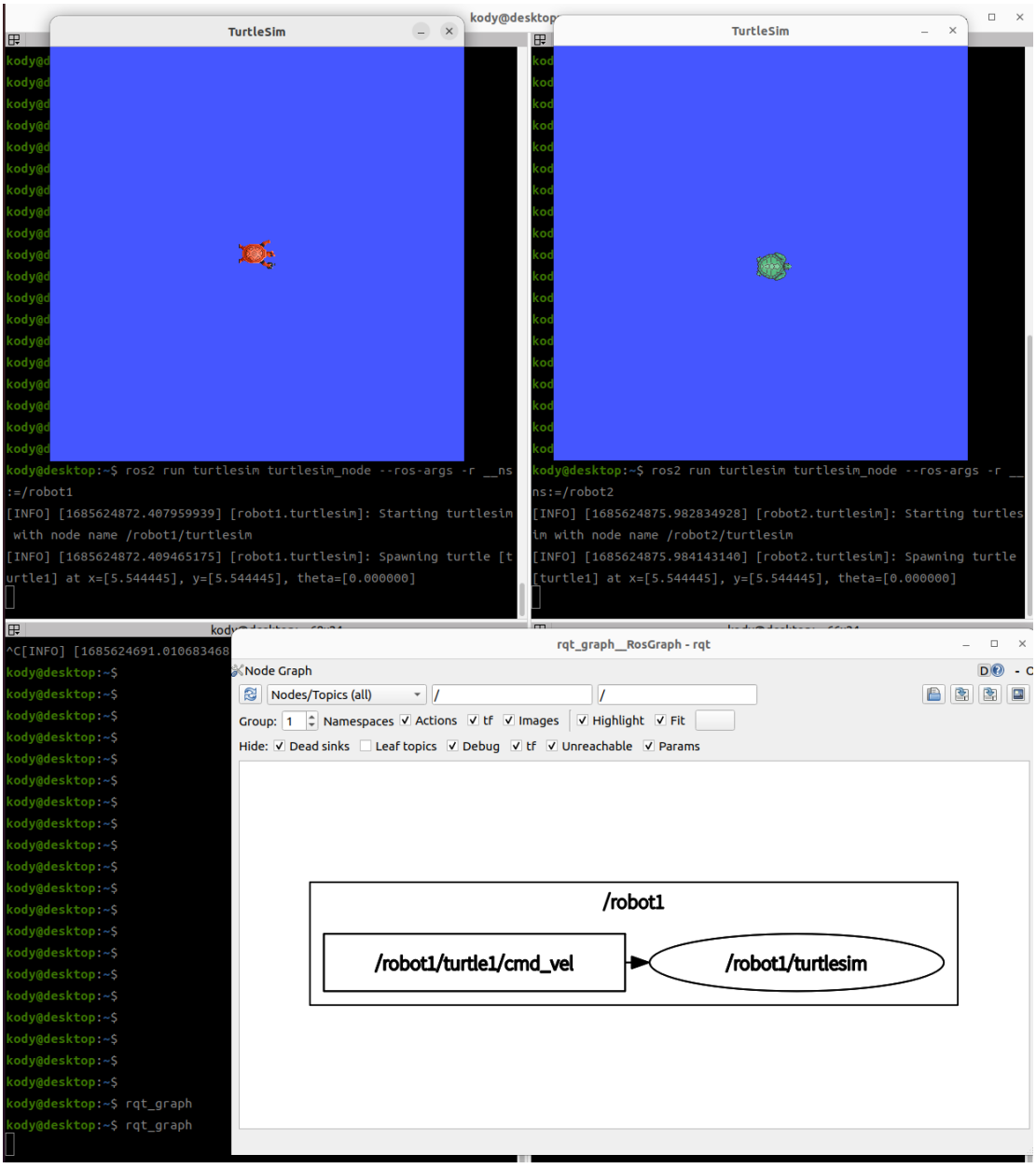



- demo talker / listener을 실행해 보았을 때,
  - 도메인 아이디가 같은 listener은 잘 듣고
  - 도메인 아이디가 다른 listener은 반응이 없음을 확인할 수 있다.

## 4.4.3 ROS2 Namespace

- 노드/토픽/서비스/액션/파라미터는 **고유한 이름**을 가지고

  이러한 **고유 이름**에 Namespace를 **추가로 붙혀주면** 고유한 네트워크를 그룹화 할 수 있다.




- /robot1 Namespace는 잘 표시 되는데,
  /robot2 Namespace는 표시되지 않는 문제가 발생했다.

## 4.5 colcon과 vcstool에 대한 명령어 자동 완성 기능

- 빌드 툴인 colcon과 버전 관리 툴인 vcstool은 [옵션] [서브-옵션]을 가지고 있지만
  자동 완성 기능을 기본 설정으로 지원하지 않아 불편하다.

  자동 완성 기능을 기본으로 사용하려면

- colcon-argcomplete.bash파일과
- vcs.bash파일을

  .bashrc파일(시스템 전역 설정)에 설정하면 자동 완성 기능을 사용할 수 있다.

- .bashrc 파일에 추가

```
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
source /usr/share/vcscool-completion/vcs.bash
```

# 5장 토픽,서비스,액션 인터페이스

## 5.1 ROS2 인터페이스 신규 작성

- ROS2 프로그래밍은 메시지 통신을 위해 기본적인 인터페이스들을 제공한다.
  - std_msgs
  - geometry_msgs의 Twist.msg
  - sensor_msgs의 LaserScan.msg등이 있다.

- 하지만 이러한 **기본적인 인터페이스**만을 이용해서 사용자의 의도를 실현하거나 개발 과정에서의 모든 요구를 반영하기는 쉽지 않기 때문에 커스텀 인터페이스를 만드는 것이 필요하다.

- ROS2의 인터페이스는 패키지를 사용하려는 **패키지에 포함시키는 방법으로도 구현할 수 있지만**,

  **인터페이스로만 구성된 별도 패키지를 구성**하는 것이 **의존성 관리 측면**에서 더 변하다고 할 수 있다.

- 책의 예제에서는 msg_srv_action_interface_example로 예제를 만든다.

  이름을 줄이기 위해서 msa_example으로 패키지를 만든다.
  - ArithArgument.msg
  - ArithOperator.srv
  - ArithChecker.action

## 5.2 인터페이스 패키지 만들기

```
$ ros2 pkg create --build-type ament_cmake msa_example
$ cd msa_example
$ mkdir msg srv action
```

```
kody@desktop:~$ cd ros2_study
kody@desktop:~/ros2_study$ ROS2
ROS2 Humble Activated
kody@desktop:~/ros2_study$ ros2 pkg create --build-type ament_cmake msa_example
going to create a new package
package name: msa_example
destination directory: /home/kody/ros2_study
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['kody <pg01198@naver.com>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./msa_example
creating ./msa_example/package.xml
creating source and include folder
creating folder ./msa_example/src
creating folder ./msa_example/include/msa_example
creating ./msa_example/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
kody@desktop:~/ros2_study$ cd msa_example
kody@desktop:~/ros2_study/msa_example$ mkdir msg srv action
kody@desktop:~/ros2_study/msa_example$ dir
CMakeLists.txt  action  include  msg  package.xml  src  srv
kody@desktop:~/ros2_study/msa_example$
```

```
kody@desktop:~/ros2_study/msa_example$ tree
.
├── CMakeLists.txt
├── action
│   └── ArithChecker.action
├── include
│   └── msa_example
├── msg
│   └── ArithArgument.msg
├── package.xml
├── src
└── srv
    └── ArithOperator.srv
```

[ArithArgument.msg]

```
# Messages
builtin_interfaces/Time stamp
float32 argument_a
float32 argument_b
```

- builtin_interfaces/Time 형식으로 stamp를 정의했다.
- builtin_interfaces/Time 형식으로 정의된 stamp는 sec, nanosec 필드로 구성된다.

---

[ArithOperator.src]

- Constants(상수) 부분을 정의해준 후에
- Request / Response 부분을 정의해 주었다.
    - Constants 부분은 Request / Response 부분의 일부가 아니고,
      서비스 정의의 최상위 수준에서 사용되었으며
      request와 response 부분에서 사용할 수 있는 부분이라 할 수 있다.

```
# Constants
int8 PLUS = 1
int8 MINUS = 2
int8 MULTIPLY = 3
int8 DIVISION = 4

# Request
int8 arithmetic_operator

---

# Response
float32 arithmetic_result
```

---

[ArithChecker.action]

```
# Goal
float32 goal_sum

---

# Result
string[] all_formula
float32 total_sum

---
# Feedback
string[] formula
```

### 5.3 패키지 설정 파일(package.xml)

- 파일에 추가

```
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>buildin_interfaces</exec_depend>
  <exec_depend>rosidl_defalut_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>
```

- <buildtool_depend>으로 **rosidl_default_generators**을 추가했다.
  - <buildtool_depend> 태그는 이름 그대로 패키지를 빌드(**build**)할때 필요한 툴(**tool**)의 의존성을(**depend**) 명시하는데 사용된다.
  - rosidl_defalut_generators는 ROS IDL 파이프라인의 일부로, .msg .srv .action파일을 .hpp로 변환한다(C++의 경우)

- <exec_depend>는 execute depend로, 코드를 실행할 때에 필요한 의존성을 명시하는 부분이다.
  - **buildin_interfaces**와 **rosidl_default_runtime**을 추가했다.
    - **buildin_interfaces**는 ROS2가 상호 작용을 하는데 필요한 기본 데이터 모델의 기본을 제공하는 패키지이다.

      Time, Duration, Empty등을 제공한다.
    - **rosidl_default_runtime**은 런타임에 메시지들이 인터페이스를 사용할 수 있도록 지원하는 패키지이다.

- <member_of_group>부분은 이 패키지가 어떤 그룹의 일부인지를 나타내는 부분이다.
  - rosidl_interface_packages를 사용해 주었기 때문에 이 패키지는 이 그룹에 속하고,

    이 패키지는 rosidl 기능을 제공할 것으로 기대되게 된다.

## 5.4 빌드 설정 파일(CMakeLists.txt)

CMakeLists.txt에 추가한다.

```
find_package(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

set(msg_files
  "msg/ArithArgument.msg"
)

set(srv_files
  "srv/ArithOperator.msg"
)

set(action_files
  "action/ArithChecker.msg"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
```

- package.xml에서는
  - rosidl_defalut_generators
  - builtin_interfaces
  - rosidl_default_runtime
  - rosidl_interface_packages

    를 사용해 주었지만

- CMakeLists.txt에서 find_package로는
  - builtin_interfaces와
  - rosidl_default_generators밖에 명시해주지 않은 이유는
  - CMakeLists.txt는 **runtime**이나 **패키지 그룹**에 대한 정보를 담고있지 **않고**,

    이 파일은 순수하게 **빌드 시**에만의 정보를 담고 있기 때문이다.

그렇기 때문에

- 런타임에 대한 rosidl_defalut_runtime에 대한 정보와
- 패키지 그룹에 대한 rosidl_interface_packages에 대한 정보는 담고 있지 않다.

---

- **CMakeLists.txt**는 **빌드 환경**에 대한 의존성과 환경을 제공하고
- **package.xml**은 **빌드, 런타임, 테스트 등 전반**에 대한 의존성과 환경을 제시한다.
  - 포함 관계로 비유하여 나타낸다면 이렇게 나타낼 수 있다.
    - package.xml ⊃ CMakeLists.txt

```
set(msg_files
  "msg/ArithArgument.msg"
)
```

- msg_files 파라미터의 내용을 "msg/ArithArgument.msg"으로 설정해 주었다.

```
rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
```

- rosidl_generate_interfaces()의 파라미터로
  - 패키지 이름 ${PROJECT_NAME}
  - 파일 목록

    ${msg_files} → 앞에서 설정한 msg_files 파라미터이다.
    ${srv_files}
    ${action_files}
  - 의존성(DEPENDENCIES)을 전달하였다.

## 5.5 빌드하기

```
kody@desktop:~/ros2_study$ colcon build --symlink-install --packages-select msa_example
Starting >>> msa_example
Finished <<< msa_example [0.32s]

Summary: 1 package finished [0.40s]
```