

Week_7

- 챗터 1~4

1.1 코드 스타일 가이드

- 오픈소스 소프트웨어는 커뮤니티의 협업을 기반으로 한 공동의 결과물이고, 이러한 협업을 할 때에는 일관된 규칙을 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

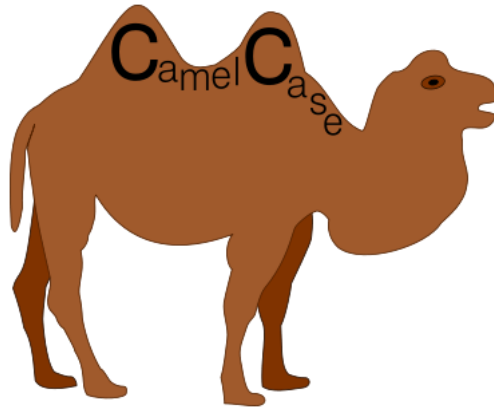
1. snake_case 이름 규칙을 사용하는 경우

- 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 규칙은 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
 - .travis.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 변수는 접두어 (g_)를 사용한다
 - 클래스 멤버 변수는 마지막에 밑줄(_)을 붙인다
 - 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 (Python Enhancement Proposals)의 PEP8을 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)

- 괄호는 계산식/배열의 인덱스로 사용하고, 자료형이 List, Dictionary, Tuple이냐에 따라서 적절한 괄호를 사용한다.

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

6. 주석

- 문서 주석은 `"""`을 사용하고, (Docstring Conventions를 기술한) PEP 257을 준수한다
- 구현 주석은 `#`을 사용한다.

7. 린터(Linters)

- 파이썬 코드의 자동 오류 검출에는 `ament_flake8`을 사용한다.

8. 기타

- 모든 문자는 “ 큰 따옴표가 아니라 ‘ 작은 따옴표를 사용하여 표현한다.

1.5 다른언어

- C언어는 C99 Standard를 준수하고, PEP-7을 참고한다
- Java언어는 Airbnb 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

```

ROS2 Humble Activated
kody@desktop:~/my_first_pkg$ tree
.
├── my_first_pkg
│   └── __init__.py
├── package.xml
├── resource
│   └── my_first_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py

3 directories, 8 files
kody@desktop:~/my_first_pkg$

```

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, 꼭 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 args값을 rclpy의 생성자 인수인 args에 대입한다.
    #rclpy의 생성자 인수를 None으로 만들어 주었다.
    node = HelloworldSubscriber()
    #인수(argument)에 아무것도 전달하지 않아 생성자(constructor)를 호출해 주었다.
    # constructor는 __init__를 호출하는 생성자이다.

    # HelloworldSubscriber의 클래스 정의 부분에서 constructor를 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('Keyboard 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()

```

```

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$ ros2 run my_first_ros_rclpy_pkg
--prefix                helloworld_publisher  helloworld_subscriber
kody@desktop:~/ros2_study$ ros2 run my_first_ros_rclpy_pkg helloworld_publisher
[INFO] [1685606796.563923700] [helloworld_publisher]: Publishing message: Hello World: 0
[INFO] [1685606797.548669748] [helloworld_publisher]: Publishing message: Hello World: 1
[INFO] [1685606798.548667535] [helloworld_publisher]: Publishing message: Hello World: 2
[INFO] [1685606799.548654878] [helloworld_publisher]: Publishing message: Hello World: 3
[INFO] [1685606800.548673070] [helloworld_publisher]: Publishing message: Hello World: 4
[INFO] [1685606801.547710505] [helloworld_publisher]: Publishing message: Hello World: 5
[INFO] [1685606802.547717780] [helloworld_publisher]: Publishing message: Hello World: 6
[INFO] [1685606803.548593664] [helloworld_publisher]: Publishing message: Hello World: 7
[INFO] [1685606804.548690234] [helloworld_publisher]: Publishing message: Hello World: 8
[INFO] [1685606805.548509541] [helloworld_publisher]: Publishing message: Hello World: 9
[INFO] [1685606806.547697789] [helloworld_publisher]: Publishing message: Hello World: 10
[INFO] [1685606807.548511670] [helloworld_publisher]: Publishing message: Hello World: 11
kody@desktop:~/ros2_study 136x25
ROS2 Humble Activated
kody@desktop:~/ros2_study$ ros2study
ros2_study workspace is activated
kody@desktop:~/ros2_study$ ros2 run my_first_ros_rclpy_pkg helloworld_subscriber
[INFO] [1685606834.557680548] [helloworld_subscriber]: Received message: Hello World: 38
[INFO] [1685606835.549208620] [helloworld_subscriber]: Received message: Hello World: 39
[INFO] [1685606836.549307132] [helloworld_subscriber]: Received message: Hello World: 40
[INFO] [1685606837.549238767] [helloworld_subscriber]: Received message: Hello World: 41
[INFO] [1685606838.549234307] [helloworld_subscriber]: Received message: Hello World: 42
[INFO] [1685606839.549203265] [helloworld_subscriber]: Received message: Hello World: 43
[INFO] [1685606840.549076409] [helloworld_subscriber]: Received message: Hello World: 44
[INFO] [1685606841.549167921] [helloworld_subscriber]: Received message: Hello World: 45

```

- 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 deprecated
  warnings.warn(
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build
  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 w
ill 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 속제로...

3장 ROS C++ 프로그래밍 기초

3.2 패키지 생성

- 형식

```

$ ros2 pkg create [패키지이름] --build-type [빌드타입]
--dependencies [의존 패키지1] [의존 패키지n]

```

- 실행

```

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

```

- helloworld_publisher.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>
// 템플릿 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;
}

```

```

kody@desktop:~/ros2_study$ colcon build --symlink-install --packages-select my_first_ros_rclcpp_pkg
Starting >>> my_first_ros_rclcpp_pkg
Finished <<< my_first_ros_rclcpp_pkg [4.77s]

Summary: 1 package finished [4.86s]
kody@desktop:~/ros2_study$ ros2study
ros2_study workspace is activated
kody@desktop:~/ros2_study$

```

| kody@desktop: ~/ros2_study67x25 | kody@desktop: ~/ros2_study67x25 |
|--|---|
| ROS2 Humble Activated | ROS2 Humble Activated |
| kody@desktop:~/ros2_study\$ ros2study | kody@desktop:~/ros2_study\$ ros2study |
| ros2_study workspace is activated | ros2_study workspace is activated |
| kody@desktop:~/ros2_study\$ ros2 run my_first_ros_rclcpp_pkg helloworld_subscriber | kody@desktop:~/ros2_study\$ ros2 run my_first_ros_rclcpp_pkg helloworld_publisher |
| [INFO] [1685621253.259528375] [Helloworld_subscriber]: Received message : 'Hello World: 0' | [INFO] [1685621253.259369726] [helloworld_publisher]: Published message: 'Hello World: 0' |
| [INFO] [1685621254.259761460] [Helloworld_subscriber]: Received message : 'Hello World: 1' | [INFO] [1685621254.259440445] [helloworld_publisher]: Published message: 'Hello World: 1' |
| [INFO] [1685621255.259733177] [Helloworld_subscriber]: Received message : 'Hello World: 2' | [INFO] [1685621255.259391975] [helloworld_publisher]: Published message: 'Hello World: 2' |
| [INFO] [1685621256.259581444] [Helloworld_subscriber]: Received message : 'Hello World: 3' | [INFO] [1685621256.259419634] [helloworld_publisher]: Published message: 'Hello World: 3' |
| [INFO] [1685621257.259588923] [Helloworld_subscriber]: Received message : 'Hello World: 4' | [INFO] [1685621257.259428128] [helloworld_publisher]: Published message: 'Hello World: 4' |

- 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 ROS의 개발환경을 underlay라고 한다.
- 개발자의 워크스페이스 개발 환경을 ROS2에서는 overlay라고 한다.

→ overlay 환경은 ROS 패키지에 의존하기 때문에 underlay 환경에 종속적이게 된다.

4.2.2 setup.bash / local_setup.bash

- local_setup.bash는 overlay에 대한 환경을 설정하는 것으로, underlay에 대한 환경은 포함하지 않는다.
- setup.bash는 underlay에 대한 환경을 설정하는 것으로, 로컬의 개발환경을 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 이용하여 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)

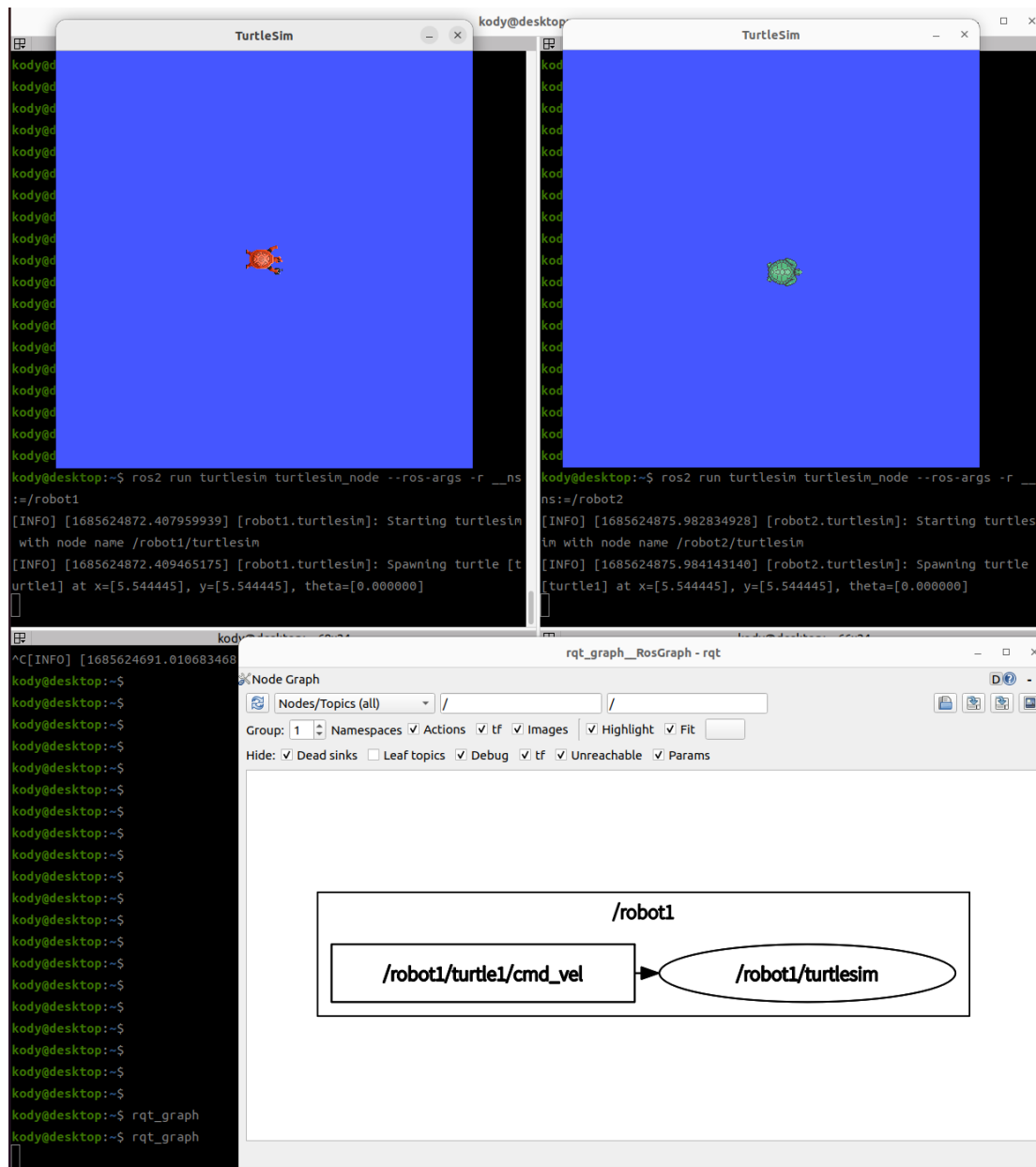
```
$ export ROS_DOMAIN_ID=<DOMAIN NUMBER>
```

| | |
|---|--|
| <pre>kody@desktop: ~ 50x14 ROS2 Humble Activated kody@desktop:~\$ export ROS_DOMAIN_ID=1 kody@desktop:~\$ ros2 run demo_nodes_cpp talker [INFO] [1685624100.965532440] [talker]: Publishing : 'Hello World: 1' [INFO] [1685624101.965492708] [talker]: Publishing : 'Hello World: 2' [INFO] [1685624102.965520022] [talker]: Publishing : 'Hello World: 3' [INFO] [1685624103.965436679] [talker]: Publishing : 'Hello World: 4' [INFO] [1685624104.965547832] [talker]: Publishing : 'Hello World: 5' [INFO] [1685624105.965577980] [talker]: Publishing</pre> | <pre>kody@desktop: ~ 48x14 ROS2 Humble Activated kody@desktop:~\$ export ROS_DOMAIN_ID=2 kody@desktop:~\$ ros2 run demo_nodes_cpp listener</pre> |
| <pre>kody@desktop: ~ 50x13 ROS2 Humble Activated kody@desktop:~\$ export ROS_DOMAIN_ID=1 kody@desktop:~\$ ros2 run demo_nodes_cpp listener [INFO] [1685624235.968769842] [listener]: I heard: [Hello World: 136] [INFO] [1685624236.968739480] [listener]: I heard: [Hello World: 137] [INFO] [1685624237.968405962] [listener]: I heard: [Hello World: 138] [INFO] [1685624238.968724806] [listener]: I heard: [Hello World: 139] [INFO] [1685624239.968773089] [listener]: I heard: [Hello World: 140]</pre> | <pre>kody@desktop: ~ 48x13 ROS2 Humble Activated kody@desktop:~\$</pre> |

- 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 이름을 가지고
이러한 고유 이름에 Namespace를 추가로 붙여주면 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
```