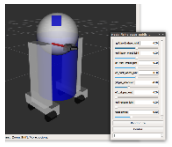
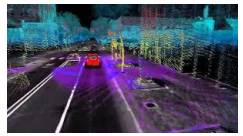
ROKEY BOOT CAMP

ROS-2 프로그래밍 실습

ROS2 프로그래밍 실습 목차

1	주제 : ROS2 CLI ▪ ROS2 CLI / CLI 명령 종류 및 사용법 ros2/ros2cli ROS 2 command line interface tools	주제 : ROS2 심화 기능1 ▪ 신규 ROS2 CLI(ros2env.zip)
2	주제 : ROS2 심화 기능1 ▪ Intra-Process Communication ▪ DDS의 QoS(Quality of Service) 이해 및 활용법 Intra-process Communication	주제 : ROS2 심화 기능1 ▪ DDS의 QoS(Quality of Service) 이해 및 활용법 ▪ QoS 실습 예제(QoS(py_pubsub.zip)) ros2/QoS
3	주제 : ROS2 심화 기능2 ▪ Component ▪ RQt Plugin 사용법 및 실습(rqt_example.zip) 	주제 : ROS2 심화 기능2 ▪ RQt Plugin 사용법 및 실습(rqt_example.zip) Qt
4	주제 : ROS2 심화 기능2 ▪ Lifecycle(노드관리)에 대한 이해 및 활용법 ▪ Security(ROS2의 보안) ROS 2 Security	주제 : 시뮬레이션 개발 ▪ Urdf, tf2 
5	주제 : 시뮬레이션 개발 ▪ Gazebo, SLAM, Nav2 ROS GAZEBO 	주제 : 두산 로봇과 ROS2 ▪ ROS2를 활용한 두산 로봇의 기본 명령어 
6	주제 : ROS2와 OpenCV ▪ OpenCV + ROS2 연동 및 기능 구현 ROS 2 + OpenCV	주제 : ROS2와 OpenCV ▪ Lane detect 구현 및 ROS2 통합실습 
7	주제 : ROS2와 3차원 영상 및 시각화 ▪ Point Cloud 개념 이해(2D, 3D RViz) ▪ Open3D 	▪ ROS2 정기 평가 

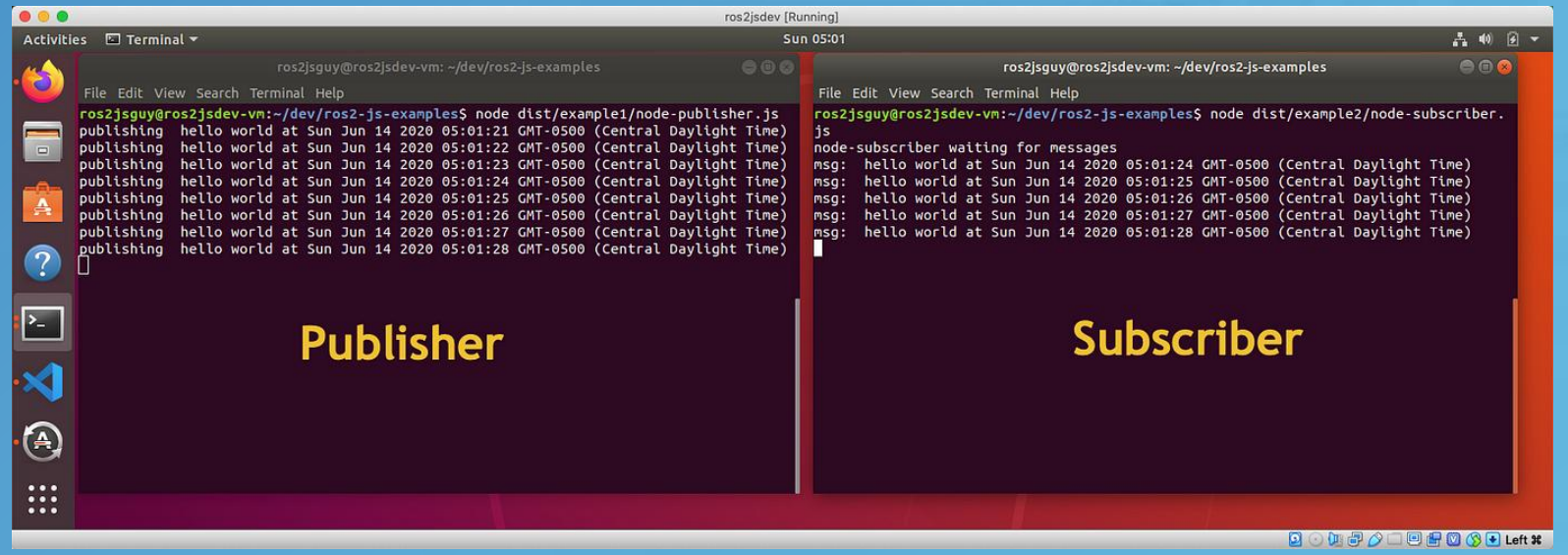
ROKEY BOOT CAMP

ROS-2 프로그래밍 실습 강의자료

1 ~ 4 차시

Contents

ROS2 CLI
(Command Line Interface)



ROS2 CLI 사용법

ROS2 CLI 명령어

```
ros2 [verbs] [sub-verbs] [options] [arguments]
```

- **verbs**: 동작을 지정하며, 수행할 작업의 유형을 나타냄. run, topic, service등이 올 수 있음
- **sub-verbs**: 특정 동작에 대한 세부 동작(sub-verb)을 지정함. Verbs가 topic인 경우 pub, echo, list 등이 올 수 있음
- **options**: 명령어의 실행 방식을 설정하는 추가 파라미터. -h, --node-name, --qos 등이 올 수 있음
- **arguments**: 실행할 때 필요한 인수를 지정함. 특정 노드의 이름이나 토픽의 이름, 서비스 이름 등이 올 수 있음

ROS2 CLI

ROS2 CLI 사용법

- ROS2 CLI 명령어
  - -h 옵션을 이용하면 verbs, sub-verbs, option등에 대하여 더 자세히 알 수 있음

```
user@user:~$ ros2 -h
usage: ros2 [-h] [--use-python-default-buffering] Call `ros2 <command> -h` for more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

options:
  -h, --help            show this help message and exit
  --use-python-default-buffering
                        Do not force line buffering in stdout and instead use the python default
                        buffering, which might be affected by PYTHONUNBUFFERED/-u and depends on
                        whatever stdout is interactive or not

Commands:
  action     Various action related sub-commands
  bag        Various rosbag related sub-commands
  component  Various component related sub-commands
  control    Various control related sub-commands
  daemon     Various daemon related sub-commands
  doctor     Check ROS setup and other potential issues
  interface  Show information about ROS interfaces
  launch     Run a launch file
  lifecycle  Various lifecycle related sub-commands
  multicast  Various multicast related sub-commands
  node       Various node related sub-commands
  param      Various param related sub-commands
  pkg        Various package related sub-commands
  run        Run a package specific executable
  security   Various security related sub-commands
  service    Various service related sub-commands
  topic      Various topic related sub-commands
  wtf        Use `wtf` as alias to `doctor`

  Call `ros2 <command> -h` for more detailed usage.
```

ROS2 CLI 실행 명령어

- ROS2 CLI + arguments

ros2cli + [verbs]	[arguments]	기능
ros2 run	〈package〉 〈executable〉	특정 패키지의 특정 **노드 실행** (1개의 노드) * executable에 따라 복수 노드도 실행 가능
ros2 launch	〈package〉 〈launch-file〉	특정 패키지의 특정 **런치 파일 실행** (0개 ~ 복수개의 노드)

ROS2 CLI

ROS2 CLI 실행 명령어

- **ROS2 CLI + arguments 예시**
  - ROS2에서 turtlesim 시뮬레이터를 실행하는 기본 명령어

```
$ ros2 run turtlesim turtlesim_node
```

  - ROS2의 demo_nodes_cpp 패키지에서 talker_listener.launch.py 파일을 실행

```
$ ros2 launch demo_nodes_cpp talker_listener.launch.py
```

ROS2 CLI

ROS2 CLI 정보 명령어

ROS2 CLI 정보 명령어

- ROS2 CLI + sub-verbs

ros2cli + [verbs]	[sub-verbs]	기능
ros2 pkg	create	새로운 ROS2 패키지 생성
	executables	지정 패키지의 실행 파일 목록 출력
	list	사용 가능한 패키지 목록 출력
	prefix	지정 패키지의 저장 위치 출력
	xml	지정 패키지의 패키지 정보 파일(xml) 출력
ros2 node	info	실행 중인 노드 중 지정한 노드의 정보 출력
	list	실행 중인 모든 노드의 목록 출력
ros2 topic	bw	지정 토픽의 대역폭 측정
	delay	지정 토픽의 지연시간 측정
	echo	지정 토픽의 데이터 출력
	find	지정 타입을 사용하는 토픽 이름 출력
	hz	지정 토픽의 주기 측정
	info	지정 토픽의 정보 출력
	list	사용 가능한 토픽 목록 출력
	pub	지정 토픽의 토픽 발행
	type	지정 토픽의 토픽 타입 출력

ROS2 CLI

ROS2 CLI 정보 명령어

- ROS2 CLI + sub-verbs

ros2cli + [verbs]	[sub-verbs]	기능
ros2 service	call	지정 서비스의 서비스 요청 전달
	find	지정 서비스 타입의 서비스 출력
	list	사용 가능한 서비스 목록 출력
	type	지정 서비스의 타입 출력
ros2 action	info	지정 액션의 정보 출력
	list	사용 가능한 액션 목록 출력
	send_goal	지정 액션의 액션 목표 전송
ros2 interface	list	사용 가능한 모든 인터페이스 목록 출력
	package	특정 패키지에서 사용 가능한 인터페이스 목록 출력
	packages	인터페이스 패키지들의 목록 출력
	proto	지정 패키지의 프로토타입 출력
	show	지정 인터페이스의 데이터 형태 출력

ROS2 CLI

ROS2 CLI 정보 명령어

- ROS2 CLI 실습

```
ros2 run image_tools cam2image
```

```
ros2 topic list
ros2 topic info /image
ros2 interface show sensor_msgs/msg/Image
```

```
ros2 run image_tools cam2image ---ros-args -r image:=/camera/image_raw
```

```
ros2 topic list
ros2 topic info /camera/image_raw
ros2 topic bw /camera/image_raw
```

ROS2 CLI

ROS2 CLI 정보 명령어

- ROS2 CLI + sub-verbs

ros2cli + [verbs]	[sub-verbs]	기능
ros2 param	delete	지정 파라미터 삭제
	describe	지정 파라미터 정보 출력
	dump	지정 파라미터 저장
	get	지정 파라미터 읽기
	list	사용 가능한 파라미터 목록 출력
	set	지정 파라미터 쓰기
ros2 bag	info	저장된 rosbag 정보 출력
	play	rosbag 기록
	record	rosbag 재생

ROS2 CLI 정보 명령어

- ROS2 CLI + sub-verbs 예시
  - turtlesim 패키지에서 실행 가능한 모든 노드 및 실행 파일들을 나열

```
$ ros2 pkg executables turtlesim
```

  - /turtlesim 노드에 대한 정보를 표시함. 노드의 이름, 관련된 토픽, 서비스 및 파라미터 정보 확인 가능
    (turtlesim 시뮬레이터가 실행중이어야 함)

```
$ ros2 node info /turtlesim
```

ROS2 CLI

ROS2 CLI 기능 정보 명령어

※ CLI 후반부에 실습

ROS2 CLI 기능 보조 명령어

- ROS2 CLI + verbs + sub-verbs

ros2cli + [verbs]	[sub-verbs] (options)	기능
ros2 extensions	(-a) (-v)	ros2cli의 extension 목록 출력 (ros2cli개발용으로 사용, 일반적 사용 x)
ros2 extension_points	(-a) (-v)	ros2cli의 extension point 목록 출력 (ros2cli개발용으로 사용, 일반적 사용 x)
ros2 daemon	start	daemon 시작
	status	daemon 상태 보기
	stop	daemon 정지
ros2 multicast	receive	multicast 수신
	send	multicast 전송

ROS2 CLI

ROS2 CLI 기능 정보 명령어

- ROS2 CLI + verbs + sub-verbs

ros2cli + [verbs]	[sub-verbs] (options)	기능
ros2 doctor	hello (-r) (-rf) (-iw)	ROS 설정 및 네트워크, 패키지 버전, rmw 미들웨어 등과 같은 잠재적 문제를 확인하는 도구
ros2 wtf	hello (-r) (-rf) (-iw)	doctor와 동일함 (ros2 doctor의 alias) (WTF: Where's The Fire)
ros2 lifecycle	get	라이프사이클 정보 출력
	list	지정 노드의 사용 가능한 상태 전이 목록 출력
	nodes	라이프사이클을 사용하는 노드 목록 출력
	set	라이프사이클 상태 전환 트리거

ROS2 CLI

ROS2 CLI 기능 정보 명령어

- ROS2 CLI + verbs + sub-verbs

ros2cli + [verbs]	[sub-verbs]	기능
ros2 component	list	실행 중인 컨테이너와 컴포넌트 목록 출력
	load	지정 컨테이너 노드의 특정 컴포넌트 실행
	standalone	표준 컨테이너 노드로 특정 컴포넌트 실행
	types	사용 가능한 컴포넌트들의 목록 출력
	unload	지정 컴포넌트의 실행 중지
ros2 security	create_key	보안키 생성
	create_keystore	보안키 저장소 생성
	create_permission	보안 허가 파일 생성
	generate_artifacts	보안 정책 파일을 이용하여 보안키 및 보안 허가 파일 생성
	generate_policy	보안 정책 파일(policy.xml) 생성
	list_keys	보안키 목록 출력

ROS2 CLI 기능 보조 명령어

- **ROS2 CLI + verbs + sub-verbs 예시**
  - ROS2에서 사용할 수 있는 모든 extension들을 나열함

```
$ ros2 extensions -a
```

  - ROS2 시스템의 상태를 진단하고 문제를 확인하는 도구

```
$ ros2 doctor -r
```

ros2 run

- **run은 특정 패키지의 특정 노드를 실행하는 명령어**
  - turtlesim 패키지의 turtle_sim node를 실행

```
$ ros2 run turtlesim turtlesim_node
```

  - turtlesim 패키지의 turtle_teleop_key를 실행

```
$ ros2 run turtlesim turtle_teleop_key
```

ros2 launch

- **launch는 특정 패키지의 특정 런치 파일을 실행하는 명령어. 복수 개의 노드 실행이나 또 다른 패키지의 다른 런치 파일을 불러와 실행 가능**
  - turtlesim 패키지의 turtle_sim node를 실행

```
$ ros2 run turtlesim turtlesim_node
```

  - ROS2에서 demo_nodes_cpp 패키지에 포함된 talker_listener.launch.py 파일을 실행하여 두 개의 노드를 동시에 시작

```
$ ros2 launch demo_nodes_cpp talker_listener.launch.py
```

ROS2 CLI

ROS2 CLI 실습

ros2 pkg

- **pkg는 지정 패키지의 정보를 얻거나 패키지를 생성하는 데 사용되는 명령어**
  - ament python 빌드 형태의 rclpy, std_msgs 패키지에 의존성을 가진 my_ros_pkg 패키지를 생성

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

  - turtlesim 패키지에 포함된 실행 파일 목록을 확인

```
$ ros2 pkg executables turtlesim
```

  - 설치된 패키지 및 본인이 직접 작성한 패키지 중 사용 가능한 모든 패키지의 목록을 확인

```
$ ros2 pkg list
```

  - turtlesim 패키지의 저장 위치를 확인

```
$ ros2 pkg prefix turtlesim
```

  - turtlesim 패키지의 패키지 정보 파일(package.xml)을 확인

```
$ ros2 pkg xml turtlesim
```

ros2 node

- node는 노드의 정보를 얻는데 사용하는 명령어
  - 실행중인 모든 노드의 목록을 확인

```
$ ros2 node list
```

  - /turtlesim 노드의 정보를 확인

```
$ ros2 node info /turtlesim
```

ros2 topic

- topic은 토픽의 구성, 대역폭, 지연 시간, 인터페이스 형태 등의 정보를 얻거나 특정 토픽을 송신 및 수신하는 데 사용되는 명령어
  - /turtle1/cmd_vel 토픽의 대역폭을 확인

```
$ ros2 topic bw /turtle1/cmd_vel
```

  - /turtle1/cmd_vel 토픽의 데이터를 확인

```
$ ros2 topic echo /turtle1/cmd_vel
```

  - 지정한 geometry_msgs/msg/Twist 인터페이스를 사용하고 있는 토픽명을 확인

```
$ ros2 topic find geometry_msgs/msg/Twist
```

  - /turtle1/cmd_vel 토픽의 주기를 확인

```
$ ros2 topic hz /turtle1/cmd_vel
```

- 예제가 정상적으로 동작하기 위해서는 turtlesim_node와 turtle_teleop_key노드가 실행되어 있어야 함
- 만일 아무것도 뜨지 않을 경우 turtle_teleop_key를 이용해 turtle을 이동

ROS2 CLI

ROS2 CLI 실습

- /turtle1/cmd_vel 토픽의 인터페이스 형태, 토픽의 퍼블리시 및 서브스크라이브 정보를 확인

```
$ ros2 topic info /turtle1/cmd_vel
```

- 현재 개발 환경에서 동작 중인 모든 노드들의 토픽 이름을 확인

```
$ ros2 topic list
```

- 현재 개발환경에서 동작중인 모든 노드들의 인터페이스 형태와 함께 토픽 이름을 확인

```
$ ros2 topic list -t
```

- /turtle1/cmd_vel 토픽을 퍼블리시한다. 테스트용으로 주로 사용

```
$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 3.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 2.0}}"
```

- /turtle1/cmd_vel 토픽의 인터페이스 형태를 확인

```
$ ros2 topic type /turtle1/cmd_vel
```

예제가 정상적으로 동작하기 위해서는 turtlesim_node와 turtle_teleop_key노드가 실행되어 있어야 함

ros2 service

- **service는 서비스의 정보를 얻거나 직접 서비스 요청을 테스트해 볼 수 있는 명령어**
  - turtlesim/srv/SetPen 인터페이스 형태를 사용하고 있는 /turtle1/set_pen 서비스를 특정 값을 요청 값으로 콜

```
$ ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r: 255, g: 255, b: 255, width: 10}"
```

  - std_srvs/srv/Empty 인터페이스 형태의 서비스를 사용하는 서비스명을 확인

```
$ ros2 service find std_srvs/srv/Empty
```

  - 현재 개발환경에서 동작 중인 모든 노드들의 서비스 이름을 확인

```
$ ros2 service list
```

  - 현재 개발 환경에서 동작 중인 모든 노드들의 인터페이스 형태와 함께 서비스 이름을 확인

```
$ ros2 service list -t
```

  - /clear 서비스의 인터페이스 형태를 확인

```
$ ros2 service type /clear
```

ros2 action

- **action은 액션의 정보를 얻거나 직접 액션 목표 전달을 테스트해 볼 수 있는 명령어**
  - turtle1/rotate_absolute 액션을 사용하는 액션 서버 및 클라이언트 노드 이름 및 개수를 확인

```
$ ros2 action info /turtle1/rotate_absolute
```

  - 현재 개발환경에서 동작 중인 모든 노드들의 액션 이름을 확인

```
$ ros2 action list
```

  - 현재 개발환경에서 동작 중인 모든 노드들의 인터페이스 형태와 액션 이름을 확인

```
$ ros2 action list -t
```

  - turtlesim/action/RotateAbsolute 인터페이스 형태를 사용하는 /turtle1/rotate_absolute 액션에 특정 값으로 액션 목표를 전달

```
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute
"{theta: 1.5708}"
```

ros2 interface

- **interface는 토픽/서비스/액션에서 사용하는 인터페이스의 정보를 얻는 데 사용되는 명령어**
  - 현재 개발 환경의 모든 msg, srv, action 인터페이스를 확인
    ```
    $ ros2 interface list
    ```
  - 지정한 turtlesim 패키지에 포함된 인터페이스들을 확인
    ```
    $ ros2 interface package turtlesim
    ```
  - Msg, srv, action 인터페이스를 담고 있는 패키지의 목록을 확인
    ```
    $ ros2 interface packages
    ```
  - 지정한 geometry_msgs/msg/Twist 인터페이스의 기본 형태를 확인
    ```
    $ ros2 interface proto geometry_msgs/msg/Twist
    ```
  - 지정한 각 메시지의 인터페이스 및 메시지 이름을 확인
    ```
    $ ros2 interface show geometry_msgs/msg/Twist
    ```

ros2 param

- **param은 파라미터의 정보를 확인하고 파라미터를 설정하거나 읽어오는 등의 일을 수행할 수 있는 명령어**
  - 사용 가능한 모든 파라미터 목록을 확인

```
$ ros2 param list
```

  - /turtlesim 노드의 background_r 파라미터의 값을 읽어 옴

```
$ ros2 param get /turtlesim background_r
```

  - /turtlesim 노드의 background_r 파라미터를 250이라는 값으로 설정

```
$ ros2 param set /turtlesim background_r 250
```

ros2 param

- 파라미터가 어떤 형태, 목적, 인터페이스 형태, 최소/최댓값을 갖는지 확인

```
$ ros2 param describe /turtlesim background_r
```

- /turtlesim 노드의 PARAMETER 1이라는 이름을 갖는 파라미터를 삭제 (현재는 삭제 가능한 파라미터가 없음)

```
$ ros2 param delete /turtlesim {PARAMTER 1}
```

- 현재 폴더에 /turtlesim노드의 파라미터들을 yaml 형태로 저장. 특정 이름을 지정하지 않으면 지정한 노드 이름으로 파일이 생성됨

```
$ ros2 param dump /turtlesim
```

ros2 bag

- **bag은 토픽을 저장하거나 재생할 때 사용하는 명령어**
  - 원하는 토픽을 'my_turtle'이라는 이름으로 저장

```
$ ros2 bag record -o my_turtle /turtle1/cmd_vel
```

  - 모든 토픽을 저장하고 싶다면 "-a"옵션 사용

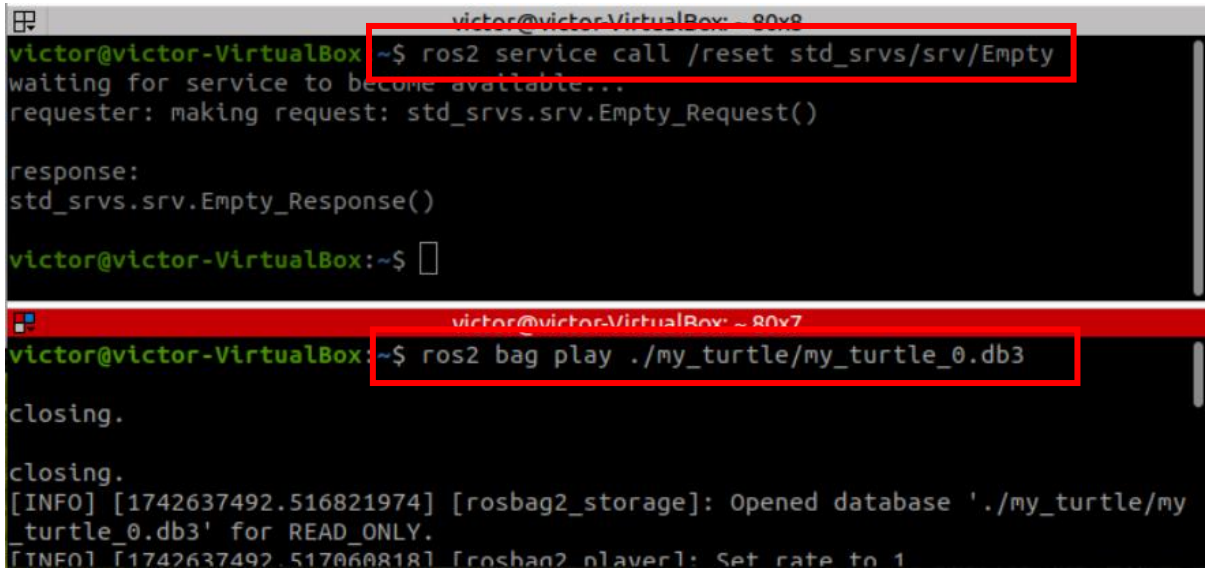
```
$ ros2 bag record -o my_topic -a
```

  - 'my_turtle'이라는 rosbag 파일의 정보를 확인

```
$ ros2 bag info my_turtle
```

  - 지정한 rosbag 파일을 재생

```
$ ros2 bag play my_turtle
```



※ CLI 후반부에 실습

ros2 extensions

- **extensions 명령어는 ros2cli 개발용으로 사용되는 명령어로, ROS2 CLI에 추가할 수 있는 확장(extensions) 목록을 보여주고 관리하는 역할**
  - 현재 설치된 extension의 간단한 목록을 표시

```
$ ros2 extensions
```

  - 로드에 실패했거나 호환되지 않는 extension도 표시

```
$ ros2 extensions -a
```

ros2 extension_points

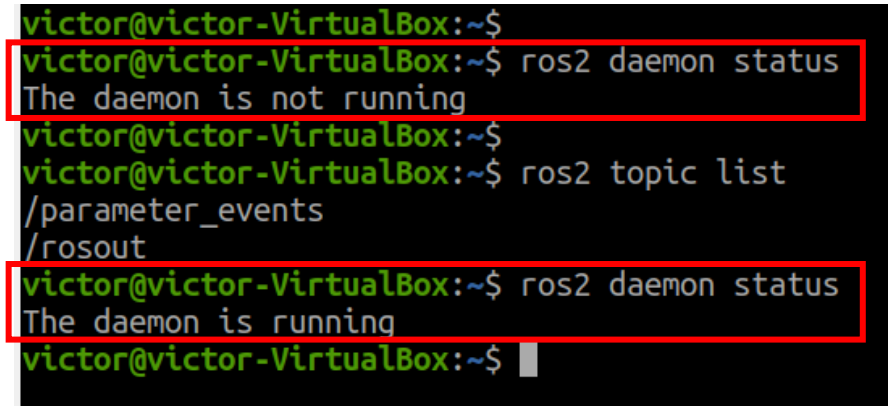
- extension_points 명령어는 ros2cli 개발용으로 사용되는 명령어로, extension points(확장 가능한 지점) 목록을 보여주는 역할
  - 현재 사용 가능한 extension points 목록을 표시

```
$ ros2 extension_points
```

ros2 daemon

- Daemon은 ROS2 도구들의 빠른 실행을 위해 도입된 툴로 주로 백그라운드에서 실행되는 프로그램이나 프로세스를 말함
- ROS2 Daemon 프로세스는 노드들을 발견하고 연결하는 역할을 하며, 특히 다음과 같은 명령어로 관리할 수 있음
  - Daemon을 시작
    ```
    $ ros2 daemon start
    ```
  - Daemon 상태를 확인
    ```
    $ ros2 daemon status
    ```
  - Daemon을 정지
    ```
    $ ros2 daemon stop
    ```

- 시스템에서 실행 중인 노드들의 정보를 유지하고 관리하는 역할
- 이를 통해 새로운 노드나 도구가 실행될 때, 데몬이 기존의 노드 정보를 제공하여 탐색 시간을 단축시키고 시스템의 전반적인 응답성을 향상시킴
- 통신, 관리, 서비스 제공 등의 역할



ros2 multicast

- multicast는 ROS2 DDS 테스트용으로 나온 명령어로 Multicast 송/수신 테스트에 사용되는 명령어
  - 단일 UDP 멀티 캐스트 패킷 수신 (송신된 패킷을 받기 전까지 대기함)

```
$ ros2 multicast receive
```

  - 단일 UDP 멀티 캐스트 패킷 송신 (새로운 터미널을 열어 송신 시 기존 터미널에서 수신 대기 중인 터미널이 패킷을 수신함)

```
$ ros2 multicast send
```

ros2 doctor

- doctor는 ROS2 설정 및 네트워크, 패키지 버전, RMW 등과 같은 ROS2 개발 환경의 잠재적 문제를 진단 및 점검하는 명령어
  - 네트워크 연결 확인

```
$ ros2 doctor hello
```

  - -r 옵션은 report를 의미하며 체크한 모든 아이템을 확인함

```
$ ros2 doctor -r
```

  - -rf 옵션은 report-fail을 의미하며 체크할 때 실패한 아이템을 확인함

```
$ ros2 doctor -rf
```

  - -iw 옵션은 include-warnings를 의미하며 경고성 아이템을 확인함

```
$ ros2 doctor -iw
```

ros2 wtf

- **wtf는 What's The Fast를 의미하며, 성능 최적화 및 데이터 전송 성능을 개선하는 도구**
- **데이터 전송 속도, 지연 시간, 대역폭 사용 효율 등을 개선**
  - 네트워크 연결 확인
    ```
    $ ros2 wtf hello
    ```
  - -r 옵션은 report를 의미하며 체크한 모든 아이템을 확인함
    ```
    $ ros2 wtf -r
    ```
  - -rf 옵션은 report-fail을 의미하며 체크할 때 실패한 아이템을 확인함
    ```
    $ ros2 wtf -rf
    ```
  - -iw 옵션은 include-warnings를 의미하며 경고성 아이템을 확인함
    ```
    $ ros2 wtf -iw
    ```

ros2 lifecycle

※ CLI 후반부에 실습

- Lifecycle은 노드의 수명 주기(lifecycle)를 관리하는 명령어
- 기본적으로 노드는 4개의 상태(state), Unconfigured, Inactive, Active, Finalized로 구분
  - 실행 중인 노드의 lifecycle 상태를 가져오기
    ```
    $ ros2 lifecycle get
    ```
  - /lc_talker 노드의 상태 전이가 가능한 lifecycle 목록을 출력
    ```
    $ ros2 lifecycle list /lc_talker
    ```
  - Lifecycle 상태를 가지고 있는 노드 목록을 출력
    ```
    $ ros2 lifecycle nodes
    ```
  - /lc_talker 노드의 lifecycle 상태를 configure 상태로의 전환을 트리거
    ```
    $ ros2 lifecycle set /lc_talker configure
    ```

- unconfigured : 초기 상태
- inactive : 설정은 됐지만, 동작은 멈춰진 상태
- active : 노드가 실제로 동작중
- finalized : 노드가 종료되고 리소스 정리된 상태
- errorprocessing : 에러 발생 후 에러 복구 중인 상태

- 예제를 위해 다음 명령어로 lifecycle_talker 예제 노드를 실행
- ros2 run lifecycle lifecycle_talker

ROS2 CLI

ROS2 CLI 실습

ros2 component

※ CLI 후반부에 실습

- **Component는 여러 노드를 단일 프로세스에서 실행하여 시스템의 효율성과 성능을 향상시키는 방식**
- **컴포넌트의 목록 조회, 로드, 언로드 등의 작업을 수행할 수 있음**
  - 실행 중인 컨테이너와 컴포넌트 목록 출력

```
$ ros2 component list
```

  - 지정 컨테이너 노드의 특정 컴포넌트 실행

```
$ ros2 component load /my_container composition composition::Talker
```

  - 표준 컨테이너 노드로 특정 컴포넌트 실행

```
$ ros2 component standalone composition composition::Talker
```

  - 사용 가능한 컴포넌트들의 목록 출력

```
$ ros2 component types
```

  - 지정 컴포넌트의 실행 중지

```
$ ros2 component unload /my_container 1
```

- 컴포넌트 노드는 여러 노드를 하나의 프로세스 내에서 실행할 수 있도록 설계된 ROS2의 기능
- 이를 통해 노드 간의 통신 오버헤드를 줄이고, 시스템의 자원 활용을 최적화할 수 있음
- 이러한 방식을 **컴포지션(Composition)**이라고 함

ROS2 CLI

ROS2 CLI 실습

ros2 security

※ CLI 후반부에 실습

- **Security는 SROS의 유틸리티로, DDS-Security를 ROS2에서 사용하기 위해 필요한 도구를 모아둔 것**
  - ros2 security는 **ROS 2에서 보안 기능을 설정하고 관리**하는 명령어
  - ROS 2는 DDS (Data Distribution Service)를 기반으로 하지만 기본적으로 보안이 비활성화
  - 이를 활성화하려면 **SROS 2 (Secure ROS 2)** 및 DDS-Security 표준을 사용해야 함.
  - 보안 기능을 활성화하면 **인증(Authentication), 암호화(Encryption), 액세스 제어(Access Control)** 등의 기능을 사용할 수 있음

[ROS 2 보안 기능 활용 예시]

1) 자율주행 로봇 데이터 보호

- 카메라, LiDAR 센서 데이터를 **암호화**하여 보호
- 외부에서 **허가되지 않은 노드**가 데이터를 읽거나 수정하지 못하도록 설정

2) 산업용 로봇 시스템 보안

- 로봇 제어 명령을 허가된 노드에서만 보낼 수 있도록 제한
- 공장 네트워크에서 보안이 유지되도록 암호화된 통신 사용

3) 클라우드 연동 IoT 시스템 보안 강화

- 클라우드에서 ROS 2 기반 로봇과 안전하게 통신
- TLS 및 DDS 보안 프로토콜을 적용하여 외부 공격으로부터 보호

ROS2 CLI의 빠른 실행

ROS2 CLI의 빠른 실행

alias

- 홈 폴더(~/)의 .bashrc 파일에 자주 사용하는 ROS2 CLI 명령어를 단축 명령어로 지정해 두면 특정 ROS2 CLI를 빠르게 실행 가능
  - ~/.bashrc 파일에 아래 명령어를 추가

```
alias tn='ros2 run turtlesim turtlesim_node'
alias tt='ros2 run turtlesim turtle_teleop_key'

alias testpub='ros2 run demo_nodes_cpp talker'
alias testsub='ros2 run demo_nodes_cpp listener'

alias rt='ros2 topic list'
alias re='ros2 topic echo'
alias rn='ros2 node list'
```

※ 터미널창 2개 이상 띄우고 source ~/.bashrc 해보기

ROS2 CLI의 빠른 실행

ROS2 CLI의 빠른 실행

alias

- 홈 폴더(~/)의 .bashrc 파일에 자주 사용하는 ROS2 CLI 명령어를 단축 명령어로 지정해 두면 특정 ROS2 CLI를 빠르게 실행 가능
  - ~/.bashrc 파일을 저장한 다음 현재 셀 세션에 설정을 적용

```
$ source ~/.bashrc
```

  - 앞서 지정한 단축키를 이용하여 명령어를 빠르게 사용 가능

```
user@user:~$ rt
/image
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
user@user:~$ rn
/rqt_gui_py_node_57947
/showimage
/teleop_turtle
/turtle1/rqt_example
/turtlesim
```

```
user@user:~$ re /turtle1/pose
x: 5.403021812438965
y: 4.110311508178711
theta: -1.5807411670684814
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.403021812438965
y: 4.110311508178711
theta: -1.5807411670684814
linear_velocity: 0.0
angular_velocity: 0.0
---
```

CLI에서 arguments 사용하기

CLI에서 arguments 사용하기

ROS arguments

- ROS arguments는 주로 run 또는 launch 명령어와 같은 ROS2 실행 명령어와 함께 사용되며 “--ros-args” 옵션을 통해 지정

```
$ ros2 run 패키지이름 노드이름 --ros-args (ROS argument)
```

- 많이 사용되는 ROS arguments
  - -r __ns:=사용할 네임스페이스
  - -r __node:=변경할 노드 이름
  - -r 본래의 토픽/서비스/액션명:=변경할 이름
  - -p 파라미터 이름:=변경할 파라미터 값
  - --params-file 파라미터 파일

CLI에서 arguments 사용하기

CLI에서 arguments 사용하기

ROS arguments 예제

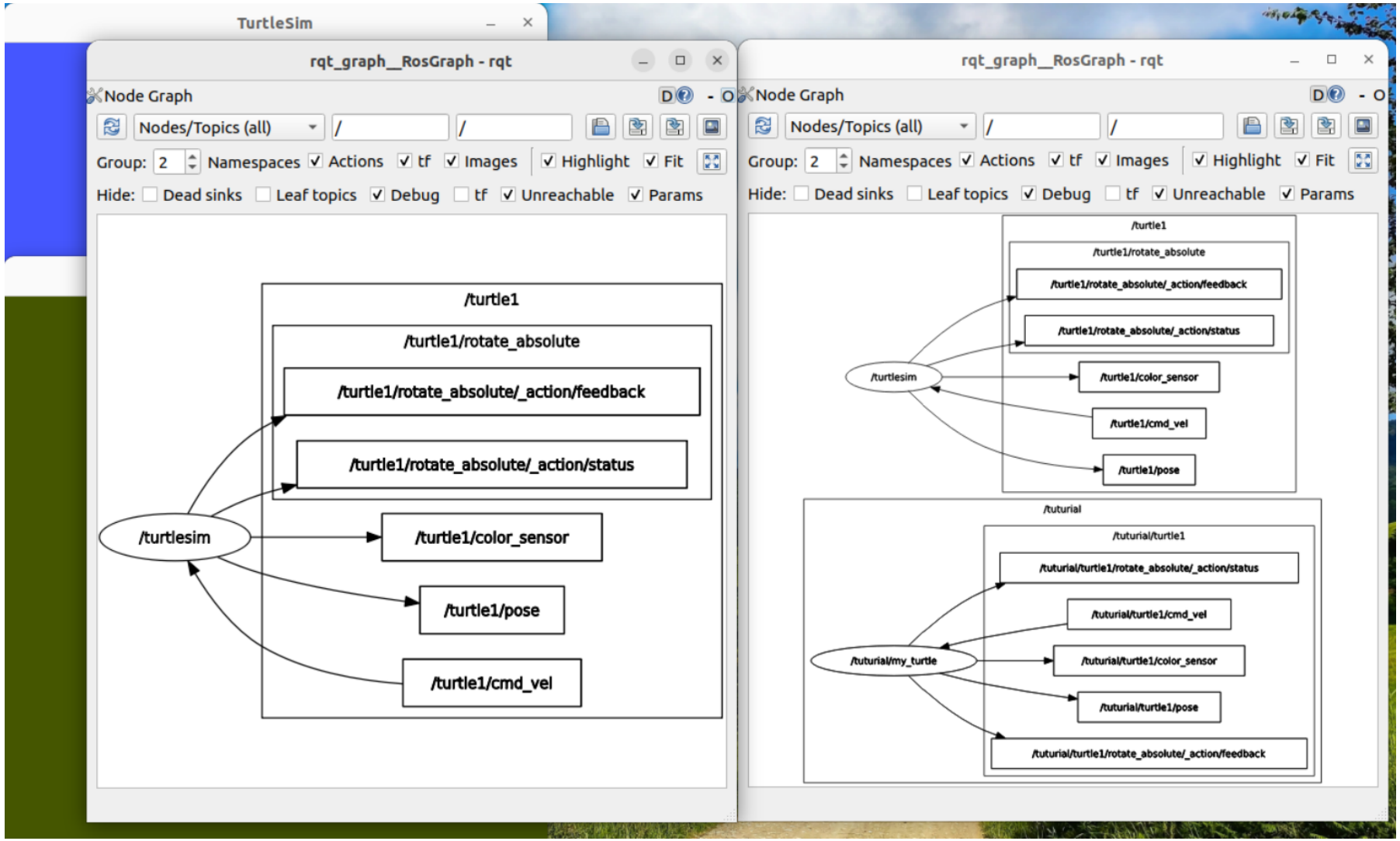
- **아래와 같은 설정으로 파라미터 설정**
  - 네임스페이스: /tutorial
  - 변경할 노드 이름: my_turtle
  - turtle1/cmd_vel을 cmd_vel로 퍼블리시 되도록 수정
  - background_b 파라미터를 0으로 변경

```
user@user:~$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/tutorial -r __node:=my_turtle -r turtle1/cmd_ve
l:=cmd_vel -p background_b:=0
```

CLI에서 arguments 사용하기

CLI에서 arguments 사용하기

[TURTLESIM 1마리 상태에서 rqt_graph 이후 앞 페이지 command 실행 후 rqt_graph확인



CLI에서 arguments 사용하기

CLI에서 arguments 사용하기

ROS arguments 예제

- **yaml 파일로 파라미터 재설정**
  - 아래와 같은 yaml파일 생성

```yaml
turtlesim:
  ros__parameters:
    background_b: 0
    background_g: 250
    background_r: 250
    use_sim_time: false
```

  - yaml파일의 설정을 이용하여 turtlesim 실행

```
$ ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml
```

CLI에서 arguments 사용하기

CLI에서 arguments 사용하기

신규 ROS2 cli 작성법

신규 ROS2 cli 작성법

소개

- 앞에서 ROS2 CLI 명령어의 개념, 사용법 및 종류에 대해 학습하였음
- 지금부터는 새로운 ROS2 CLI 명령어를 생성하는 방법을 탐구하고자 함

신규 ROS2 cli 작성법

- ros2 env라는 기존에 없던 ROS2 CLI를 만들기

※ 필요한 파일 : ros2env.zip

틀린그림 찾기

```
victor@victor-VirtualBox:~/ros2_ws$ ros2 -h
usage: ros2 [-h] [--use-python-default-buffering]
            Call `ros2 <command> -h` for more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

options:
  -h, --help            show this help message and exit
  --use-python-default-buffering
                        Do not force line buffering in stdout and instead use
                        the python default buffering, which might be affected
                        by PYTHONUNBUFFERED/-u and depends on whatever stdout
                        is interactive or not

Commands:
  action     Various action related sub-commands
  bag        Various rosbag related sub-commands
  component  Various component related sub-commands
  daemon     Various daemon related sub-commands
  doctor     Check ROS setup and other potential issues
  interface  Show information about ROS interfaces
  launch     Run a launch file
  lifecycle  Various lifecycle related sub-commands
  multicast  Various multicast related sub-commands
  node       Various node related sub-commands
  param      Various param related sub-commands
  pkg        Various package related sub-commands
  run        Run a package specific executable
  security   Various security related sub-commands
  service    Various service related sub-commands
  topic      Various topic related sub-commands
  wtf        Use `wtf` as alias to `doctor`

  Call `ros2 <command> -h` for more detailed usage.
victor@victor-VirtualBox:~/ros2_ws$
```

```
victor@victor-VirtualBox:~/ros2_ws$ ros2 -h
usage: ros2 [-h] [--use-python-default-buffering] Call `ros2 <command> -h` for
 more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

options:
  -h, --help            show this help message and exit
  --use-python-default-buffering
                        Do not force line buffering in stdout and instead use
the python default buffering, which might be affected by
                        PYTHONUNBUFFERED/-u and depends on whatever stdout is
interactive or not

Commands:
  action     Various action related sub-commands
  bag        Various rosbag related sub-commands
  component  Various component related sub-commands
  daemon     Various daemon related sub-commands
  doctor     Check ROS setup and other potential issues
  env        Various env related sub-commands
  interface  Show information about ROS interfaces
  launch     Run a launch file
  lifecycle  Various lifecycle related sub-commands
  multicast  Various multicast related sub-commands
  node       Various node related sub-commands
  param      Various param related sub-commands
  pkg        Various package related sub-commands
  run        Run a package specific executable
  security   Various security related sub-commands
  service    Various service related sub-commands
  topic      Various topic related sub-commands
  wtf        Use `wtf` as alias to `doctor`

  Call `ros2 <command> -h` for more detailed usage.
victor@victor-VirtualBox:~/ros2_ws$
```

✓ 실행 예제

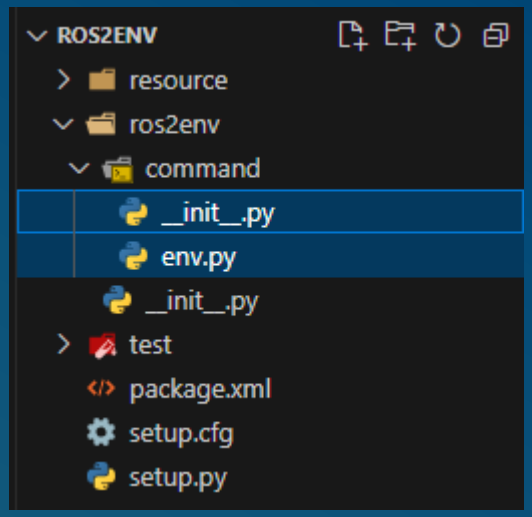
1. ros2 폴더의 src폴더로 이동하여 ros2env 패키지 생성

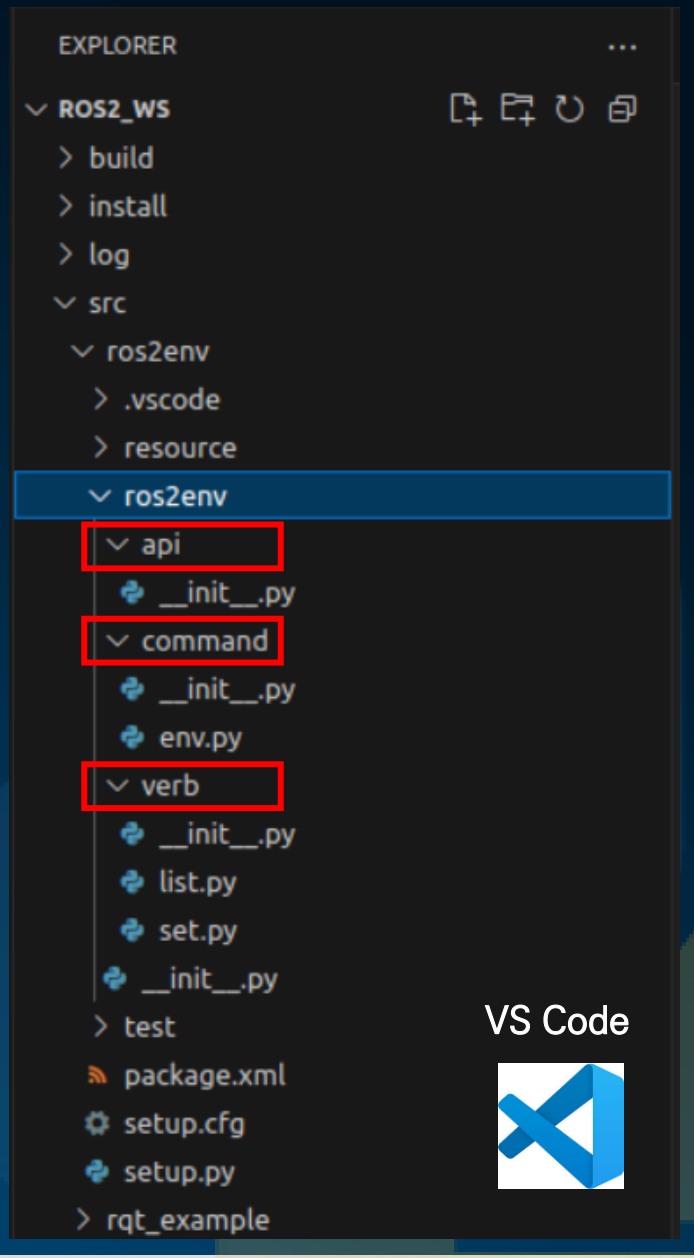
```
$ cd ~/ros2_ws/src/
$ ros2 pkg create --build-type ament_python ros2env
```

2. ros2env 폴더로 이동하여 vscode 실행

```
$ cd ros2env/
$ code .
```

3. ros2env 폴더 안에 command 폴더 생성 후 env.py 파일과 __init__.py 파일 생성





Build 하고 실행해보기

```
victor@victor-VirtualBox:~/ros2_ws$ ros2 env list
ROS_VERSION         = 2
ROS_DISTRO          = humble
ROS_PYTHON_VERSION = 3
ROS_DOMAIN_ID       = None
RMW_IMPLEMENTATION = None

victor@victor-VirtualBox:~/ros2_ws$ ros2 env list -a
ROS_VERSION         = 2
ROS_DISTRO          = humble
ROS_PYTHON_VERSION = 3
ROS_DOMAIN_ID       = None
RMW_IMPLEMENTATION = None

victor@victor-VirtualBox:~/ros2_ws$ ros2 env list -r
ROS_VERSION         = 2
ROS_DISTRO          = humble
ROS_PYTHON_VERSION = 3

victor@victor-VirtualBox:~/ros2_ws$ ros2 env list -d
ROS_DOMAIN_ID       = None
RMW_IMPLEMENTATION = None

victor@victor-VirtualBox:~/ros2_ws$
```

← DOMAIN_ID 바꿔서 실행해보기

```
sparkx@sparkx-550XDA:~/ros2_ws$ echo $ROS_DOMAIN_ID
214
sparkx@sparkx-550XDA:~/ros2_ws$ ros2 env set ROS_DOMAIN_ID 99
[Changed ROS environment variable]:
ROS_DOMAIN_ID = 99

[Current ROS environment variable]:
ROS_VERSION         = 2
ROS_DISTRO          = humble
ROS_PYTHON_VERSION = 3
ROS_DOMAIN_ID       = 99
RMW_IMPLEMENTATION = None
```

✓실행 예제 – env.py

1. env.py: 인터페이스 (CLI)에서 확장 기능을 추가하는 EnvCommand 클래스를 정의

← CLI의 “env” 메인 명령을 정의($ros2 env list or $ros2 env set)

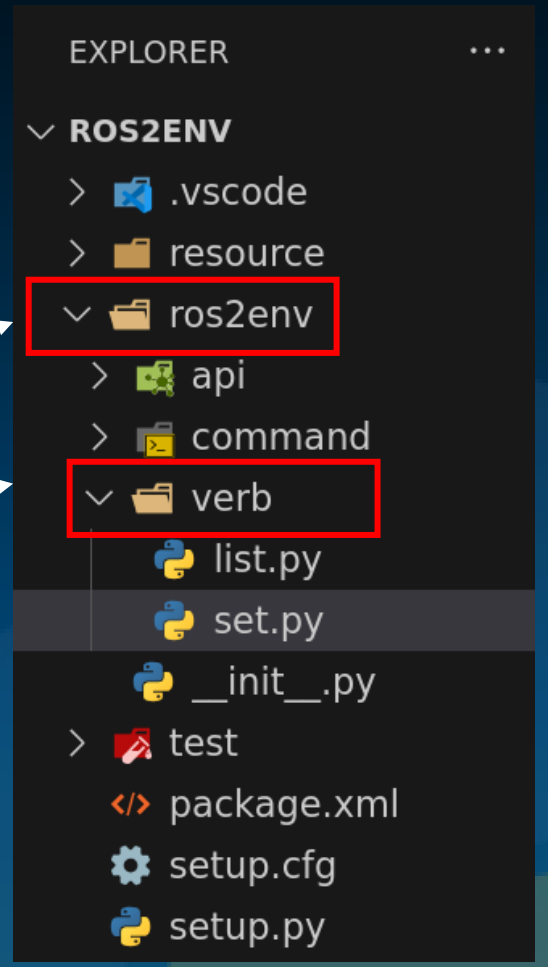
```python
from ros2cli.command import add_subparsers_on_demand
from ros2cli.command import CommandExtension


class EnvCommand(CommandExtension):
    def add_arguments(self, parser, cli_name):
        self._subparser = parser

        add_subparsers_on_demand(
            parser, cli_name, "_verb", "ros2env.verb", required=False
        )

    def main(self, *, parser, args):
        if not hasattr(args, "_verb"):
            self._subparser.print_help()
            return 0

        extension = getattr(args, "_verb")

        return extension.main(args=args)
```

✓실행 예제 – env.py

2. add_arguments 함수

- 파서(parser)에 서브 명령어를 동적으로 추가하여 확장 가능한 구조를 만듦

```python
def add_arguments(self, parser, cli_name):
    self._subparser = parser

    # add arguments and sub-commands of verbs
    add_subparsers_on_demand(
        parser, cli_name, "_verb", "ros2env.verb", required=False
    )
```



ros2env.verb는 서브 커맨드 들이 모여 있는 python Module 경로임

- ros2env.verb.list
- ros2env.verb.set

✓실행 예제 – env.py

3. add_subparsers_on_demand 함수

- ROS2 CLI(Command Line Interface)에서 서브 명령어(보통 'verb'라 불림)를 필요에 따라 동적으로 로드하고 추가하는 역할을 함
- ros2env.verb아래에서 플러그인(verb)를 찾고 인스턴스화하고 argparse의 subparser로 등록해주는 함수

[주요 인자]

- parser: 메인 명령어의 파서 객체. 이 파서 객체에 서브 명령어를 추가하여 명령어 라인에서 사용할 수 있도록 함
- cli_name: CLI 도구의 이름을 문자열로 전달(ros2 env 같은 상위 명령어 이름)
- attribute_name(_verb): 서브 명령어(verb)의 이름을 속성으로 사용하여 이 속성을 기준으로 실행할 명령어를 결정
- module_name(ros2env.verb): 서브 명령어(verb) 구현이 들어있는 모듈의 이름
- required: 서브 명령어가 필수인지 여부를 결정. False로 설정할 경우 서브 명령어가 없을 때 기본적으로 도움말이 출력됨

```python
def add_arguments(self, parser, cli_name):
    self._subparser = parser

    # add arguments and sub-commands of verbs
    add_subparsers_on_demand(     ← ros2env.verb로 부터 필요한 시점에서 command를 가져옴
        parser, cli_name, "_verb", "ros2env.verb", required=False
    )
```

✓실행 예제 – env.py

4. main 메서드

- 서브 명령어가 주어지지 않은 경우 도움말을 출력하고, 주어진 경우 해당 서브 명령어의 main 메서드를 호출하여 실행

```
    def main(self, *, parser, args):
        if not hasattr(args, "_verb"):
            # 서브 명령어가 전달되지 않았을 때, 도움말을 출력
            self._subparser.print_help()
            return 0
        # 속성 `_verb`를 기준으로 서브 명령어(verb)를 가져옴
        extension = getattr(args, "_verb")
        # 해당 서브 명령어의 main 메서드를 호출하여 실행
        return extension.main(args=args)
```

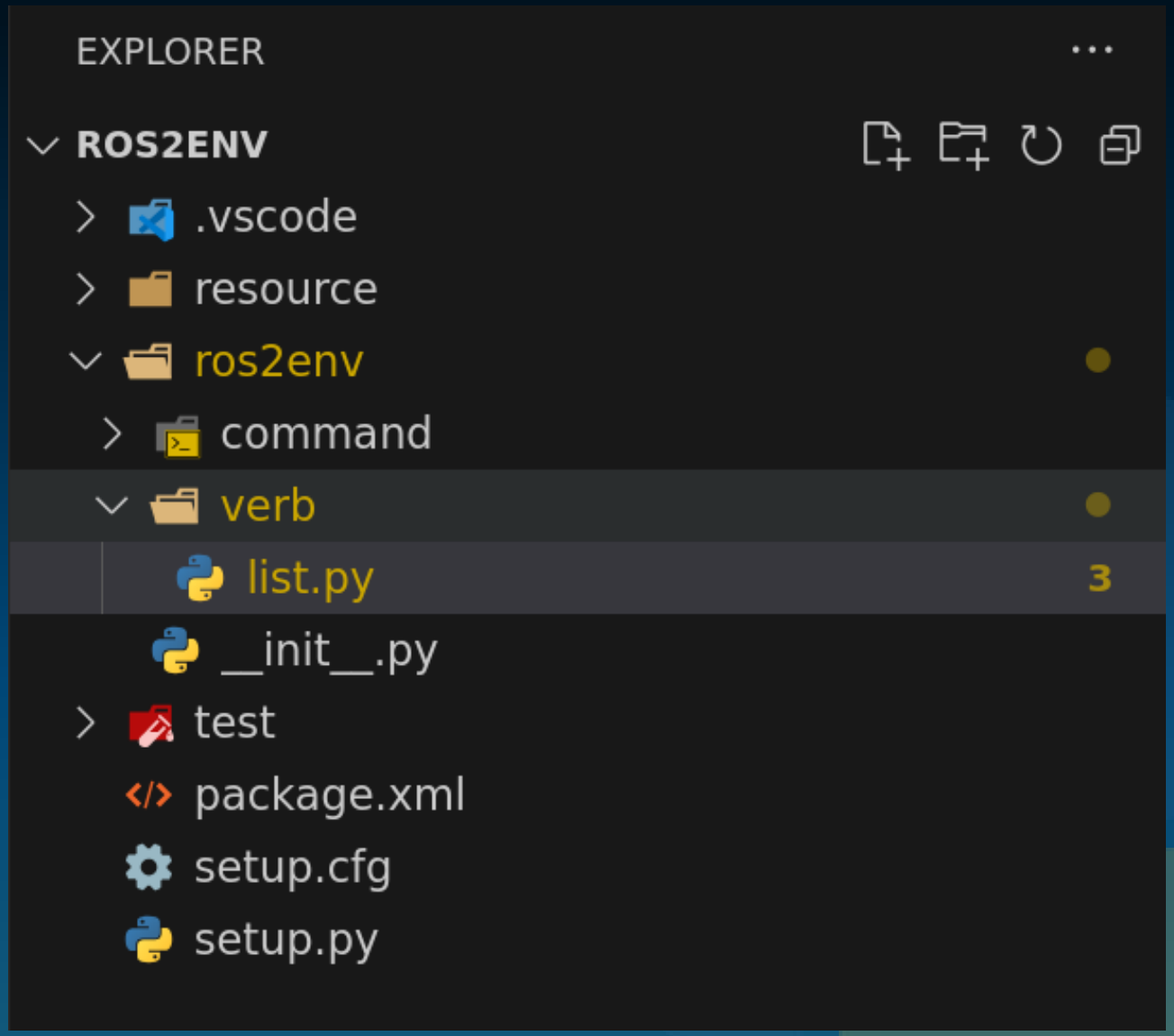
← args._verb에 사용자가 입력한 하위 명령어(verb)가 들어옴. 서브 커맨드 입력 여부 검사

← 만약 사용자가 서브 커맨드를 입력하지 않았다면 help 출력

← ros2 env list나 ros2 env set에서 list와 set

✓실행 예제 – list.py

1. verb폴더 생성 후 안에 list.py파일 생성



✓실행 예제 – list.py

2. list.py: ros2env 패키지의 일부로, ROS2 환경 변수들을 출력하는 ListVerb 클래스를 정의

```python
from ros2env.api import get_all_env_list
from ros2env.api import get_dds_env_list
from ros2env.api import get_ros_env_list
from ros2env.verb import VerbExtension


class ListVerb(VerbExtension):

    def add_arguments(self, parser, cli_name):
        parser.add_argument(
            "-a",
            "--all",
            action="store_true",
            help="Display all environment variables.",
        )
        parser.add_argument(
            "-r",
            "--ros-env",
            action="store_true",
            help="Display the ROS environment variables.",
        )
        parser.add_argument(
            "-d",
            "--dds-env",
            action="store_true",
            help="Display the DDS environment variables.",
        )
```

```python
def main(self, *, args):
        if args.ros_env:        ← ros2 env –r
            message = get_ros_env_list()
        elif args.dds_env:      ← ros2 env –d
            message = get_dds_env_list()
        else:
            message = get_all_env_list()
        print(message)                  ← ros2 env –a
```

✓실행 예제 – list.py

3. code import
   - get_all_env_list, get_dds_env_list, get_ros_env_list : 각각 모든 환경 변수, DDS 관련 환경 변수, ROS 관련 환경 변수를 반환하는 함수들
     * DDS: 네트워크에 연결된 여러 장치들이 데이터를 주고받을 수 있게 해주는 통신 기술
   - VerbExtension: ROS2 CLI 명령어 확장 기능을 제공하는 기본 클래스

```
from ros2env.api import get_all_env_list
from ros2env.api import get_dds_env_list
from ros2env.api import get_ros_env_list
from ros2env.verb import VerbExtension
```

※ 왜 ros2env.api 모듈로 별도 분리해서 사용하는지 확인해보자!!!

✓실행 예제 – list.py

4. ListVerb 클래스

- ros2env와 관련된 명령어로 환경 변수를 리스트로 출력할 수 있는 기능을 제공

```
class ListVerb(VerbExtension):
    ...
```

✓실행 예제 – list.py

5. add_arguments 함수

- ros2 list 명령어를 실행할 때 사용할 수 있는 옵션을 정의

```
    def add_arguments(self, parser, cli_name):
        parser.add_argument(
            "-a",
            "--all",
            action="store_true",
            help="Display all environment variables.",
        )
        parser.add_argument(
            "-r",
            "--ros-env",
            action="store_true",
            help="Display the ROS environment variables.",
        )
        parser.add_argument(
            "-d",
            "--dds-env",
            action="store_true",
            help="Display the DDS environment variables.",
        )
```

- -a, --all: 모든 환경 변수를 출력하도록 설정하는 플래그
- -r, --ros-env: ROS 관련 환경 변수만 출력하도록 설정하는 플래그
- -d, --dds-env: DDS 관련 환경 변수만 출력하도록 설정하는 플래그
- 각 옵션은 action="store_true"로 설정되어, 명령어에 옵션을 포함할 경우 True 값을 가지며, 그렇지 않을 경우 False 값을 가짐
- help 매개변수는 각 옵션의 설명을 제공하여 ros2 list --help로 명령어에 대한 도움말을 볼 때 사용됨

✓실행 예제 – list.py

6. main 메서드

- 명령어 실행의 main 로직을 담당
- args 인자로 전달된 옵션 값에 따라 특정 환경 변수 리스트를 가져옴

```
def main(self, *, args):
    if args.ros_env:
        message = get_ros_env_list()
    elif args.dds_env:
        message = get_dds_env_list()
    else:
        message = get_all_env_list()
    print(message)
```

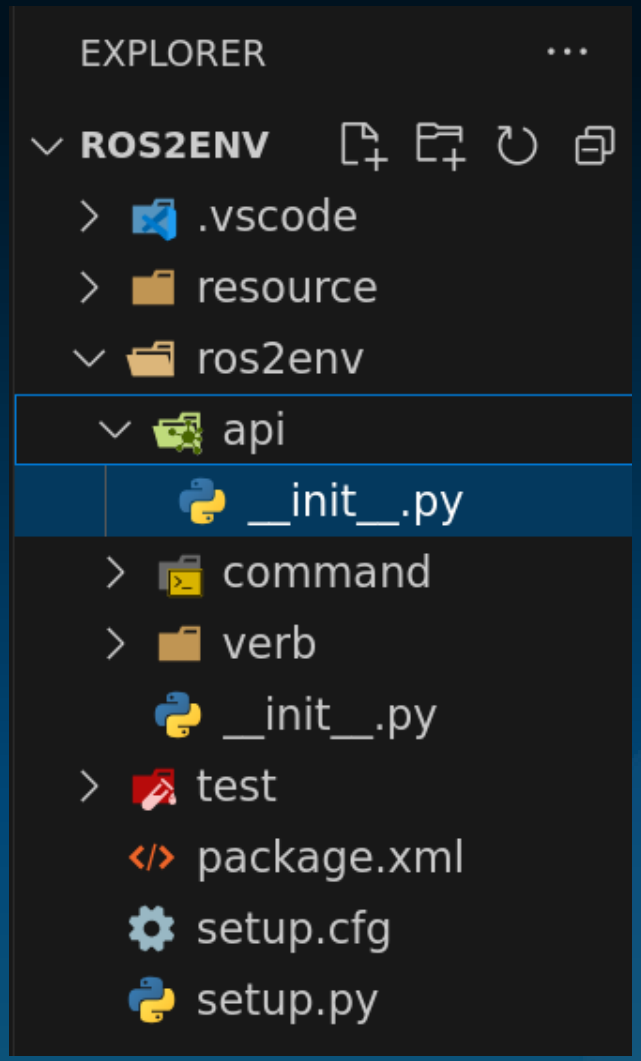
- args.ros_env가 True일 경우, get_ros_env_list() 함수를 호출하여 ROS 관련 환경 변수를 가져옴
- args.dds_env가 True일 경우, get_dds_env_list() 함수를 호출하여 DDS 관련 환경 변수를 가져옴
- 두 옵션 모두 False일 경우(즉, --all 옵션이거나 아무 옵션도 사용하지 않은 경우), get_all_env_list()를 호출하여 모든 환경 변수를 가져옴

7. 예시 사용법

- 모든 환경 변수 출력: ros2 list –a
- ROS 관련 환경 변수 출력: ros2 list –r
- DDS 관련 환경 변수 출력: ros2 list –d

✓실행 예제 – api/__init__.py

1. api 폴더 생성 후 __init__.py 파일 생성



✓실행 예제 – api/__init__.py

1. __init__.py: ROS2와 관련된 환경 변수를 읽고 설정

```python
import os


def get_ros_env_list():
    ros_version = os.getenv("ROS_VERSION", "None")
    ros_distro = os.getenv("ROS_DISTRO", "None")
    ros_python_version = os.getenv("ROS_PYTHON_VERSION", "None")
    ros_env_list = "ROS_VERSION         = {0}\n\
ROS_DISTRO          = {1}\n\
ROS_PYTHON_VERSION = {2}\n".format(
        ros_version, ros_distro, ros_python_version
    )
    return ros_env_list


def get_dds_env_list():
    ros_domain_id = os.getenv("ROS_DOMAIN_ID", "None")
    rmw_implementation = os.getenv("RMW_IMPLEMENTATION", "None")
    dds_env_list = "ROS_DOMAIN_ID       = {0}\n\
RMW_IMPLEMENTATION = {1}\n".format(
        ros_domain_id, rmw_implementation
    )
    return dds_env_list
```

```python
def get_all_env_list():
    ros_env_list = get_ros_env_list()
    dds_env_list = get_dds_env_list()
    all_env_list = ros_env_list + dds_env_list
    return all_env_list


def set_ros_env(env_name, env_value):
    os.environ[env_name] = env_value
    value = os.getenv(env_name, "None")
    return "{0} = {1}".format(env_name, value)
```

✓실행 예제 – api/__init__.py

2. get_ros_env_list 함수

- ROS2와 관련된 환경 변수인 ROS_VERSION, ROS_DISTRO, ROS_PYTHON_VERSION 값을 가져옴
- os.getenv() 함수를 사용하여 각 환경 변수를 읽어 옴
- 세 환경 변수를 문자열로 포맷팅하여 반환

```
def get_ros_env_list():
    # os.getenv() 함수를 사용하여 ROS와 관련된 환경 변수를 가져옴
    # 변수값이 존재하지 않으면 기본값 'None'을 반환
    ros_version = os.getenv('ROS_VERSION', 'None')
    ros_distro = os.getenv('ROS_DISTRO', 'None')
    ros_python_version = os.getenv('ROS_PYTHON_VERSION', 'None')  # ROS에서 사용하는 파이썬 버전

    # 가져온 변수들을 포맷팅하여 문자열로 반환
    ros_env_list = 'ROS_VERSION         = {0}\n\
ROS_DISTRO          = {1}\n\
ROS_PYTHON_VERSION = {2}\n'.format(ros_version, ros_distro, ros_python_version)
    return ros_env_list
```

✓실행 예제 – api/__init__.py

3. get_dds_env_list 함수

- 이 함수는 DDS와 관련된 환경 변수인 ROS_DOMAIN_ID와 RMW_IMPLEMENTATION 값을 가져옴
- ROS2에서 DDS를 설정하기 위한 역할을 함
- 환경 변수 값이 없을 경우 'None'을 반환하도록 하여, 존재 여부를 확인

*** RMW_IMPLEMENTATION: ROS2가 데이터를 주고받을 때 어떤 방식을 사용할지 정하는 환경 변수**

```
def get_dds_env_list():
    # os.getenv() 함수를 사용하여 DDS와 관련된 환경 변수를 가져옴
    # 변수값이 존재하지 않으면 기본값 'None'을 반환
    ros_domain_id = os.getenv('ROS_DOMAIN_ID', 'None')        # ROS 네트워크 도메인 ID
    rmw_implementation = os.getenv('RMW_IMPLEMENTATION', 'None')  # DDS 미들웨어 구현

    # 가져온 변수들을 포맷팅하여 문자열로 반환
    dds_env_list = 'ROS_DOMAIN_ID       = {0}\n\
RMW_IMPLEMENTATION = {1}\n'.format(ros_domain_id, rmw_implementation)
    return dds_env_list
```

✓실행 예제 – api/__init__.py

4. get_all_env_list 함수

- 이 함수는 앞서 설명한 get_ros_env_list()와 get_dds_env_list()를 호출하여, 모든 ROS 및 DDS 관련 환경 변수를 가져옴
- 두 함수의 반환 값을 결합하여 모든 환경 변수를 하나의 문자열로 반환

```
def get_all_env_list():
    ros_env_list = get_ros_env_list()
    dds_env_list = get_dds_env_list()

    # 두 리스트를 합쳐서 하나의 문자열로 반환
    all_env_list = ros_env_list + dds_env_list
    return all_env_list
```

✓실행 예제 – api/__init__.py

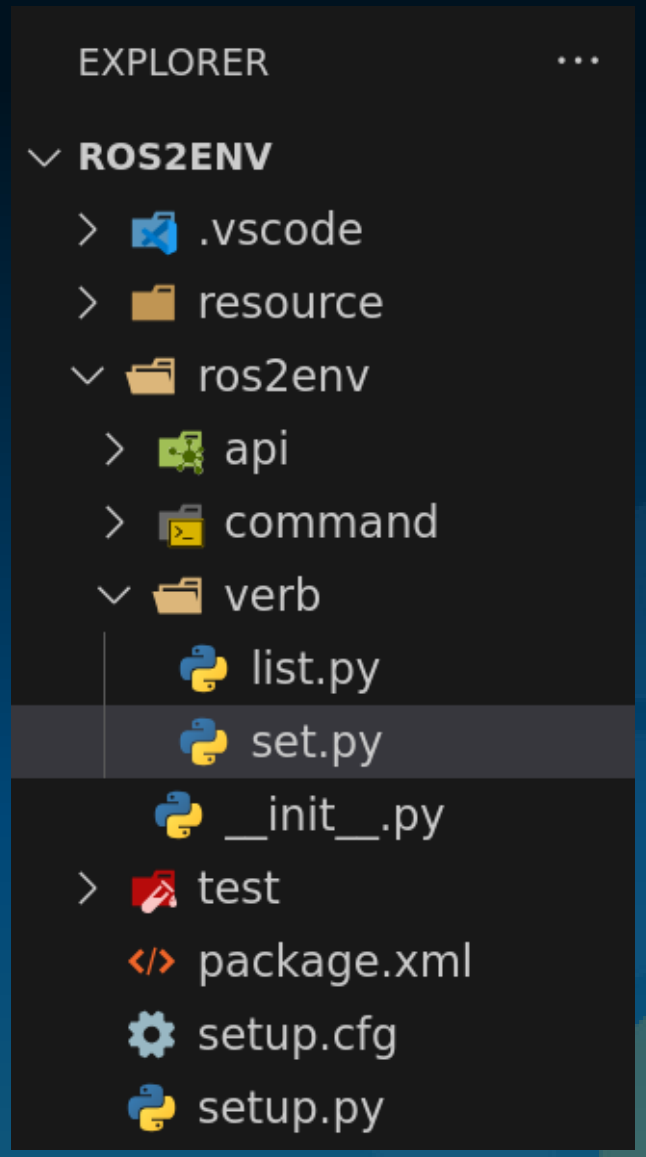
5. set_ros_env 함수

- 이 함수는 env_name과 env_value라는 두 인자를 받아 해당 환경 변수를 설정
- os.environ 딕셔너리에 값을 직접 할당하여 환경 변수를 설정하며, os.getenv()로 설정된 값을 다시 확인하여 반환
- 환경 변수 설정 후 반환 값을 통해 변수명과 설정된 값을 문자열 형태로 확인 가능

```python
def set_ros_env(env_name, env_value):
    # os.environ을 사용하여 환경 변수를 설정합니다.
    os.environ[env_name] = env_value

    # 설정한 변수의 값을 확인하여 문자열로 반환합니다.
    value = os.getenv(env_name, 'None')
    return '{0} = {1}'.format(env_name, value)
```

✓실행 예제 – set.py

1. 이전에 만들어 두었던 verb 폴더 안에 set.py 파일 생성



✓실행 예제 – set.py

2. set.py: ros2env 패키지를 사용하여 ROS2 Humble에서 환경 변수를 설정하고 출력하는 기능을 제공

```python
from ros2env.api import get_all_env_list
from ros2env.api import set_ros_env
from ros2env.verb import VerbExtension


class SetVerb(VerbExtension):
    """Set ROS environment variables."""

    def add_arguments(self, parser, cli_name):
        parser.add_argument("env_name", help="Name of the environment variable")
        parser.add_argument("value", help="Value of the environment variable")

    def main(self, *, args):
        if args.env_name or args.value:
            message = set_ros_env(args.env_name, args.value)
            print("[Changed ROS environment variable]:")
            print(message)
        message = get_all_env_list()
        print("\n[Current ROS environment variable]:"
        print(message)
```

✓실행 예제 – set.py

3. code import

- get_all_env_list: 모든 ROS와 DDS 관련 환경 변수를 가져오는 함수. 환경 변수를 조회하여 현재 설정 상태 확인 가능
- set_ros_env: 특정 ROS 환경 변수를 설정하는 함수. 환경 변수의 이름과 값을 입력하여 새로운 설정 가능
- VerbExtension: ROS2 CLI 확장 명령어를 구현하기 위한 기본 클래스. 이 클래스는 ros2 〈verb〉와 같은 형식으로 명령어를 확장할 수 있도록 지원하며, ROS2 Humble에서 SetVerb 명령어를 추가하는데 사용됨

```
from ros2env.api import get_all_env_list  # 모든 환경 변수를 가져오는 함수
from ros2env.api import set_ros_env  # 특정 환경 변수를 설정하는 함수
from ros2env.verb import VerbExtension  # ROS 2 CLI 확장 명령어의 기본 클래스
```

✓실행 예제 – set.py

4. SetVerb 클래스

- ROS 환경 변수를 설정하는 데 사용되는 클래스로, VerbExtension을 상속받아 ROS2 명령어 확장을 구현함

```
class SetVerb(VerbExtension):
    ...
```

✓실행 예제 – set.py

5. add_arguments 함수

- ROS2 명령어에 필요한 옵션과 인수를 정의
- parser.add_argument() 함수를 통해 두 개의 필수 인자를 추가
  - ✓ env_name: 설정할 환경 변수의 이름을 입력. ROS_VERSION, ROS_DISTRO 등의 변수명이 해당됨
  - ✓ value: 환경 변수에 설정할 값. 사용자가 ros2 set 〈env_name〉 〈value〉 형식으로 입력한 값이 여기에 전달됨
- help 매개변수는 각 인자의 설명을 제공. ros2 set --help 명령어로 실행됨

```
    def add_arguments(self, parser, cli_name):
        parser.add_argument("env_name", help="Name of the environment variable")
        parser.add_argument("value", help="Value of the environment variable"
```

✓실행 예제 – set.py

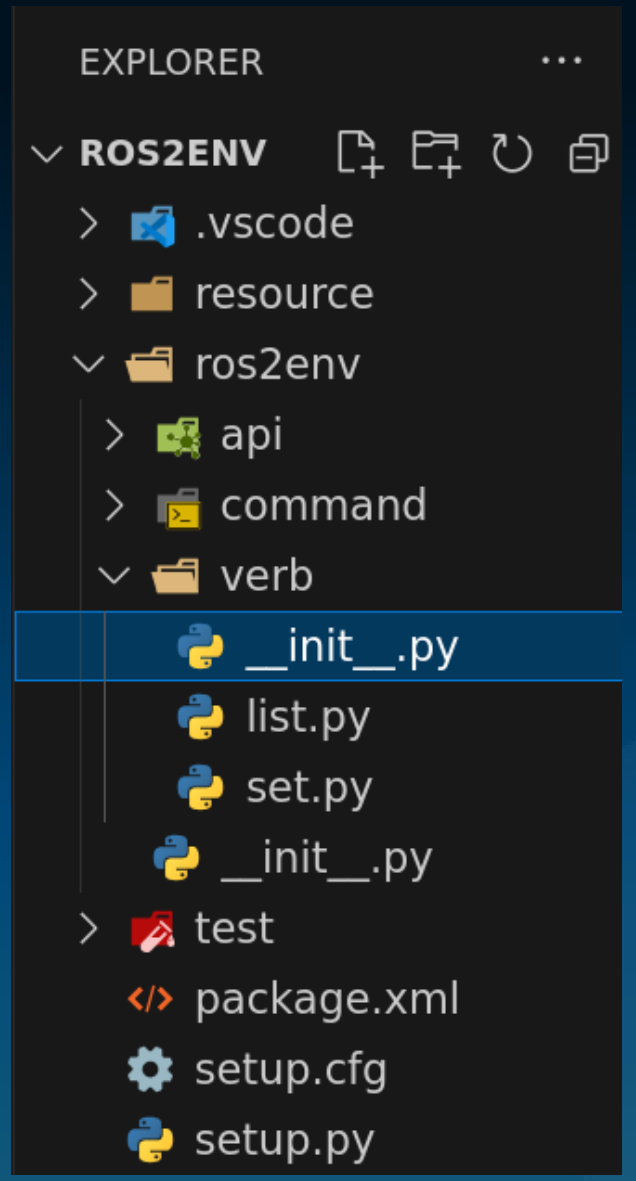
6. main 함수

- 실행의 주 로직을 담당하는 함수. args 인자로 전달된 env_name과 value 값을 사용하여 특정 ROS 환경 변수를 설정
- env_name 또는 value 값이 있는 경우에만 환경 변수를 설정
  - 환경 변수 설정: set_ros_env(args.env_name, args.value) 함수가 호출되어 해당 환경 변수에 값을 설정
- get_all_env_list() 함수를 호출하여 모든 환경 변수의 현재 상태를 가져오고, [Current ROS environment variable]: 메시지와 함께 출력

```python
def main(self, *, args):
    # 환경 변수 이름과 값이 전달되었는지 확인
    if args.env_name or args.value:
        # set_ros_env 함수를 호출하여 환경 변수를 설정하고 결과 메시지를 받아옴.
        # args.env_name에 환경 변수 이름, args.value에 해당 변수의 값을 설정.
        message = set_ros_env(args.env_name, args.value)
        print("[Changed ROS environment variable]:")
        print(message)

    # get_all_env_list 함수를 호출하여 현재 모든 ROS와 DDS 관련 환경 변수를 가져옴
    message = get_all_env_list()
    print("\n[Current ROS environment variable]:")
    print(message)
```

✓실행 예제 – verb/__init__.py

1. verb 폴더 안에 __init__.py 파일 생성



✓실행 예제 – verb/__init__.py

2. __init__.py: ROS2 Humble에서 env 명령어에 대한 확장 포인트를 정의하기 위한 파일로 ROS2 CLI 확장을 위한 기본 템플릿으로 사용됨

```python
from ros2cli.plugin_system import PLUGIN_SYSTEM_VERSION
from ros2cli.plugin_system import satisfies_version


class VerbExtension:
    NAME = None
    EXTENSION_POINT_VERSION = "0.1"

    def __init__(self):
        super(VerbExtension, self).__init__()
        satisfies_version(PLUGIN_SYSTEM_VERSION, "^0.1")

    def add_arguments(self, parser, cli_name):
        pass

    def main(self, *, args):
        raise NotImplementedError()
```

✓실행 예제 – verb/__init__.py

3. code import
   - PLUGIN_SYSTEM_VERSION: 현재 사용 중인 ROS2 CLI 플러그인 시스템의 버전을 나타냄
   - satisfies_version: 플러그인 시스템의 버전과 확장의 버전이 호환되는지 검사하는 함수. 특정 버전 규칙을 따르는지 확인하여, 버전 불일치로 인한 오류를 방지

```
from ros2cli.plugin_system import PLUGIN_SYSTEM_VERSION
from ros2cli.plugin_system import satisfies_version
```

✓실행 예제 – verb/__init__.py

4. VerbExtension 클래스

- ROS2 CLI 확장 시스템에서 명령어 확장을 위한 기본 클래스로 사용됨
- NAME : 확장의 이름을 설정할 때 사용되는 속성
- EXTENSION_POINT_VERSION : 이 확장이 구현하는 확장 포인트의 버전

```python
class VerbExtension:
    NAME = None
    EXTENSION_POINT_VERSION = "0.1“
    …
```

```python
class SetVerb(VerbExtension):
    """Set ROS environment variables."""

    def add_arguments(self, parser, cli_name):
        parser.add_argument("env_name", help="Name of the environment variable")
        parser.add_argument("value", help="Value of the environment variable")

    def main(self, *, args):
        if args.env_name or args.value:
            message = set_ros_env(args.env_name, args.value)
            print("[Changed ROS environment variable]:")
            print(message)
        message = get_all_env_list()
        print("\n[Current ROS environment variable]:")
        print(message)
```

```python
class ListVerb(VerbExtension):

    def add_arguments(self, parser, cli_name):
        parser.add_argument(
            "-a",
            "--all",
            action="store_true",
            help="Display all environment variables.",
        )
        parser.add_argument(
            "-r",
            "--ros-env",
            action="store_true",
            help="Display the ROS environment variables.",
        )
        parser.add_argument(
            "-d",
            "--dds-env",
            action="store_true",
            help="Display the DDS environment variables.",
        )

    def main(self, *, args):
        if args.ros_env:
            message = get_ros_env_list()
        elif args.dds_env:
            message = get_dds_env_list()
        else:
            message = get_all_env_list()
        print(message)
```

✓실행 예제 – verb/__init__.py

5. __init__ 함수

- satisfies_version 함수를 사용해 현재 플러그인 시스템 버전이 EXTENSION_POINT_VERSION과 호환되는지 확인

```
    def __init__(self):
        super(VerbExtension, self).__init__()
        satisfies_version(PLUGIN_SYSTEM_VERSION, "^0.1")
```

6. add_arguments 함수

- ROS2 명령어에 필요한 인자들을 정의하기 위한 메서드. 기본 클래스에서는 비어 있으며, 구체적인 확장에서 필요에 따라 이 메서드를 오버라이드하여 구현함

```
def add_arguments(self, parser, cli_name):
    pass
```

✓실행 예제 – verb/__init__.py

7. main 함수

- 각 명령어 확장에서 반드시 구현해야 하는 메서드로, 명령어의 주요 로직을 수행하는 함수
- 기본 클래스에서는 NotImplementedError를 발생시켜

  이 메서드가 반드시 하위 클래스에서 오버라이드 되어야 함을 알림

```python
def main(self, *, args):
    raise NotImplementedError()
```

✓복습 Override(Object Oriented Programming)

Override란?

- 상속받은 부모 클래스(슈퍼클래스)의 메서드를, 자식 클래스(서브 클래스)에서 다시 정의함
- 자식 클래스에서 상속 받은 메서드를 내 스타일로 다시 작성하는 것

Override가 필요한 이유?

- 부모 클래스는 일반적 / 공통적인 기능을 정의
- 자식 클래스는 특수화 / 구체적인 기능을 하고 싶을때
- 코드 재상용성을 높이고, 다형성(Polymorphism)도 구현

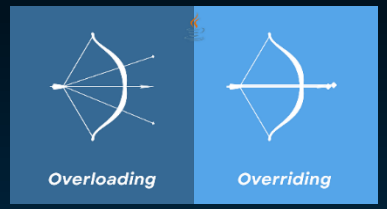
```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):  # 오버라이드!
        print("Dog barks")

a = Animal()
a.speak()  # Animal speaks

d = Dog()
d.speak()  # Dog barks   ← 오버라이드된 메서드 호출
```

다형성? 하나의 인터페이스(메서드 이름)가 여러 형태로 동작하는 것

```
class Animal:
    def speak(self):
        print("동물이 소리를 낸다.")

class Dog(Animal):
    def speak(self):
        print("멍멍!")

class Cat(Animal):
    def speak(self):
        print("야옹~")

def make_it_speak(animal):
    animal.speak()
```

```
dog = Dog()
cat = Cat()

make_it_speak(dog)  # 멍멍!
make_it_speak(cat)  # 야옹~
```

✓복습 Override vs Overload



Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

```
int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

String add(String a, String b) {
    return a + b;
}
```

※ C++, Java에서는 Overload지원하나 Python에서는 엄밀하게 보면 지원하지 않으나 source code내에서 분기하는 형태로 구현 가능

✓복습 interface와 추상클래스

Interface란?

- 이런 기능들을 갖춘 객체여야 한다고 선언
- 어떤 기능을 구현해야 하는지 약속만 정해 놓은 것

Interface가 필요한 이유?

- 여러 개발자가 같은 시스템 안에서 다양한 기능을 추가
- 개발자 각자의 방식으로 메서드 이름, 동작 방식 정한다면?
- 코드 재사용, 일정 규모 이상 프로젝트 협업 필수!!!

```java
interface Animal {
    void speak();
}

class Dog implements Animal {  ← 구현체
    public void speak() {
        System.out.println("멍멍!");
    }
}

class Cat implements Animal {  ← 구현체
    public void speak() {
        System.out.println("야옹~");
    }
}
```


```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print("멍멍!")

class Cat(Animal):
    def speak(self):
        print("야옹~")
```

※ Python에서는 interface keyword가 없으며 대신 추상 클래스를 사용

※ Interface는 규칙(틀)만 제공하나 추상 클래스는 틀 + 기본 기능 제공

✓실행 예제 – setup.py

1. setup.py의 entry_points를 아래와 같이 수정

```
entry_points={
    "ros2cli.command": [
        "env = ros2env.command.env:EnvCommand",
    ],
    "ros2cli.extension_point": [
        "ros2env.verb = ros2env.verb:VerbExtension",
    ],
    "ros2env.verb": [
        "list = ros2env.verb.list:ListVerb",
        "set = ros2env.verb.set:SetVerb",
    ],
},
```

← env라는 이름으로 ros2env/command/env.py 파일안의 EnvCommand 클래스를 등록(ros2 env 명령이 동작)

← extention_point 등록. ros2env/verb.py파일안의 VerbExtention클래스를 연결. Verb가 어떤 인터페이스를 사용하는지 알려줌

← 서브 명령어(verb)를 등록하는 부분. ros2env/verb/list.py와 ros2env/verb/list.py 2개 클래스. ros2 env list와 ros2 env set 입력가능

```
from ros2cli.command import add_subparsers_on_demand
from ros2cli.command import CommandExtension


class EnvCommand(CommandExtension):
    def add_arguments(self, parser, cli_name):
        self._subparser = parser

        add_subparsers_on_demand(
            parser, cli_name, "_verb", "ros2env.verb", required=False
        )

    def main(self, *, parser, args):
        if not hasattr(args, "_verb"):
            self._subparser.print_help()
            return 0

        extension = getattr(args, "_verb")

        return extension.main(args=args)
```

✓실행 예제 – setup.py

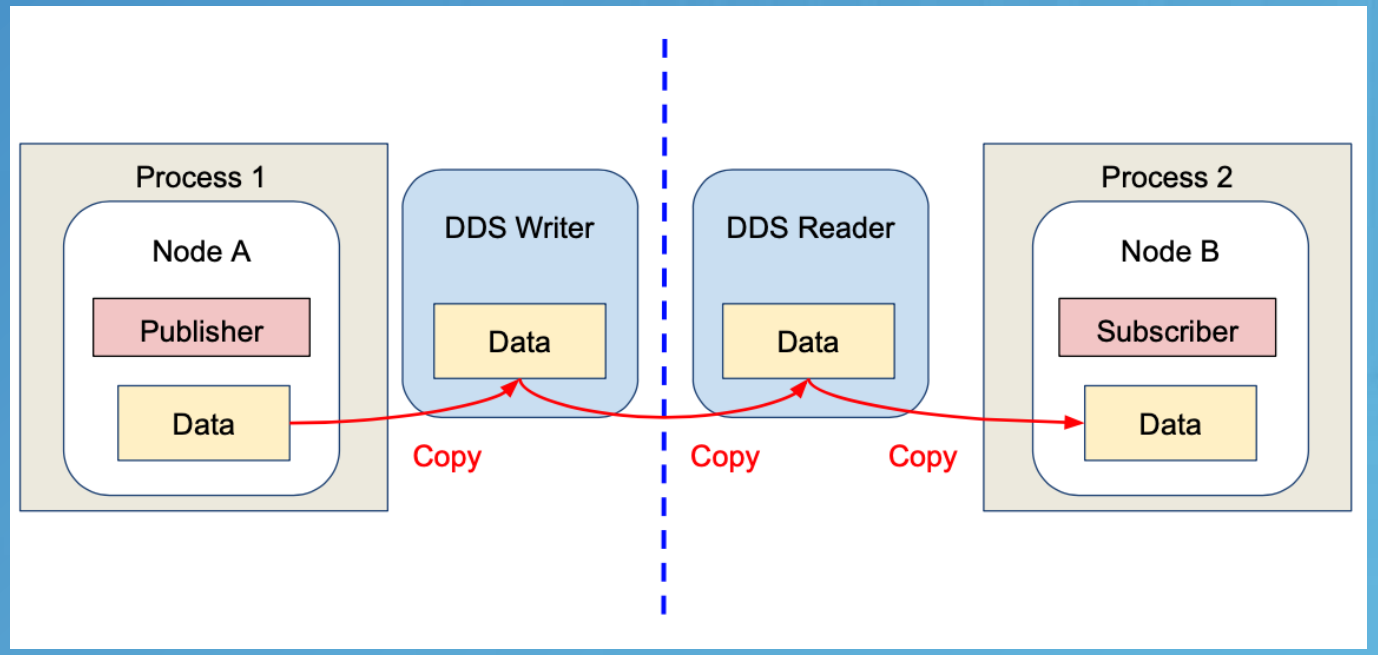
2. 터미널을 열어 패키지를 빌드하고, 새로 빌드 된 패키지를 사용

```
$ colcon build
$ source install/setup.bash
```

3. 아래 명령어들을 실행하여 제대로 빌드가 되었음을 확인

```
$ ros2 -h            # Show this help message and exit
$ ros2 env list      # Display all environment variables.
$ ros2 env list -r   # Display the ROS environment variables.
$ ros2 env list -d   # Display the DDS environment variables.
```

Intra-Process Communication



Intra-process communication

Intra-process communication

- ROS는 복수개의 node를 사용하여 개발이 이루어짐
- 단일 컴퓨팅 시스템에서 복수개의 node 사용시
- 데이터 통신을 위한 작업으로 인한 전체적인 성능 저하 및 메모리 사용량 증가하는 단점
- ROS2에서는 이를 해결하기 위해 IPC(Intra-Process Communication)제공
- 예, 모바일 로봇 : 라이다 데이터 노드, 모터 제어 노드, 로봇 위치 추종 노드, 경로 생성 노드 등.....

Intra-process communication

- ROS에서 서로 다른 프로세스의 아이디를 확인해 보면 명확히 다른 것을 확인 가능

```
user@user:~$ ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
[INFO] [1728878217.077062097] [minimal_subscriber]: I heard: 'Hello, world! 0'
[INFO] [1728878217.576641396] [minimal_subscriber]: I heard: 'Hello, world! 1'
```

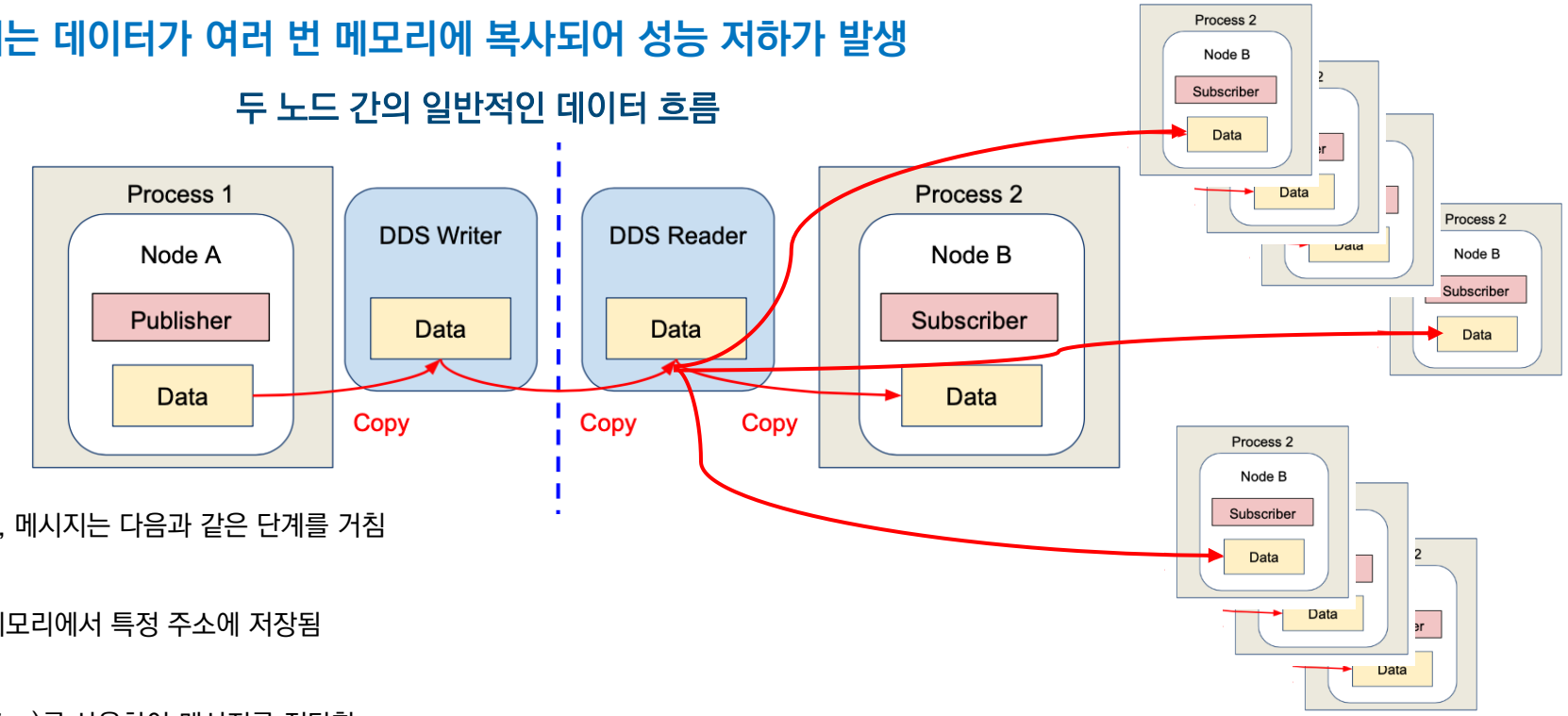
```
user@user:~$ ros2 run examples_rclcpp_minimal_publisher publisher_member_function
[INFO] [1728878217.075749315] [minimal_publisher]: Publishing: 'Hello, world! 0'
[INFO] [1728878217.575734858] [minimal_publisher]: Publishing: 'Hello, world! 1'
```

```
user@user:~$ ps -e
    PID TTY          TIME CMD
2849329 pts/0    00:00:01 subscriber_memb
2849511 pts/1    00:00:01 publisher_membe
```

Intra-process communication

- **서로 다른 프로세스는 송수신되는 데이터가 여러 번 메모리에 복사되어 성능 저하가 발생**

두 노드 간의 일반적인 데이터 흐름



[기본적인 데이터 복사]

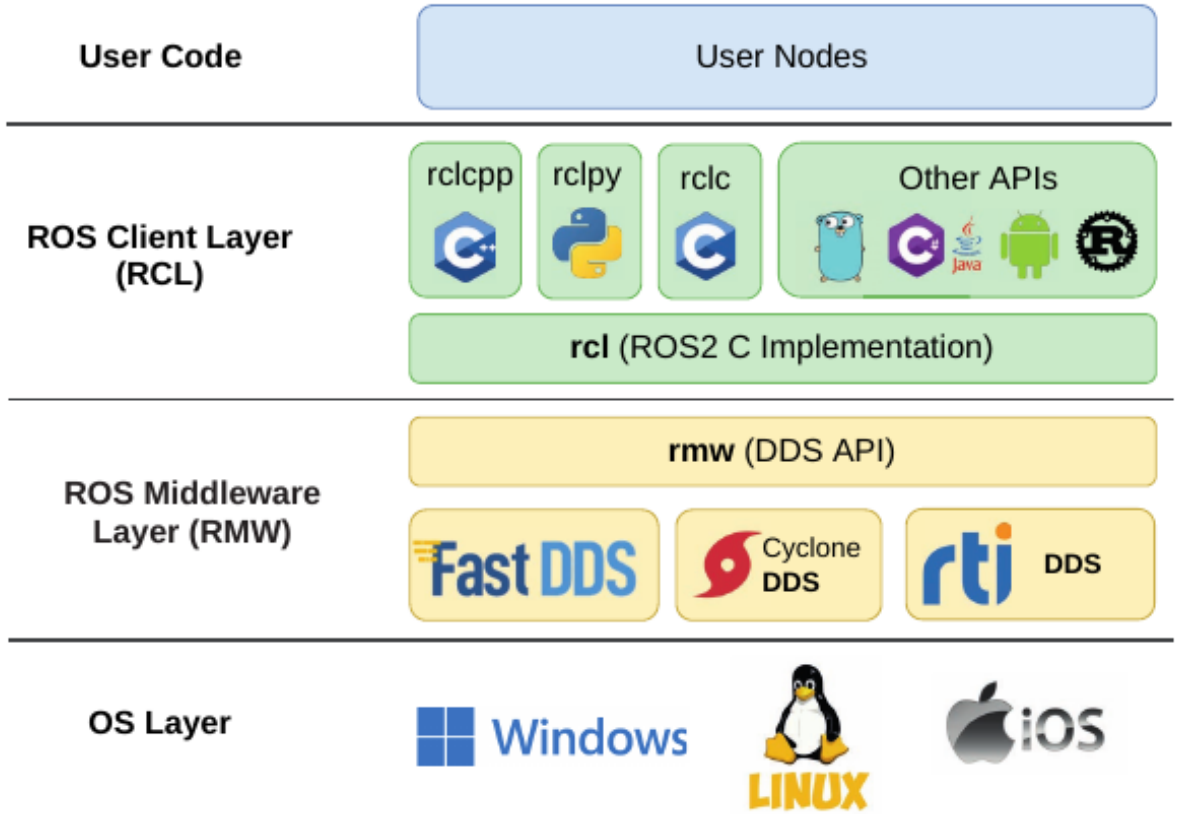
일반적으로 ROS 2에서 노드 간에 데이터를 전달할 때, 메시지는 다음과 같은 단계를 거침

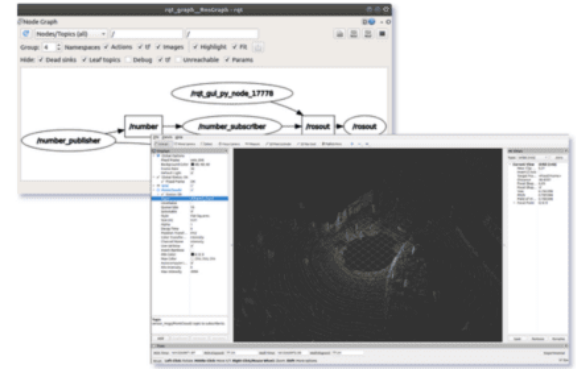
1. **퍼블리셔가 메시지를 생성**
   - 사용자가 메시지를 생성하면, 해당 데이터가 메모리에서 특정 주소에 저장됨
2. **DDS 미들웨어를 통한 데이터 전달**
   1. ROS2는 DDS(Data Distribution Service)를 사용하여 메시지를 전달함
   2. 일반적인 DDS 구현에서는 데이터를 **네트워크 버퍼 또는 공유 메모리**로 복사하여 송신함
3. **구독자가 데이터를 수신**
   1. 수신된 데이터는 다시 사용자 프로그램의 메모리로 복사됨
   2. 여러 개의 구독자가 존재할 경우, 각 구독자에게 별도로 복사됨

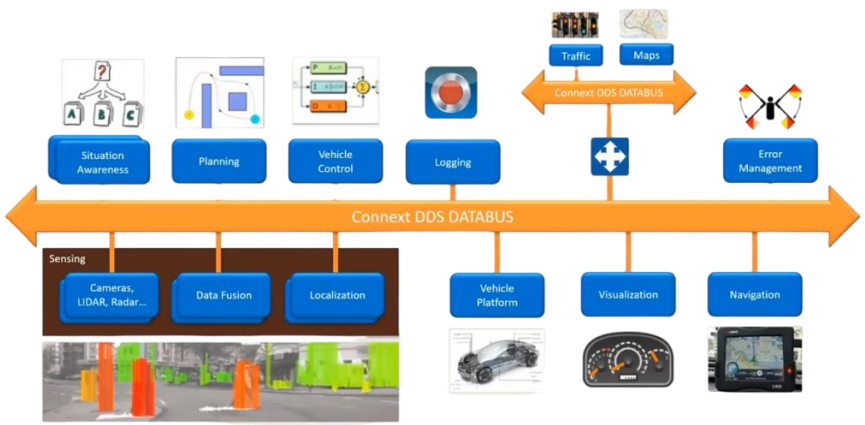
이 과정에서 **여러 번의 메모리 복사가 발생**하며,

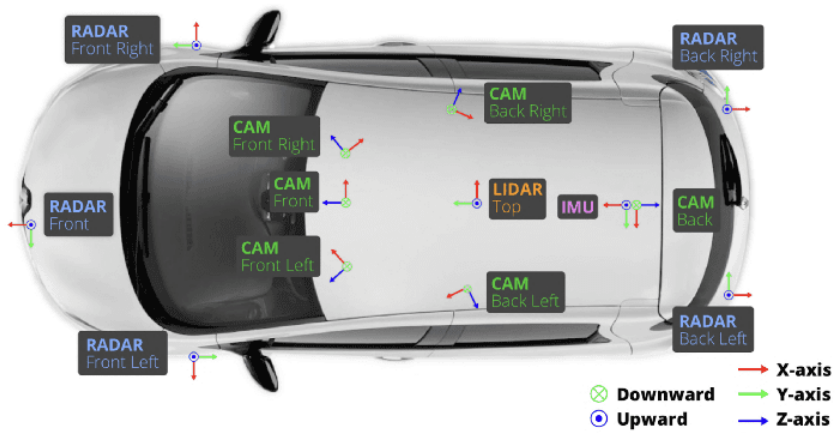
특히 **대용량 데이터(이미지, LiDAR 포인트 클라우드 등)** 전송 시

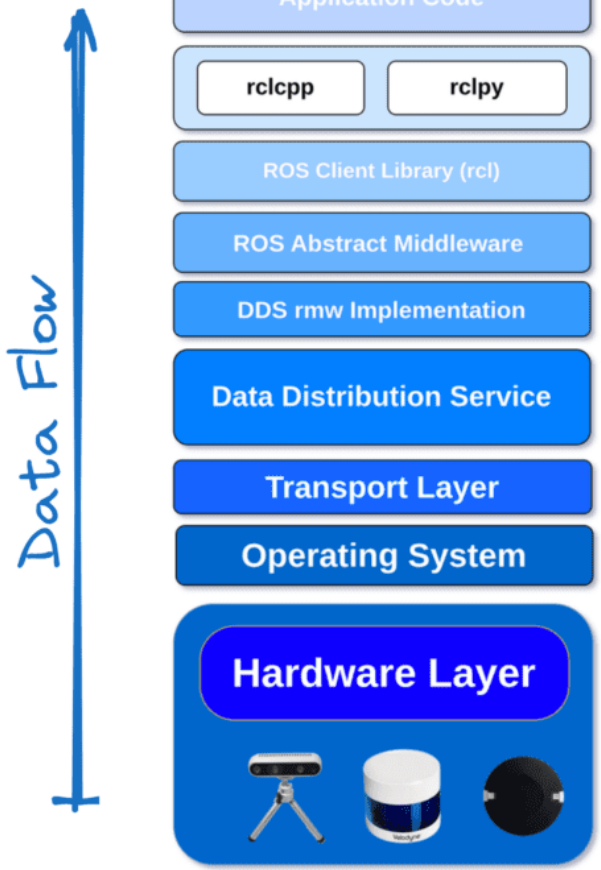
메모리 사용량 증가와 성능 저하 발생


User Code
User Nodes
ROS Client Layer (RCL)
rclcpp
rclpy
rclc
Other APIs
rcl (ROS2 C Implementation)
ROS Middleware Layer (RMW)
rmw (DDS API)
Fast DDS
Cyclone DDS
rti DDS
OS Layer
Windows
LINUX
iOS



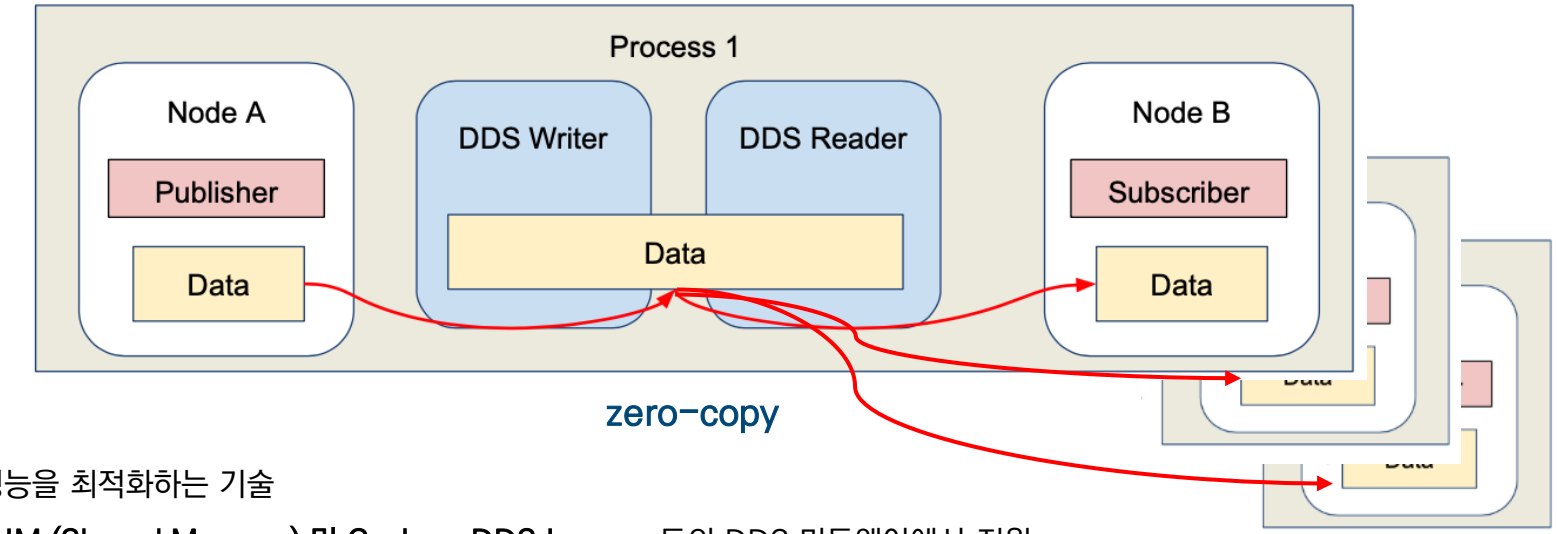

Traffic
Maps
Connext DDS DATABUS
Situation Awareness
Planning
Vehicle Control
Logging
Error Management
Connext DDS DATABUS
Sensing
Cameras, LiDAR, Radar...
Data Fusion
Localization
Vehicle Platform
Visualization
Navigation


RADAR Front Right
RADAR Back Right
CAM Back Right
CAM Front Right
CAM Front
RADAR Front
LIDAR Top
IMU
CAM Back
CAM Front Left
CAM Back Left
RADAR Front Left
RADAR Back Left
Downward
Upward
X-axis
Y-axis
Z-axis


Data Flow
Application Code
rclcpp
rclpy
ROS Client Library (rcl)
ROS Abstract Middleware
DDS rmw Implementation
Data Distribution Service
Transport Layer
Operating System
Hardware Layer

Intra-process communication

- IPC를 이용하면 복수개의 노드를 단일 프로세스에서 처리하여 해당 문제를 해결



[Zero-Copy란?]

데이터 복사를 최소화하여 성능을 최적화하는 기술

ROS 2에서는 **Fast DDS SHM (Shared Memory) 및 Cyclone DDS Iceoryx** 등의 DDS 미들웨어에서 지원

[Zero-Copy 방식의 데이터 흐름]

1. 퍼블리셔가 메시지를 생성하면, 데이터가 공유 메모리(SHM, Shared Memory)에 저장됨
2. DDS는 데이터를 네트워크로 전송하지 않고, 같은 프로세스 또는 동일한 머신 내의 구독자들에게 참조 방식으로 전달
3. 구독자는 데이터를 복사 없이 직접 참조하여 사용함
4. 즉, 데이터가 한 번만 생성되고 여러 구독자가 이를 직접 읽을 수 있어 메모리 복사가 필요 없음

Intra-process communication

GitHub https://github.com/ros2/demos/tree/humble/intra_process_demo/src

```
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$ ros2 run intra_process_demo two_node_pipeline
Published message with value: 0, and address: 0x59B41DB36C30
 Received message with value: 0, and address: 0x59B41DB36C30
Published message with value: 1, and address: 0x59B41DB36C30
 Received message with value: 1, and address: 0x59B41DB36C30
Published message with value: 2, and address: 0x59B41DB36C30
 Received message with value: 2, and address: 0x59B41DB36C30
Published message with value: 3, and address: 0x59B41DB36C30
 Received message with value: 3, and address: 0x59B41DB36C30
Published message with value: 4, and address: 0x59B41DB36C30
 Received message with value: 4, and address: 0x59B41DB36C30
Published message with value: 5, and address: 0x59B41DB36C30
 Received message with value: 5, and address: 0x59B41DB36C30
Published message with value: 6, and address: 0x59B41DB36C30
 Received message with value: 6, and address: 0x59B41DB36C30
Published message with value: 7, and address: 0x59B41DB36C30
 Received message with value: 7, and address: 0x59B41DB36C30
Published message with value: 8, and address: 0x59B41DB36C30
 Received message with value: 8, and address: 0x59B41DB36C30
Published message with value: 9, and address: 0x59B41DB36C30
```

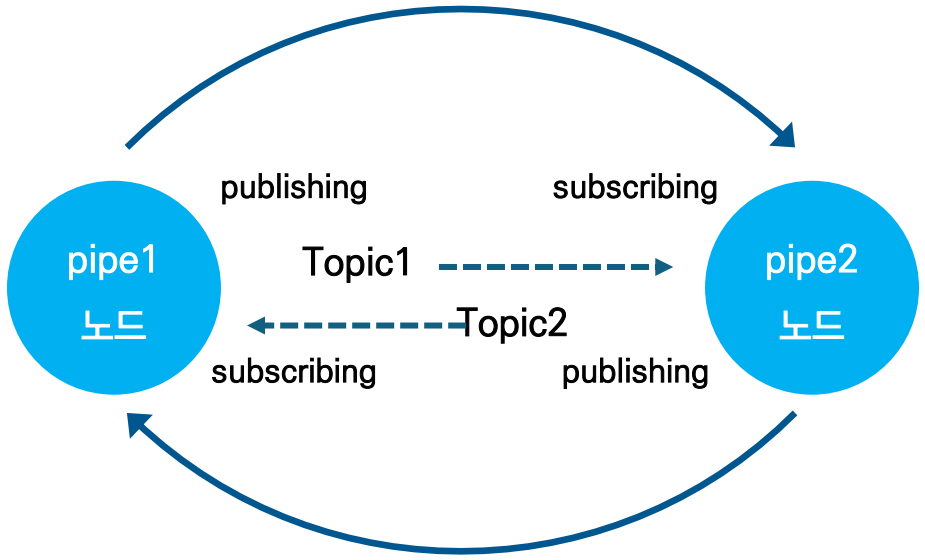
← 메시지 address 1개

- 노드 2개(Producer, Consumer)가 1개 프로세스에 있음
- publisher → subscriber
- 메모리 복사 없이 빠르게 메시지 전달(메모리 주소)

```
victor@victor-VirtualBox: ~ 82x8
   2457 ?        00:00:00 kworker/6:1H-kblockd
   2458 ?        00:00:00 kworker/0:1-events
   2460 ?        00:00:00 kworker/5:0-events
   2463 pts/1    00:00:00 bash
   2530 pts/0    00:00:00 ros2
   2531 pts/0    00:00:00 two_node_pipeli
   2541 pts/1    00:00:00 ps
victor@victor-VirtualBox:~$
```

← 프로세스 1개

Intra-process communication




```
victor@victor-VirtualBox:~$ ros2 run intra_process_demo cyclic_pipeline
Published first message with value:  42, and address:  0x5CDB8ED09B90
Received message with value:         42, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 43, and address:  0x5CDB8ED09B90
Received message with value:         43, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 44, and address:  0x5CDB8ED09B90
Received message with value:         44, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 45, and address:  0x5CDB8ED09B90
Received message with value:         45, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 46, and address:  0x5CDB8ED09B90
Received message with value:         46, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 47, and address:  0x5CDB8ED09B90
Received message with value:         47, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 48, and address:  0x5CDB8ED09B90
Received message with value:         48, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 49, and address:  0x5CDB8ED09B90
Received message with value:         49, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
Incrementing and sending with value: 50, and address:  0x5CDB8ED09B90
Received message with value:         50, and address:  0x5CDB8ED09B90
  sleeping for 1 second...
  done.
```

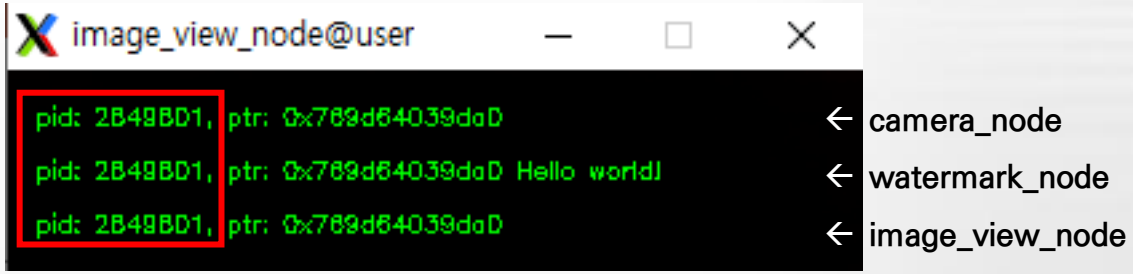
← 메시지 address 1개

Intra-process communication

Image pipeline demo

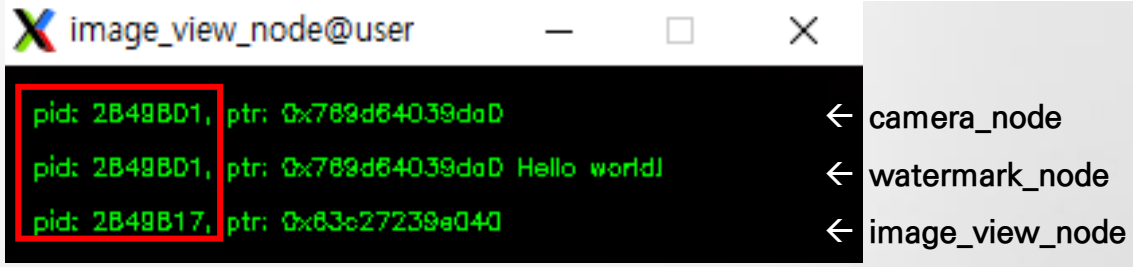
- 다음 명령어를 이용하여 이미지 파이프라인을 실행

```
user@user:~$ ros2 run intra_process_demo image_pipeline_all_in_one
```



← camera_node
← watermark_node
← image_view_node

```
user@user:~$ ros2 run intra_process_demo image_view_node
```



← camera_node
← watermark_node
← image_view_node

이 예제는 총 3개 노드로 구성되어 있음

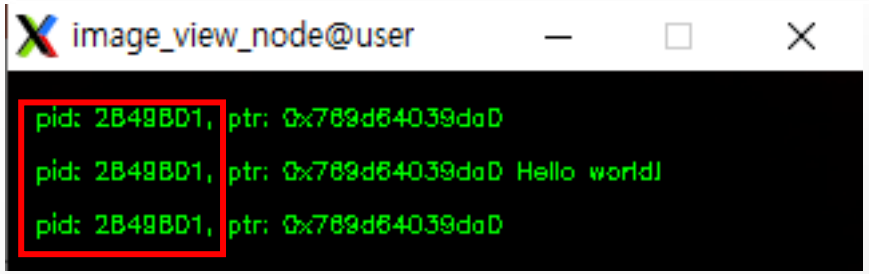
- camera_node : OpenCV 라이브러리를 이용하여 카메라 입력값을 받아 sensor_msg::msg::Image 메시지 타입으로 publishing해주는 역할
- watermark_node : camera_node에서 publishing하는 이미지를 subscribing하고 이미지에 text추가하여 publishing
- Image_view_node : camera_node에서 publishing하는 이미지를 subscribing하여 cv::imshow를 통해 보여줌

Intra-process communication

Image pipeline demo

- 첫번째 터미널은 모두 같은 pid와 동일한 주소값

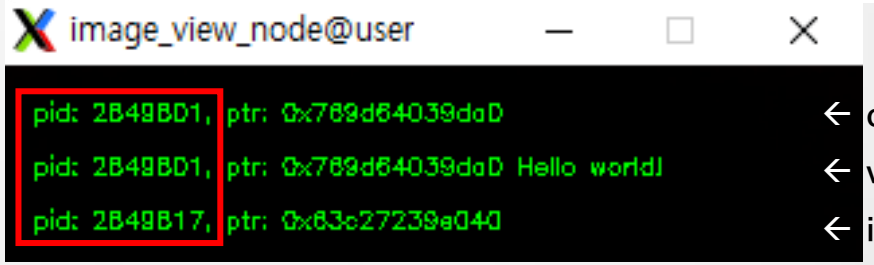
```
user@user:~$ ros2 run intra_process_demo image_pipeline_all_in_one
```



pid는 process id의 약자로 컴퓨터에서 실행 중인 각 프로세스를 구별하기 위해 부여된 고유한 번호

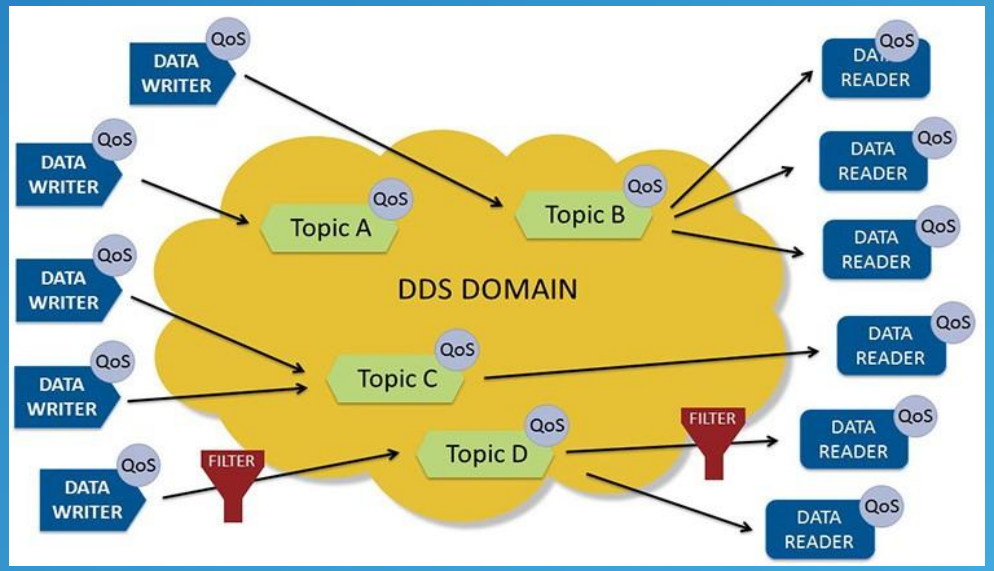
- 두번째 터미널은 camera_node와 watermark_node는 같은 프로세스에서 zero-copy로 이미지 송수신하지만, image_view_node는 다른 프로세스에서 실행되며 참조하는 메모리 주소도 다름

```
user@user:~$ ros2 run intra_process_demo image_view_node
```



← camera_node
← watermark_node
← image_view_node

QoS
(Quality of Service)



DDS의 QoS

DDS의 서비스 품질(QoS, Quality of Service)

QoS

- **QoS(Quality of Service)란 쉽게 말해 '데이터 통신 옵션'**
- **ROS2는 TCP 방식과 UDP 방식을 선택적으로 사용 가능**
  - TCP: 신뢰성(Reliability) 중심
  - UDP: 속도 중심
- **이를 위해 ROS2에서는 DDS의 QoS 도입**
  - 퍼블리셔 또는 서브스크라이버 선언 시 QoS를 매개 변수로 지정하여 원하는 통신 방식 설정 가능
  - QoS로 바꿀 수 있는 것은 데이터 전송 시 실시간성(real time) 설정 관련 부분, 대역폭 옵션, 데이터 지속성, 중복성 등이 있음

DDS의 QoS
QoS의 종류

QoS의 종류

- **현재 DDS에서 설정 가능한 QoS 항목으로는 22가지가 있음**
- **대표적인 QoS 항목**
  - **Reliability** : ROS2에서는 신뢰도를 우선(reliable)으로 설정하거나 통신 속도 최우선(best effort)으로 설정
  - **History** : 정해진 크기만큼 데이터를 보관하는 기능(=depth)
  - **Durability** : 데이터 수신하는 서브스크라이버가 생성되기 전, 데이터의 사용 유무를 설정
  - **Deadline** : 정해진 주기 내 데이터의 발신 및 수신이 없는 경우 이벤트 함수 실행
  - **Lifespan** : 정해진 주기 내 수신되는 데이터에만 유효 판정, 이외 데이터는 삭제
  - **Liveliness** : 정해진 주기 내 노드 또는 토픽의 생사를 확인

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

History

- Values

History	데이터를 몇 개나 보관할 지 결정하는 QoS 옵션
KEEP_LAST	정해진 메시지 큐 사이즈(depth) 만큼 데이터 보관 - depth: 메시지 큐 사이즈 (KEEP_LAST 설정일 경우에만 유효)
KEEL_ALL	모든 데이터 보관 (최대 사이즈는 DDS 벤더마다 다름)

- 예시

```
qos_profile = QoSProfile(history=QoSHistoryPolicy.KEEP_LAST, depth=10)
```

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

Reliability

- Values

Reliability	신뢰성 또는 속도 우선 설정
BEST_EFFORT	데이터 송신에 집중. 전송 속도를 중시하며 네트워크에 따라 유실 발생 가능성
RELIABLE	데이터 수신에 집중. 신뢰성 중시하며 유실 발생 시 재전송을 통해 수신 보장

- 예시

```
qos_profile = QoSProfile(reliability=QoSReliabilityPolicy.BEST_EFFORT)
```

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

Durability

- Values

Durability	데이터 수신하는 서브스크라이버가 생성되기 전, 데이터의 사용 유무를 설정
TRANSIENT_LOCAL	Subscription이 생성되기 전 데이터도 보관(Publisher에만 적용 가능)
VOLATILE	Subscription이 생성되기 전 데이터는 무효

- 예시

```
qos_profile = QoSProfile(durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)
```

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

Deadline

- Values

Deadline	정해진 주기 내 데이터의 발신 및 수신이 없는 경우 이벤트 함수 실행
deadline_duration	Deadline을 확인하는 주기

- 예시

```
qos_profile = QoSProfile(depth=10, deadline=Duration(0.1))
```

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

Lifespan

- Values

Lifespan	정해진 주기 내 수신되는 데이터에만 유효 판정, 이외 데이터는 삭제
lifespan_duration	Lifespan을 확인하는 주기

- 예시

```
qos_profile = QoSProfile(lifespan=Duration(0.01))
```

DDS의 QoS

ROS2에서 사용하는 QoS 옵션

Liveliness

- Values

Liveliness	정해진 주기 내 노드 또는 토픽의 생사를 확인
liveliness	자동 또는 매뉴얼로 확인할 지 지정하는 옵션 (3가지 중 선택) (AUTOMATIC, MANUAL_BY_NODE, MANUAL_BY_TOPIC)
lease_duration	Liveliness를 확인하는 주기

- 예시

```
qos_profile = QoSProfile(
    liveliness=AUTOMATIC,
    liveliness_lease_duration=Duration(1.0))
```

DDS의 QoS

rmw_qos_profile 사용과 유저 QoS 프로파일 사용

rmw_qos_profile

- RMW QoS Profile: ROS2의 RMW에서 가장 많이 사용하는 QoS 설정을 하나의 세트로 표현한 것
- 목적에 따라 Default, Sensor Data, Service, Action Status, Parameters, Parameters Events의 6가지로 구분

	Default	Sensor Data	Service	Action Status	Parameters	Parameter Events
Reliability	RELIABLE	BEST_EFFORT	RELIABLE	RELIABLE	RELIABLE	RELIABLE
History	KEEP_LAST	KEEP_LAST	KEEP_LAST	KEEP_LAST	KEEP_LAST	KEEP_LAST
Depth (History)	10	5	10	1	1000	1000
Durability	VOLATILE	VOLATILE	VOLATILE	TRANSIENT LOCAL	VOLATILE	VOLATILE

DDS의 QoS

DDS Vendor Directory

OMG Standards Development Organization

https://www.omg.org

- 1989년에 설립된 국제 표준화 기구
- 분산 시스템, 모델링 언어, 미들웨어 등 다양한 표준을 개발 및 관리
- 대표적인 표준 중 하나가 DDS(Data Distribution Service)

DDS FOUNDATION

https://www.dds-foundation.org

- OMG가 만든 미들웨어 통신 표준
- 실시간성, 높은 신뢰성, 확장성
- 로봇 뿐만 아니라 항공우주, 국방, 자동차, 산업 자동화 등 다양한 분야 활용

ROS 2

https://docs.ros.org

- ROS1에서의 통신 인프라 관련 한계점과 문제점(네트워크 확장성, 신뢰성 부족)
- OMG가 표준화한 DDS를 기본 통신 프로토콜로 채택
- ROS2의 Publisher, Subscriber 통신, 서비스 호출, 액션 서버 등 저수준 통신이 DDS기반으로 작동
- DDS구현체들 위에 RMW(ROS Middleware Interface)라는 추상화 레이어
- 이로 인해 다양한 DDS Vendor를 선택할 수 있음
- 선택한 이유 : 다양한 QoS, 멀티캐스트, P2P통신, 보안(암호화, 인증), 실시간성, 다양한 상용/오픈소스 구현체 존재

Fast DDS

https://www.eprosima.com

rti

https://www.rti.com

Cyclone DDS

https://cyclonedds.io

DDS의 QoS

DDS & QoS

	DDS (Data Distribution Service)	QoS (Quality of Service)
개념	▪ ROS2 기본 통신 미들웨어로 중앙 서버 없이 분산 네트워크에서 데이터를 교환 ▪ DDS는 ROS2의 기본 통신 프로토콜	▪ DDS통신의 품질을 조정하는 설정으로, 메시지 신뢰성, 내구성, 저장 개수 등을 관리 ▪ DDS를 통해 통신 품질을 조정하는 방식 ▪ QoS설정에 따라 DDS의 동작 방식이 달라지며 각 애플리케이션에 맞는 성능을 최적화 ▪ 네트워크 환경과 응용 프로그램의 요구사항에 따라 적절한 QoS를 설정해야 함
특징	실시간 데이터 교환을 위한 미들웨어 표준, ROS2의 통신기반이 되는 핵심 기술	▪ QoS 주요 설정 6종류 Reliability(신뢰성), Durability(내구성), History(데이터 저장 개수), Lifespan(수명), Deadline(데드라인), Liveliness(활성 상태)
기타	▪ Publisher-subscriber 모델 ▪ Brokerless(중앙 서버 없음) ▪ 자동 발견(Discovery) : 네트워크상의 노드들이 서로를 자동으로 인식하고 연결 ▪ 신뢰성 & 확장성 : 다양한 QoS 설정을 통해 신뢰성과 성능을 조절	▪ BEST_EFFORT + VOLITILE : 카메라 영상 스트리밍(실시간성 우선, 일부 프레임 손실 가능) ▪ RELIABLE + TRANSIENT_LOCAL : 로봇 센서 데이터(정확한 수신 보장, 최신 데이터 유지) ▪ KEEP_LAST(10 + LIFESPAN(5s) : 5초 동안 최신 10개의 데이터를 유지하는 센서 데이터

```
# 예시: Cyclone DDS로 바꾸기
export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
ros2 run demo_nodes_cpp talker
```

rti ADLINK eProsima The Middleware Experts ECLIPSE FOUNDATION Gurum Networks GurumDDS TWINOAKS COMPUTING INC. PRACTICAL MIDDLEWARE EXPERTISE OBJECT COMPUTING YOUR OUTCOMES ENGINEERED

DDS의 QoS

rmw_qos_profile 사용과 유저 QoS 프로파일 사용

rmw_qos_profile

- 예를 들어, 센서와 같이 지속성이 높으며 순간적으로 데이터를 빠르게 전달해야 하는 경우 아래와 같이 설정

```
static const rmw_qos_profile_t rmw_qos_profile_sensor_data =
  {
    RMW_QOS_POLICY_HISTORY_KEEP_LAST,
    5,
    RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,
    RMW_QOS_POLICY_DURABILITY_VOLATILE,
    RMW_QOS_DEADLINE_DEFAULT,
    RMW_QOS_LIFESPAN_DEFAULT,
    RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
    RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
    false
};
```

DDS의 QoS

rmw_qos_profile 사용과 유저 QoS 프로파일 사용

rmw_qos_profile

- 실제 파이썬 코드에서 사용은 다음과 같이 'qos_profile_sensor_data' 모듈을 import하여 사용함

```
from rclpy.qos import qos_profile_sensor_data

...

self.sensor_publisher = self.create_publisher(Int8MultiArray, 'sensor',
qos_profile_sensor_data)
```

DDS의 QoS

rmw_qos_profile 사용과 유저 QoS 프로파일 사용

유저 QoS 프로파일

- 사전 정의한 rmw_qos_profile 이외에도 유저가 직접 설정하여 새로운 프로파일을 만들어 사용 가능
- 아래와 같은 QoS 모듈을 import

```
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy
```

- 코드에서 'QoSProfile'을 선언하여 원하는 옵션을 커스텀하게 설정

```
QOS_RKL10V = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_LAST,
    depth=10,
    durability=QoSDurabilityPolicy.VOLATILE)
```

DDS의 QoS

rmw_qos_profile 사용과 유저 QoS 프로파일 사용

유저 QoS 프로파일

- 다음 'create_publisher'와 같은 함수를 사용할 때 'rmw_qos_profile' 대신 유저가 정의한 커스텀 QoS 프로파일을 매개변수로 사용

```
self.sensor_publisher = self.create_publisher(Int8MultiArray, 'sensor',
QOS_RKL10V)
```

- 유저 QoS 프로파일을 사용하는 것이 커스터마이징에 용이하기 때문에, 실제 개발 시 더 많이 사용됨

QoS programming

- Topic, Service, Action의 QoS 설정

  ⇢ 예제 코드 중심의 QoS 프로그래밍 코드 분석

※ 필요한 파일 : py_pubsub_qos.zip

QoS Programming

Topic, Service, Action의 QoS 설정

Topic

- **Topic의 기본 QoS 설정은 RMW QoS Profile의 기본 설정과 동일**
  - 즉, Reliability는 RELIABLE , History는 KEEP_LAST에 Depth = 10을 따르며 Durability는 VOLATILE이 기본
- **배포한 패키지 ex_calculator의 예시를 보면 다음과 같음**
  - ex_calculator/ex_calculator/arithmetic/argument.py

```
QOS_RKL10V = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_LAST,
    depth=qos_depth,
    durability=QoSDurabilityPolicy.VOLATILE)

self.arithmetic_argument_publisher = self.create_publisher(
    ArithmeticArgument,
    'arithmetic_argument',
    QOS_RKL10V)
```

QoS Programming

Topic, Service, Action의 QoS 설정

Service

- ROS2 Service의 경우, 특별한 케이스 외에는 기본 QoS 사용
- /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/node.py → line 1436~1444

```
def create_service(
    self,
    srv_type,
    srv_name: str,
    callback: Callable[[SrvTypeRequest, SrvTypeResponse], SrvTypeResponse],
    *,
    qos_profile: QoSProfile = qos_profile_services_default,
    callback_group: Optional[CallbackGroup] = None
) → Service:
```

QoS Programming

Topic, Service, Action의 QoS 설정

Service

- qos_profile_services_default는 RMW의 qos_profiles과 types.h 헤더 파일에서 확인 할 수 있음
- /opt/ros/humble/include/rmw/rmw/qos_profiles.h → line 64

```
static const rmw_qos_profile_t rmw_qos_profile_services_default =
{
  RMW_QOS_POLICY_HISTORY_KEEP_LAST,
  10,
  RMW_QOS_POLICY_RELIABILITY_RELIABLE,
  RMW_QOS_POLICY_DURABILITY_VOLATILE,
  RMW_QOS_DEADLINE_DEFAULT,
  RMW_QOS_LIFESPAN_DEFAULT,
  RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
  RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
  false
};
```

QoS Programming

Topic, Service, Action의 QoS 설정

Open | qos_profiles.h [Read-Only] /opt/ros/humble/include/rmw/rmw | Save

```
17
18 #ifdef __cplusplus
19 extern "C"
20 {
21 #endif
22
23 #include "rmw/types.h"
24
25 static const rmw_qos_profile_t rmw_qos_profile_sensor_data =
26 {
27   RMW_QOS_POLICY_HISTORY_KEEP_LAST,
28   5,
29   RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT,
30   RMW_QOS_POLICY_DURABILITY_VOLATILE,
31   RMW_QOS_DEADLINE_DEFAULT,
32   RMW_QOS_LIFESPAN_DEFAULT,
33   RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
34   RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
35   false
36 };
37
38 static const rmw_qos_profile_t rmw_qos_profile_parameters =
39 {
40   RMW_QOS_POLICY_HISTORY_KEEP_LAST,
41   1000,
42   RMW_QOS_POLICY_RELIABILITY_RELIABLE,
43   RMW_QOS_POLICY_DURABILITY_VOLATILE,
44   RMW_QOS_DEADLINE_DEFAULT,
45   RMW_QOS_LIFESPAN_DEFAULT,
46   RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT,
47   RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT,
48   false
49 };
50
51 static const rmw_qos_profile_t rmw_qos_profile_default =
52 {
```

C/ObjC Header | Tab Width: 8 | Ln 36, Col 3 | INS

QoS Programming

Topic, Service, Action의 QoS 설정

Service

- qos_profile_services_default는 RMW의 qos_profiles과 types.h 헤더 파일에서 확인할 수 있음
- /opt/ros/humble/include/rmw/rmw/types.h

```
RMW_QOS_POLICY_LIVELINESS_SYSTEM_DEFAULT = 0
RMW_QOS_DEADLINE_DEFAULT {0, 0}
RMW_QOS_LIFESPAN_DEFAULT {0, 0}
RMW_QOS_LIVELINESS_LEASE_DURATION_DEFAULT {0, 0}
```

QoS Programming

Topic, Service, Action의 QoS 설정

Action

- **액션은 토픽과 서비스를 모두 사용하는 복합 형태**
  - 액션 토픽의 경우 qos_profile_services_default를 기본 설정
  - 피드백 퍼블리셔의 경우 QoSProfile (depth = 10) 혹은 rmw_qos_profile_default를 초기값으로 사용
  - 액션 상태 퍼블리셔의 경우, 전용 프로파일인 qos_profile_action_status_default를 기본값으로 사용

- **파이썬의 경우, goal_service_qos_profile, result_service_qos_profile, cancel_service_qos_profile, feedback_pub_qos_profile, status_pub_qos_profile에 대한 기본 설정을 사용**

- **/opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/action/server.py**

QoS Programming

Topic, Service, Action의 QoS 설정

Action

- /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/action/server.py

```python
class ActionServer(Waitable):
    """ROS Action server."""

    def __init__(
        self,
        node,
        action_type,
        action_name,
        execute_callback,
        *,
        callback_group=None,
        goal_callback=default_goal_callback,
        handle_accepted_callback=default_handle_accepted_callback,
        cancel_callback=default_cancel_callback,
        goal_service_qos_profile=qos_profile_services_default,
        result_service_qos_profile=qos_profile_services_default,
        cancel_service_qos_profile=qos_profile_services_default,
        feedback_pub_qos_profile=QoSProfile(depth=10),
        status_pub_qos_profile=qos_profile_action_status_default,
        result_timeout=900
    ):
```

QoS Programming
QoS 실습

실습

- 6가지 QoS에 대해 실습
  - ROS2에서 기본적으로 제공하는 데모 QoS를 이용하여, QoS 설정 값 변화에 따른 결과를 확인할 수 있음

1. History
2. Reliability
3. Durability
4. Deadline
5. Lifespan
6. Liveliness

QoS Programming
QoS 실습

History 메시지를 몇 개까지 버퍼에 저장할지를 설정하는 옵션

- **데이터 전송 시점 이후 보관할 데이터의 정책을 설정하는 옵션**
  - **KEEP_LAST**: Depth로 설정한 만큼 사이즈의 데이터 보관(최근 몇 개 까지만 저장)
  - **KEEP_ALL**: 모든 데이터 보관(시스템 메모리 한도까지)
- **제공된 'py_pubsub' 코드를 활용하여 History QoS 설정 실습**
  - Publisher와 Subscriber의 QoS 프로파일 값을 변경

✓ py_pubsub/src/publisher_member_function.py 설정

- **Repository 생성**
  - qos_ws/src 폴더를 만든 후 해당 폴더로 이동

```
mkdir -p qos_ws/src
cd qos_ws/src/
```

  - 제공된 코드 파일의 압축 풀기

```
unzip QoS.zip
```

✓ py_pubsub/src/publisher_member_function.py 설정

▪ QoS profile 변경

- Publisher에는 QoS(Quality of Service) 프로파일의 기본값으로 rmw_qos_profile_default로 사용
- 'create_publisher'의 3번째 인자에 10을 넣어주면, depth가 10인 기본 프로파일 'QoSProfile(depth=10)'이 입력되는 구조

```
21  class MinimalPublisher(Node):
22
23      def __init__(self):
24          super().__init__('minimal_publisher')
25          self.publisher_ = self.create_publisher(String, 'topic', 10)
26          timer_period = 0.5  # seconds
27          self.timer = self.create_timer(timer_period, self.timer_callback)
28          self.i = 0
29
30      def timer_callback(self):
31          msg = String()
32          msg.data = 'Hello World: %d' % self.i
33          self.publisher_.publish(msg)
34          self.get_logger().info('Publishing: "%s"' % msg.data)
35          self.i += 1
```

✓ py_pubsub/src/publisher_member_function.py 설정

- **QoS profile 변경**
  - History의 값을 KEEP_LAST로 변경하고 싶다면, 다음과 같이 작성

```
qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_LAST,
    depth=10,
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)
```

  - KEEP_ALL로 변경하고 싶다면

```
qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_ALL,
    depth=10,
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)
```

✓ py_pubsub/src/publisher_member_function.py 설정

- **QoS profile 변경**
  - 작성한 QoS profile을 적용하려면, create_publisher 부분에 작성한 qos_profile을 인자로 전달
  - 이번 예제에서는 History 값을 KEEP_ALL로 설정하여 테스트
  - publisher 노드를 실행한 후 한참이 지나도 모든 데이터가 subscriber에게 전달되는지 확인

✓ py_pubsub/src/publisher_member_function.py

- **퍼블리셔 QoS profile 변경**
  - Reliability는 RELIABLE로 변경
  - History는 다시 KEEP_ALL로 변경
  - Durability를 TRANSIENT_LOCAL로 변경

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy

qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_ALL,
    depth=10,
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', qos_profile=qos_profile)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1

# 생략
```

- RELIABLE: 손실을 방지하고 신뢰도를 우선시 함
- KEEP_ALL : 모든 데이터 보관
- VOLATILE: Subscription이 생성되기 전 데이터도 보관

✓ py_pubsub/src/subscriber_member_function.py 설정

▪ QoS profile 변경

- ▪ 서브스크라이버에도 마찬가지로, QoS 프로파일을 적용

[활용 사례]

- ▪ 카메라가 0.01초마다 사진을 Publishing, Subscriber는 딥러닝 Inference 0.1초 소요

  → 최근 5개 데이터만 버퍼에 저장해두고 그 이전 데이터는 버림
- ▪ 로봇이 경로 명령(move to waypoint)를 보내고 있는 경우 하나라도 메시지를 놓치면 문제 발생 → 이때 KEEP_ALL로 모든 데이터(waypoint) 보관
- ▪ 배터리 잔량, 온도 센서 모니터링 등 → 과거 이력데이터 보다는 지금 실시간 현재 상태만 모니터링 하고 싶은 경우 → 가장 최신 메시지 1개만 받음

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String

from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy

qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_ALL,
    depth=10,
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)


class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            qos_profile=qos_profile)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)

# 이하 생략
```

✓ 빌드 후 py_pubsub 실행

- 빌드

```
colcon build
```

- Talker 실행

```
ros2 run py_pubsub talker
```

- Listener 실행 (Talker의 Publish 메시지가 10개 이상 발행된 뒤 실행)

```
ros2 run py_pubsub listener
```

✓ History 결과 확인

※ QosHistoryPolicy를 KEEP_ALL과 KEEP_LAST로 옵션을 변경해서 build 후 실행해서 비교해보기

▪ Talker

```
[INFO] [1728960028.755602035] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [1728960029.251136348] [minimal_publisher]: Publishing: "Hello World: 1"
[INFO] [1728960029.751138216] [minimal_publisher]: Publishing: "Hello World: 2"
[INFO] [1728960030.250953449] [minimal_publisher]: Publishing: "Hello World: 3"
[INFO] [1728960030.751144174] [minimal_publisher]: Publishing: "Hello World: 4"
[INFO] [1728960031.251150013] [minimal_publisher]: Publishing: "Hello World: 5"
[INFO] [1728960031.751143261] [minimal_publisher]: Publishing: "Hello World: 6"
[INFO] [1728960032.251148763] [minimal_publisher]: Publishing: "Hello World: 7"
[INFO] [1728960032.750984085] [minimal_publisher]: Publishing: "Hello World: 8"
[INFO] [1728960033.251142889] [minimal_publisher]: Publishing: "Hello World: 9"
[INFO] [1728960033.751142925] [minimal_publisher]: Publishing: "Hello World: 10"
[INFO] [1728960034.251157856] [minimal_publisher]: Publishing: "Hello World: 11"
[INFO] [1728960034.751124254] [minimal_publisher]: Publishing: "Hello World: 12"
[INFO] [1728960035.251122154] [minimal_publisher]: Publishing: "Hello World: 13"
[INFO] [1728960035.750943916] [minimal_publisher]: Publishing: "Hello World: 14"
[INFO] [1728960036.251145644] [minimal_publisher]: Publishing: "Hello World: 15"
[INFO] [1728960036.751100000] [minimal_publisher]: Publishing: "Hello World: 16"
[INFO] [1728960037.251112268] [minimal_publisher]: Publishing: "Hello World: 17"
[INFO] [1728960037.751119333] [minimal_publisher]: Publishing: "Hello World: 18"
[INFO] [1728960038.250957542] [minimal_publisher]: Publishing: "Hello World: 19"
[INFO] [1728960038.751153949] [minimal_publisher]: Publishing: "Hello World: 20"
[INFO] [1728960039.251152932] [minimal_publisher]: Publishing: "Hello World: 21"
[INFO] [1728960039.751040849] [minimal_publisher]: Publishing: "Hello World: 22"
[INFO] [1728960040.251106418] [minimal_publisher]: Publishing: "Hello World: 23"
[INFO] [1728960040.751208876] [minimal_publisher]: Publishing: "Hello World: 24"
[INFO] [1728960041.251003452] [minimal_publisher]: Publishing: "Hello World: 25"
[INFO] [1728960041.751118928] [minimal_publisher]: Publishing: "Hello World: 26"
[INFO] [1728960042.251085540] [minimal_publisher]: Publishing: "Hello World: 27"
[INFO] [1728960042.751083962] [minimal_publisher]: Publishing: "Hello World: 28"
[INFO] [1728960043.251170868] [minimal_publisher]: Publishing: "Hello World: 29"
[INFO] [1728960043.751039776] [minimal_publisher]: Publishing: "Hello World: 30"
[INFO] [1728960044.251048364] [minimal_publisher]: Publishing: "Hello World: 31"
[INFO] [1728960044.751046030] [minimal_publisher]: Publishing: "Hello World: 32"
[INFO] [1728960045.251054532] [minimal_publisher]: Publishing: "Hello World: 33"
[INFO] [1728960045.751076418] [minimal_publisher]: Publishing: "Hello World: 34"
[INFO] [1728960046.251119306] [minimal_publisher]: Publishing: "Hello World: 35"
[INFO] [1728960046.751025297] [minimal_publisher]: Publishing: "Hello World: 36"
[INFO] [1728960047.251077625] [minimal_publisher]: Publishing: "Hello World: 37"
[INFO] [1728960047.751082959] [minimal_publisher]: Publishing: "Hello World: 38"
[INFO] [1728960048.251148269] [minimal_publisher]: Publishing: "Hello World: 39"
[INFO] [1728960048.751182109] [minimal_publisher]: Publishing: "Hello World: 40"
```

▪ Listener

```
[INFO] [1728960046.792327110] [minimal_subscriber]: I heard: "Hello World: 0"
[INFO] [1728960046.792646271] [minimal_subscriber]: I heard: "Hello World: 1"
[INFO] [1728960046.792869351] [minimal_subscriber]: I heard: "Hello World: 2"
[INFO] [1728960046.793068243] [minimal_subscriber]: I heard: "Hello World: 3"
[INFO] [1728960046.793273114] [minimal_subscriber]: I heard: "Hello World: 4"
[INFO] [1728960046.793474782] [minimal_subscriber]: I heard: "Hello World: 5"
[INFO] [1728960046.793674403] [minimal_subscriber]: I heard: "Hello World: 6"
[INFO] [1728960046.793862265] [minimal_subscriber]: I heard: "Hello World: 7"
[INFO] [1728960046.794054844] [minimal_subscriber]: I heard: "Hello World: 8"
[INFO] [1728960046.794252309] [minimal_subscriber]: I heard: "Hello World: 9"
[INFO] [1728960046.794447119] [minimal_subscriber]: I heard: "Hello World: 10"
[INFO] [1728960046.794641667] [minimal_subscriber]: I heard: "Hello World: 11"
[INFO] [1728960046.794836070] [minimal_subscriber]: I heard: "Hello World: 12"
[INFO] [1728960046.795030504] [minimal_subscriber]: I heard: "Hello World: 13"
[INFO] [1728960046.795222185] [minimal_subscriber]: I heard: "Hello World: 14"
[INFO] [1728960046.795414168] [minimal_subscriber]: I heard: "Hello World: 15"
[INFO] [1728960046.795605814] [minimal_subscriber]: I heard: "Hello World: 16"
[INFO] [1728960046.795799690] [minimal_subscriber]: I heard: "Hello World: 17"
[INFO] [1728960046.795992286] [minimal_subscriber]: I heard: "Hello World: 18"
[INFO] [1728960046.796182085] [minimal_subscriber]: I heard: "Hello World: 19"
[INFO] [1728960046.796378211] [minimal_subscriber]: I heard: "Hello World: 20"
[INFO] [1728960046.796572096] [minimal_subscriber]: I heard: "Hello World: 21"
[INFO] [1728960046.796768184] [minimal_subscriber]: I heard: "Hello World: 22"
[INFO] [1728960046.796958612] [minimal_subscriber]: I heard: "Hello World: 23"
[INFO] [1728960046.797158629] [minimal_subscriber]: I heard: "Hello World: 24"
[INFO] [1728960046.797354246] [minimal_subscriber]: I heard: "Hello World: 25"
[INFO] [1728960046.797548259] [minimal_subscriber]: I heard: "Hello World: 26"
[INFO] [1728960046.797742884] [minimal_subscriber]: I heard: "Hello World: 27"
[INFO] [1728960046.797937509] [minimal_subscriber]: I heard: "Hello World: 28"
[INFO] [1728960046.798132134] [minimal_subscriber]: I heard: "Hello World: 29"
[INFO] [1728960046.798326759] [minimal_subscriber]: I heard: "Hello World: 30"
[INFO] [1728960046.798521384] [minimal_subscriber]: I heard: "Hello World: 31"
[INFO] [1728960046.798716009] [minimal_subscriber]: I heard: "Hello World: 32"
[INFO] [1728960046.798910634] [minimal_subscriber]: I heard: "Hello World: 33"
[INFO] [1728960046.799105259] [minimal_subscriber]: I heard: "Hello World: 34"
[INFO] [1728960046.799299884] [minimal_subscriber]: I heard: "Hello World: 35"
[INFO] [1728960046.799494509] [minimal_subscriber]: I heard: "Hello World: 36"
[INFO] [1728960047.251450259] [minimal_subscriber]: I heard: "Hello World: 37"
[INFO] [1728960047.751401218] [minimal_subscriber]: I heard: "Hello World: 38"
[INFO] [1728960048.251428612] [minimal_subscriber]: I heard: "Hello World: 39"
[INFO] [1728960048.751573584] [minimal_subscriber]: I heard: "Hello World: 40"
```

QoSHistoryPolicy 설정	Depth 적용여부	메시지 저장방식
KEEP_ALL	X (무시됨)	모든 메시지를 저장 (RMW에 따라 차이는 있지만 시스템 메모리에 의해 제한)
KEEP_LAST	O (필수 설정)	최근 depth 개수만 유지

QoS Programming
QoS 실습

Reliability 메시지를 얼마나 신뢰성 있게 전달할지를 설정하는 옵션

- TCP처럼 손실을 방지하면서 신뢰도를 우선시(전달 보장) : RELIABLE
- UDP처럼 손실을 감안하고 통신 속도를 우선시(데이터 유실 허용, 실시간성) : BEST_EFFORT

- 이번 예제에서는 인위적으로 네트워크 손실을 발생한 뒤, BEST_EFFORT를 진행
- ROS2에서 기본적으로 제공하는 'demo_node_cpp' 패키지를 활용

[활용 사례]

- BEST_EFFORT(약간의 손실을 감수하더라도 지연 없는 빠른 처리가 더 중요한 경우)
  1. 카메라 영상 스트리밍(frame 몇 개 누락되어도 괜찮음)
  2. LiDAR 센서 데이터
  3. 주행 중 실시간 거리 센서
  4. 드론 영상 중계

[활용 사례]

- RELIABLE(모든 메시지는 반드시 전달되어야 하는 경우)
  1. 로봇 제어 명령(STOP, TURN, MOVE)
  2. 지도 데이터 전송(SLAM map)
  3. 긴급 정지 신호(Emergency Stop)
  4. 산업용 로봇 공정 제어 신호

✓ Reliable 테스트

- tc 명령어를 통한 데이터 손실 명령
  - 이번 예제에서는 45%의 손실로 설정 → ※ 테스트 후에 반드시 원복해주어야 함

```
sudo tc qdisc add dev lo root netem loss 45%
```

- Best effort 옵션의 Listener 실행 (Listener 먼저 실행)

```
ros2 run demo_nodes_cpp listener_best_effort
```

- Talker 실행

```
ros2 run demo_nodes_cpp talker
```

✓ Reliable 결과 확인

- Talker

```
user@user:~$ ros2 run demo_nodes_cpp talker
[INFO] [1728960488.932079343] [talker]: Publishing: 'Hello World: 1'
[INFO] [1728960489.932066752] [talker]: Publishing: 'Hello World: 2'
[INFO] [1728960490.932098547] [talker]: Publishing: 'Hello World: 3'
[INFO] [1728960491.932108026] [talker]: Publishing: 'Hello World: 4'
[INFO] [1728960492.932092876] [talker]: Publishing: 'Hello World: 5'
[INFO] [1728960493.931961456] [talker]: Publishing: 'Hello World: 6'
[INFO] [1728960494.932100749] [talker]: Publishing: 'Hello World: 7'
[INFO] [1728960495.932079122] [talker]: Publishing: 'Hello World: 8'
[INFO] [1728960496.932100239] [talker]: Publishing: 'Hello World: 9'
[INFO] [1728960497.932058943] [talker]: Publishing: 'Hello World: 10'
[INFO] [1728960498.932064152] [talker]: Publishing: 'Hello World: 11'
[INFO] [1728960499.931990475] [talker]: Publishing: 'Hello World: 12'
[INFO] [1728960500.931972003] [talker]: Publishing: 'Hello World: 13'
[INFO] [1728960501.931903943] [talker]: Publishing: 'Hello World: 14'
```

- Listener

```
user@user:~$ ros2 run demo_nodes_cpp listener_best_effort
[INFO] [1728960493.932286785] [listener]: I heard: [Hello World: 6]
[INFO] [1728960494.932551303] [listener]: I heard: [Hello World: 7]
[INFO] [1728960495.932398329] [listener]: I heard: [Hello World: 8]
[INFO] [1728960496.932418975] [listener]: I heard: [Hello World: 9]
[INFO] [1728960497.932431071] [listener]: I heard: [Hello World: 10]
[INFO] [1728960498.932358265] [listener]: I heard: [Hello World: 11]
[INFO] [1728960499.932265881] [listener]: I heard: [Hello World: 12]
```

✓ Reliable 결과 확인

- **결과 해석**
  - Listener가 먼저 실행되어, Talker가 발행하기를 대기하였음
  - 그러나, 데이터 손실에 의해 1 ~ 5번째의 데이터가 손실
  - Listener는 손실된 데이터를 받지 못한 결과를 확인할 수 있음

- **데이터 손실 명령 복원**
  - 추후 원활한 네트워크 통신을 위해 데이터 손실 명령 초기화
  - 명령어의 add 부분을 delete로 변경하여 복원

```
sudo tc qdisc delete dev lo root netem loss 45%
```

✓ Reliable 결과 확인

- **데이터 손실 명령 복원 확인**
  - Listener와 Talker를 다시 실행하여 데이터 손실 명령어가 제대로 복원 되었는지 확인하기

- **Listener**

```
user@user:~/qos_ws$ ros2 run demo_nodes_cpp listener_best_effort
[INFO] [1734408243.824953073] [listener]: I heard: [Hello World: 1]
[INFO] [1734408244.825688816] [listener]: I heard: [Hello World: 2]
[INFO] [1734408245.826127734] [listener]: I heard: [Hello World: 3]
[INFO] [1734408246.826121321] [listener]: I heard: [Hello World: 4]
[INFO] [1734408247.826074597] [listener]: I heard: [Hello World: 5]
[INFO] [1734408248.826097804] [listener]: I heard: [Hello World: 6]
[INFO] [1734408249.826065331] [listener]: I heard: [Hello World: 7]
[INFO] [1734408250.825935695] [listener]: I heard: [Hello World: 8]
[INFO] [1734408251.826235144] [listener]: I heard: [Hello World: 9]
[INFO] [1734408252.826329798] [listener]: I heard: [Hello World: 10]
[INFO] [1734408253.826224073] [listener]: I heard: [Hello World: 11]
```

- **Talker**

```
user@user:~/qos_ws$ ros2 run demo_nodes_cpp talker
[INFO] [1734408243.824671002] [talker]: Publishing: 'Hello World: 1'
[INFO] [1734408244.824851964] [talker]: Publishing: 'Hello World: 2'
[INFO] [1734408245.825079703] [talker]: Publishing: 'Hello World: 3'
[INFO] [1734408246.825107529] [talker]: Publishing: 'Hello World: 4'
[INFO] [1734408247.825186846] [talker]: Publishing: 'Hello World: 5'
[INFO] [1734408248.825150804] [talker]: Publishing: 'Hello World: 6'
[INFO] [1734408249.825076432] [talker]: Publishing: 'Hello World: 7'
[INFO] [1734408250.825018821] [talker]: Publishing: 'Hello World: 8'
[INFO] [1734408251.825231553] [talker]: Publishing: 'Hello World: 9'
[INFO] [1734408252.825253190] [talker]: Publishing: 'Hello World: 10'
[INFO] [1734408253.825154695] [talker]: Publishing: 'Hello World: 11'
```

QoS Programming
QoS 실습

Durability

Publisher가 보낸 메시지를 얼마나 오래 저장해서 새로운 Subscriber에게 줄 것인가

- Subscriber가 생성되기 전, 데이터를 사용할지 폐기할 지에 대한 QoS 옵션
- TRANSIENT_LOCAL : Publisher가 마지막으로 보낸 메시지를 메모리에 저장. 새로운 Subscriber가 연결되면 전달(Publisher에만 적용가능)
- VOLATILE : Subscriber가 연결되기 전 데이터는 사용하지 않고 버림. 새로 연결되면 새로운 메시지부터 받음
- 'py_pubsub' 패키지에서 QoS 프로파일을 변경하여 적용
- VOLATILE 로 설정하는 예제 실행

[활용 사례]

- TRANSIENT_LOCAL
  1. 새로운 Subscriber가 붙었을때 지금 현재 로봇의 상태를 알고 싶을 때
  2. SLAM 후 완성된 맵을 Publishing 하는 노드
  3. 나중에 들어오는 네비게이션 노드가 맵을 받아야 하는 경우

[활용 사례]

- VOLATILE
  1. 과거 영상 프레임은 의미가 없고 실시간 프레임만 받고 싶을때
  2. Publisher : /camera/image_raw
  3. Subscriber : 영상 Viewer

✓ py_pubsub/src/publisher_member_function.py 설정

- **퍼블리셔 QoS profile 변경**
  - History는 다시 KEEP_LAST로 변경
  - durability를 VOLATILE로 변경

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy

qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.RELIABLE,
    history=QoSHistoryPolicy.KEEP_LAST,
    depth=10,
    durability=QoSDurabilityPolicy.VOLATILE)


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', qos_profile=qos_profile)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1
```

- KEEP_LAST : 정해진 메시지 큐 사이즈만큼 데이터 보관
- VOLATILE : Subscriber가 생성되기 전 데이터는 사용하지 않음

✓ 빌드 후 py_pubsub 실행

- 빌드

```
colcon build
```

- Talker 실행

```
ros2 run py_pubsub talker
```

- Listener 실행 (Talker의 Publish 메시지가 10개 이상 발행된 뒤 실행)

```
ros2 run py_pubsub listener
```

✓ Durability 테스트 결과 확인

- Talker

```
ros2 run py_pubsub talker
2692] [minimal_publisher]: Publishing: "Hello World: 0"
7019] [minimal_publisher]: Publishing: "Hello World: 1"
0313] [minimal_publisher]: Publishing: "Hello World: 2"
1761] [minimal_publisher]: Publishing: "Hello World: 3"
7874] [minimal_publisher]: Publishing: "Hello World: 4"
0143] [minimal_publisher]: Publishing: "Hello World: 5"
0730] [minimal_publisher]: Publishing: "Hello World: 6"
8326] [minimal_publisher]: Publishing: "Hello World: 7"
2622] [minimal_publisher]: Publishing: "Hello World: 8"
5847] [minimal_publisher]: Publishing: "Hello World: 9"
9427] [minimal_publisher]: Publishing: "Hello World: 10"
3459] [minimal_publisher]: Publishing: "Hello World: 11"
8532] [minimal_publisher]: Publishing: "Hello World: 12"
4153] [minimal_publisher]: Publishing: "Hello World: 13"
8446] [minimal_publisher]: Publishing: "Hello World: 14"
7386] [minimal_publisher]: Publishing: "Hello World: 15"
3945] [minimal_publish
5330] [minimal_publish
1567] [minimal_publish
5898] [minimal_publish
4803] [minimal_publish
6834] [minimal_publish
7348] [minimal_publish
2605] [minimal_publish
9120] [minimal_publisher]: Publishing: "Hello World: 24"
3220] [minimal_publisher]: Publishing: "Hello World: 25"
```

- Listener

```
ros2 run py_pubsub listener
33516] [minimal_subscriber]: I heard: "Hello World: 17"
39660] [minimal_subscriber]: I heard: "Hello World: 18"
11015] [minimal_subscriber]: I heard: "Hello World: 19"
22808] [minimal_subscriber]: I heard: "Hello World: 20"
21839] [minimal_subscriber]: I heard: "Hello World: 21"
96456] [minimal_subscriber]: I heard: "Hello World: 22"
31389] [minimal_subscriber]: I heard: "Hello World: 23"
71447] [minimal_subscriber]: I heard: "Hello World: 24"
54366] [minimal_subscriber]: I heard: "Hello World: 25"
```

Durability	Depth 적용여부	메시지 저장방식
VOLATILE	X (무시됨)	메시지는 발행될 때만 존재, 새로운 구독자는 이전 메시지 받을 수 없음
TRANSIENT_LOCAL	O	최근 depth 개수만 유지

QoS Programming
QoS 실습

History와 Durability의 관계

	History	Durability
개념	▪ 얼마나 많은 메시지(몇 개)를 보관할까?	▪ 새로 등장한 Subscriber에게 이전 메시지를 줄지 안줄지를 결정
종류	▪ KEEP_LAST : 마지막 n개 저장(Depth) ▪ KEEP_ALL : 가능한 모든 메시지 저장	▪ VOLATILE : 새로운 메시지만 받음 ▪ TRANSIENT_LOCAL : Publisher가 최근 발행한 메시지 저장하고 나중에 등장한 Subscriber에게 메시지 보내줌(개수는 History에서 결정)
예시	▪ VOLOTILE + KEEP_LAST(10) → Publisher는 최근 10개 메시지 저장. New Subscriber가 나중에 붙으면? 과거 데이터 못 받음 ▪ TRANSIENT_LOCAL + KEEP_LAST(10) → Publisher는 최근 10개 메시지 저장. New Subscriber가 나중에 붙으면? 최근 10개중 가능한 메시지 다시 보내줌	
사례	▪ 센서 스트리밍(LiDAR등) → VOLATILE + KEEP_LAST(depth 적당히) ▪ 중요한 공지사항(경고/에러메시지) → TRANSIENT_LOCAL + KEEP_LAST(1 ~ 몇 개) ▪ 초기 설정 정보(맵 데이터, Config) → TRANSIENT_LOCAL + KEEP_ALL(메모리 여유 있는 경우)	

※ Durability가 TRANSIENT_LOCAL이어야 New Subscriber가 과거 메시지를 받을 수 있다! 얼마나 받을 수 있는지는 History(Depth)에 따라!

QoS Programming
QoS 실습

Deadline

정해진 시간 안에 메시지가 도착해야 한다. 설정한 시간 내 도착하지 않으면 이벤트 발생

- 정해진 주기 내 데이터의 발신 및 수신이 없는 경우, EventCallback 함수를 실행하는 QoS 옵션
- ROS2의 기본 패키지 quality_of_service_demo의 deadline.py 예제 살펴보기
- /opt/ros/humble/lib/python3.10/site-packages/quality_of_service_demo_py/deadline.py

[활용 사례]

- 100ms마다 센서 값을 Publishing하는 노드가 센서 고장으로 1초 동안 데이터를 못 보낸다면
- Subscriber는 센서가 문제 있다는 것을 알 수 있음 → 센서 데이터 감시
- 로봇의 Heartbeat 또는 제어 명령 : Robot이 일정 주기로 살아있다는 메시지를 보내야 하거나 500ms 이상 끊어지면 로봇은 안정상 멈춰야 할 수도 있음.
- 500ms 동안 새로운 명령이 없으면 로봇이 Emergency Stop 모드로 전환해야 할 수도
- 환자의 Vital Sign 데이터를 주기적으로 받아야 하는 모니터링 시스템 → 데이터가 일정 주기 이상 끊어지면 즉시 경고 발생

✓ deadline.py – main 함수

```python
def main(args=None):
    parsed_args = parse_args()
    rclpy.init(args=args)

    topic = 'qos_deadline_chatter'
    deadline = Duration(seconds=parsed_args.deadline / 1000.0)

    qos_profile = QoSProfile(
        depth=10,
        deadline=deadline)

    subscription_callbacks = SubscriptionEventCallbacks(
        deadline=lambda event: get_logger('Listener').info(str(event)))
    listener = Listener(topic, qos_profile, event_callbacks=subscription_callbacks)

    publisher_callbacks = PublisherEventCallbacks(
        deadline=lambda event: get_logger('Talker').info(str(event)))
    talker = Talker(topic, qos_profile, event_callbacks=publisher_callbacks)

    publish_for_seconds = parsed_args.publish_for / 1000.0
    pause_for_seconds = parsed_args.pause_for / 1000.0
    pause_timer = talker.create_timer(  # noqa: F841
        publish_for_seconds,
        lambda: talker.pause_for(pause_for_seconds))

    executor = SingleThreadedExecutor()
    executor.add_node(listener)
    executor.add_node(talker)
    try:
        executor.spin()
    except KeyboardInterrupt:
        pass
    except ExternalShutdownException:
        sys.exit(1)
    finally:
        rclpy.try_shutdown()

```

deadline.py [Read-Only]
/opt/ros/humble/lib/python3.10/site-packages/quality_of_service_demo_py

✓deadline.py 분석

- **주기 설정**
  - 'parsed_args.deadline'을 통해 사용자가 정의한 주기를 설정할 수 있음
  - 'Duration' 객체로 주기 정보를 갖는 변수 'deadline'의 선언
  - 만약, deadline을 500으로 설정 시, 500/1000초 = 0.5초로 설정되는 코드

```
deadline = Duration(seconds=parsed_args.deadline / 1000.0)
```

- **QoS 프로파일 설정**
  - deadline 인자에, 'Duration' 객체로 선언된 변수 'deadline' 할당

```
qos_profile = QoSProfile(
    depth=10,
    deadline=deadline)
```

✓deadline.py 분석

- Callback 함수 선언 및 Talker, Listener 선언

```
subscription_callbacks = SubscriptionEventCallbacks(
    deadline=lambda event: get_logger('Listener').info(str(event)))
listener = Listener(topic, qos_profile, event_callbacks=subscription_callbacks)

publisher_callbacks = PublisherEventCallbacks(
    deadline=lambda event: get_logger('Talker').info(str(event)))
talker = Talker(topic, qos_profile, event_callbacks=publisher_callbacks)
```

- Timer로 Publishing/Pause 반복 설정

```
publish_for_seconds = parsed_args.publish_for / 1000.0
pause_for_seconds = parsed_args.pause_for / 1000.0
pause_timer = talker.create_timer(  # noqa: F841
    publish_for_seconds,
    lambda: talker.pause_for(pause_for_seconds))
```

- Talker가 publish_for_seconds 동안 메시지를 보낸 다음 pause_for_seconds 동안 발행을 멈춤 → 의도적으로 deadline 위반상황 발생시킴

✓터미널에서 Deadline 데모 실행

- Deadline 0.7초, 데이터 발행 기간 3초, 일시 정지 0초로 설정해보자

```
ros2 run quality_of_service_demo_py deadline 700 --publish-for 3000 --pause-for 0
```

- deadline 700 : deadline 시간 700ms
- publish-for : talker가 몇 초간 발행할지
- pause-for : talker가 몇 초간 멈출지
- 실행 시 /qos_talker와 /qos_listener 노드가 함께 실행
- 0.7초 내 데이터가 수신되지 않을 경우 EventCallback 함수 실행
- 3초 동안 데이터가 발신되고 0초 동안 일시정지이기 때문에, 결국 쉴 틈없이 데이터를 발신

✓터미널에서 Deadline 데모 실행

- Deadline 0.7초, 데이터 발행 기간 3초, 일시 정지 0초로 설정해보자

```
ros2 run quality_of_service_demo_py deadline 700 --publish-for 3000 --pause-for 0
```

- 실행 결과
  - 데이터가 쉴 틈없이 발신 및 수신되기 때문에 이벤트 함수 호출이 되지 않음

```
[INFO] [1728979614.643792502] [qos_listener]: Subscription created
[INFO] [1728979614.645001165] [qos_talker]: Talker starting up
[INFO] [1728979615.145716049] [qos_talker]: Publishing: Talker says 0
[INFO] [1728979615.146114788] [qos_listener]: I heard: Talker says 0
[INFO] [1728979615.645723276] [qos_talker]: Publishing: Talker says 1
[INFO] [1728979615.646078988] [qos_listener]: I heard: Talker says 1
[INFO] [1728979616.145720307] [qos_talker]: Publishing: Talker says 2
[INFO] [1728979616.146112958] [qos_listener]: I heard: Talker says 2
[INFO] [1728979616.645710427] [qos_talker]: Publishing: Talker says 3
[INFO] [1728979616.646059055] [qos_listener]: I heard: Talker says 3
[INFO] [1728979617.145752251] [qos_talker]: Publishing: Talker says 4
[INFO] [1728979617.146148825] [qos_listener]: I heard: Talker says 4
[INFO] [1728979617.645710215] [qos_talker]: Publishing: Talker says 5
[INFO] [1728979617.646115721] [qos_listener]: I heard: Talker says 5
[INFO] [1728979617.646266967] [qos_talker]: Publishing: Talker says 6
[INFO] [1728979617.646548757] [qos_listener]: I heard: Talker says 6
[INFO] [1728979618.146882377] [qos_talker]: Publishing: Talker says 7
[INFO] [1728979618.147269039] [qos_listener]: I heard: Talker says 7
[INFO] [1728979618.646865281] [qos_talker]: Publishing: Talker says 8
```

✓터미널에서 Deadline 데모 실행

- Deadline 0.7초, 데이터 발행 기간 3초, 일시 정지 1초로 설정해보자

```
ros2 run quality_of_service_demo_py deadline 700 --publish-for 3000 --pause-for 1000
```

  - 일시정지 시간을 늘려 인위적으로 deadline을 넘어보자
  - 즉, 3초 동안 발행하다 1초 동안 쉬게 된다면 어떤 결과가 나오는지 확인

✓터미널에서 Deadline 데모 실행

- Deadline 0.7초, 데이터 발행 기간 3초, 일시 정지 1초로 설정해보자

```
ros2 run quality_of_service_demo_py deadline 700 --publish-for 3000 --pause-for 1000
```

- 실행 결과

```
[INFO] [1728979910.568033879] [qos_listener]: Subscription created
[INFO] [1728979910.569117902] [qos_talker]: Talker starting up
[INFO] [1728979911.069839009] [qos_talker]: Publishing: Talker says 0
[INFO] [1728979911.070248183] [qos_listener]: I heard: Talker says 0
[INFO] [1728979911.569822171] [qos_talker]: Publishing: Talker says 1
[INFO] [1728979911.570199056] [qos_listener]: I heard: Talker says 1
[INFO] [1728979912.069815774] [qos_talker]: Publishing: Talker says 2
[INFO] [1728979912.070167777] [qos_listener]: I heard: Talker says 2
[INFO] [1728979912.569827017] [qos_talker]: Publishing: Talker says 3
[INFO] [1728979912.570201189] [qos_listener]: I heard: Talker says 3
[INFO] [1728979913.069814375] [qos_talker]: Publishing: Talker says 4
[INFO] [1728979913.070203470] [qos_listener]: I heard: Talker says 4
[INFO] [1728979913.569806097] [qos_talker]: Publishing: Talker says 5
[INFO] [1728979913.570202932] [qos_listener]: I heard: Talker says 5
[INFO] [1728979914.269864640] [Listener]: <rclpy._rclpy_pybind11.rmw_requested_deadline_missed_status_t object at 0x72ec69007170>
[INFO] [1728979914.270059601] [Talker]: <rclpy._rclpy_pybind11.rmw_offered_deadline_missed_status_t object at 0x72ec669d2970>
[INFO] [1728979914.570503123] [qos_talker]: Publishing: Talker says 6
[INFO] [1728979914.570870628] [qos_listener]: I heard: Talker says 6
[INFO] [1728979915.071161502] [qos_talker]: Publishing: Talker says 7
[INFO] [1728979915.071556943] [qos_listener]: I heard: Talker says 7
[INFO] [1728979915.571153927] [qos_talker]: Publishing: Talker says 8
[INFO] [1728979915.571542934] [qos_listener]: I heard: Talker says 8
[INFO] [1728979916.071206836] [qos_talker]: Publishing: Talker says 9
[INFO] [1728979916.071634935] [qos_listener]: I heard: Talker says 9
```

[활용]

Deadline QoS 설정을 사용하면,

토픽을 정해진 시간 안에 Publishing 못하거나 Subscribing하지 못할 때,

이벤트 콜백 함수 호출하여 특정 루틴을 수행하게 할 수 있다.

QoS Programming
QoS 실습

Lifespan

메시지의 유효기간(수명)을 지정하는 옵션. Publisher가 데이터 발행한 순간부터 lifespan이 지나면 그 데이터는 무효

- 정해진 주기 내 수신되는 데이터만 유효 판정, 이외 데이터는 삭제하는 QoS 옵션
- ROS2의 기본 패키지 quality_of_service_demo의 lifespan.py 예제 살펴보기
- /opt/ros/humble/lib/python3.10/site-packages/quality_of_service_demo_py/lifespan.py

[활용 사례]

- 카메라 프레임, LiDAR 스캔을 오래된 데이터를 받으면 의미 없으므로 200ms이상된 데이터는 버리고 최신 데이터만 받고 싶을때
- 로봇의 위치정보(/odom)같은 경우 1초 이상된 데이터는 현재 위치와 차이가 있으므로 버려야 할 수도 있음
- 장비 상태 알림이 5초 이상 지연되면 의미 없을 수도 있으므로 lifespan을 5초로 설정
- 로봇 팔을 움직이는 명령은 1초가 지난 명령인 경우 무시

✓ lifespan.py – main 함수

```
46  def main(args=None):
47      parsed_args = parse_args()
48      rclpy.init(args=args)
49
50      topic = 'qos_lifespan_chatter'
51      lifespan = Duration(seconds=parsed_args.lifespan / 1000.0)
52
53      qos_profile = QoSProfile(
54          depth=parsed_args.history,
55          # Guaranteed delivery is needed to send messages to late-joining subscription.
56          reliability=QoSReliabilityPolicy.RELIABLE,
57          # Store messages on the publisher so that they can be affected by Lifespan.
58          durability=QoSDurabilityPolicy.TRANSIENT_LOCAL,
59          lifespan=lifespan)
60
61      listener = Listener(
62          topic, qos_profile, event_callbacks=None, defer_subscribe=True)
63      talker = Talker(
64          topic, qos_profile, event_callbacks=None, publish_count=parsed_args.publish_count)
65      subscribe_timer = listener.create_timer(  # noqa: F841
66          parsed_args.subscribe_after / 1000.0,
67          lambda: listener.start_listening())
68
69      executor = SingleThreadedExecutor()
70      executor.add_node(listener)
71      executor.add_node(talker)
72      executor.spin()
73
74      rclpy.shutdown()
75
```

✓lifespan.py 분석

▪ 주기 설정

- 'parsed_args.lifespan'을 통해 사용자가 정의한 주기를 설정할 수 있음
- 'Duration' 객체로 주기 정보를 갖는 변수 ' lifespan'의 선언
- 만약, lifespan을 500으로 설정 시, 500/1000초 = 0.5초로 설정되는 코드

```
lifespan = Duration(seconds=parsed_args.lifespan / 1000.0)
```

▪ QoS 프로파일 설정

- lifespan 인자에, 'Duration' 객체로 선언된 변수 'lifespan' 할당
- reliability는 RELIABLE(반드시 전달), durability는 TRANSIENT_LOCAL(New Subscriber에게 데이터 전달)로 설정

```
qos_profile = QoSProfile(
    depth=parsed_args.history,
    # Guaranteed delivery is needed to send messages to late-joining subscription.
    reliability=QoSReliabilityPolicy.RELIABLE,
    # Store messages on the publisher so that they can be affected by Lifespan.
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL,
    lifespan=lifespan)
```

✓터미널에서 Lifespan 데모 실행

- Lifespan 1초, 데이터 발행 개수 10개, 3초 후 Listener 시작

```
ros2 run quality_of_service_demo_py lifespan 1000 --publish-count 10 --subscribe-after 3000
```

  - 실행 시 /qos_talker와 /qos_listener 노드가 함께 실행
  - 1초 안에 수신되는 데이터만 유효 판정, 이외 데이터는 publisher의 메시지 큐에서 삭제
  - Talker는 순차적으로 10개의 데이터를 발행
  - Listener는 3초 후 시작하도록 설정
  - 즉, Lifespan이 1초로 설정되어 있기 때문에 **‘4’ Publishing 후 Listener가 시작**되어도 Listener가 ‘4’를 받을 수 있음
  - 1초 이전의 데이터들은 모두 삭제되어 수신 받지 못함
  - Lifespan을 2000으로 변경해서 테스트 해보기

✓터미널에서 Lifespan 데모 실행

- Lifespan 실행 결과

```
[INFO] [1728984671.765202664] [qos_talker]: Talker starting up
[INFO] [1728984672.266332766] [qos_talker]: Publishing: Talker says 0
[INFO] [1728984672.766260751] [qos_talker]: Publishing: Talker says 1
[INFO] [1728984673.266378328] [qos_talker]: Publishing: Talker says 2
[INFO] [1728984673.766228295] [qos_talker]: Publishing: Talker says 3
[INFO] [1728984674.266326264] [qos_talker]: Publishing: Talker says 4
[INFO] [1728984674.766260774] [qos_talker]: Publishing: Talker says 5
[INFO] [1728984674.766906211] [qos_listener]: Subscription created
[INFO] [1728984674.767209480] [qos_listener]: I heard: Talker says 4
[INFO] [1728984674.767458167] [qos_listener]: I heard: Talker says 5
[INFO] [1728984675.266224899] [qos_talker]: Publishing: Talker says 6
[INFO] [1728984675.266609840] [qos_listener]: I heard: Talker says 6
[INFO] [1728984675.766271350] [qos_talker]: Publishing: Talker says 7
[INFO] [1728984675.766682453] [qos_listener]: I heard: Talker says 7
[INFO] [1728984676.266141444] [qos_talker]: Publishing: Talker says 8
[INFO] [1728984676.266525042] [qos_listener]: I heard: Talker says 8
[INFO] [1728984676.766263144] [qos_talker]: Publishing: Talker says 9
[INFO] [1728984676.766649487] [qos_listener]: I heard: Talker says 9
```

  - Listener는 ‘5’가 발행된 이후에도 Talker의 ‘4, 5’를 성공적으로 수신 받았음을 확인

✓터미널에서 Lifespan 데모 실행

```
victor@victor-VirtualBox:~$ ros2 run quality_of_service_demo_py lifespan 4000 --publish-count 10 --subscribe-after 3000
[INFO] [1743477320.839639082] [qos_talker]: Talker starting up
[INFO] [1743477321.341852584] [qos_talker]: Publishing: Talker says 0
[INFO] [1743477321.842253050] [qos_talker]: Publishing: Talker says 1
[INFO] [1743477322.342414102] [qos_talker]: Publishing: Talker says 2
[INFO] [1743477322.842446754] [qos_talker]: Publishing: Talker says 3
[INFO] [1743477323.342325271] [qos_talker]: Publishing: Talker says 4
[INFO] [1743477323.842620723] [qos_listener]: Subscription created
[INFO] [1743477323.843199324] [qos_talker]: Publishing: Talker says 5
[INFO] [1743477323.843847083] [qos_listener]: I heard: Talker says 0
[INFO] [1743477323.844218694] [qos_listener]: I heard: Talker says 1
[INFO] [1743477323.844492144] [qos_listener]: I heard: Talker says 2
[INFO] [1743477323.844757852] [qos_listener]: I heard: Talker says 3
[INFO] [1743477323.845037606] [qos_listener]: I heard: Talker says 4
[INFO] [1743477323.845306814] [qos_listener]: I heard: Talker says 5
[INFO] [1743477324.342036509] [qos_talker]: Publishing: Talker says 6
[INFO] [1743477324.342813334] [qos_listener]: I heard: Talker says 6
[INFO] [1743477324.842491217] [qos_talker]: Publishing: Talker says 7
[INFO] [1743477324.843740544] [qos_listener]: I heard: Talker says 7
[INFO] [1743477325.342128776] [qos_talker]: Publishing: Talker says 8
[INFO] [1743477325.344034774] [qos_listener]: I heard: Talker says 8
[INFO] [1743477325.842443982] [qos_talker]: Publishing: Talker says 9
[INFO] [1743477325.843560262] [qos_listener]: I heard: Talker says 9
^CTraceback (most recent call last):
  File "/opt/ros/humble/lib/quality_of_service_demo_py/lifespan", line 33, in <module>
```

QoS Programming
QoS 실습

Liveliness

Publisher가 아직 살아있음을 Subscriber에게 어떻게 보장할지 정하는 방법

- 정해진 주기 내 노드 또는 토픽의 생사를 확인하는 QoS 옵션
- Publisher가 여전히 활성 상태인지를 Subscriber가 확인할 수 있도록 하는 QoS 정책
- 특정 시간 동안 응답하지 않으면 “비활성화됨“ 상태로 판단함(liveliness_lease_duration 안에 최소 1번은 신호를 보내야 함)
- Liveliness 설정 값(자동 또는 매뉴얼로 확인할 지 결정)
  - AUTOMATIC : 기본 옵션. Publisher 가 메시지를 보낼 때 자동으로 활성상태로 간주됨(RMW가 알아서 Publisher를 감시)
  - MANUAL_BY_TOPIC : Publisher가 특정 주기마다 DDS에게 “나는 살아있다 ” 는 신호를 보내는 방식(rclpy.assert_liveliness 메서드를 호출)
- ROS2의 기본 패키지 quality_of_service_demo의 liveliness.py 예제 살펴보기 GitHub
- /opt/ros/humble/lib/python3.10/site-packages/quality_of_service_demo_py/liveliness.py

[활용 사례]

- 로봇 제어 명령 감시 : Publisher가 죽었을 경우 즉시 로봇 모터를 멈춤
- 센서 데이터 생존 확인 : 센서로 부터 데이터 전송이 없으면 즉시 다른 예비 센서로 스위치 또는 경고 메시지
- 멀티 로봇 통신 안정성 : 각 로봇/드론이 서로 위치/상태를 공유하는 경우 특정 로봇이 통신에서 사라지면 즉시 알아채서 경로 재설정 또는 팀 전략 수정
- 시스템 운영 중 어떤 노드가 죽으면 빠르게 디버깅 해야함

GitHub https://github.com/ros2/rclpy/blob/rolling/rclpy/rclpy/publisher.py

✓ liveliness.py – main 함수

/opt/ros/humble/lib/python3.10/site-packages/quality_of_service_demo_py/liveliness.py

```python
def main(args=None):
    parsed_args = parse_args()
    rclpy.init(args=args)

    topic = 'qos_liveliness_chatter'
    liveliness_lease_duration = Duration(seconds=parsed_args.liveliness_lease_duration / 1000.0)
    liveliness_policy = POLICY_MAP[parsed_args.policy]

    qos_profile = QoSProfile(
        depth=10,
        liveliness=liveliness_policy,
        liveliness_lease_duration=liveliness_lease_duration)

    subscription_callbacks = SubscriptionEventCallbacks(
        liveliness=lambda event: get_logger('Listener').info(str(event)))
    listener = Listener(topic, qos_profile, event_callbacks=subscription_callbacks)

    publisher_callbacks = PublisherEventCallbacks(
        liveliness=lambda event: get_logger('Talker').info(str(event)))
    talker = Talker(
        topic, qos_profile,
        event_callbacks=publisher_callbacks,
        assert_topic_period=parsed_args.topic_assert_period / 1000.0)

    executor = SingleThreadedExecutor()

    def kill_talker():
        if liveliness_policy == QoSLivelinessPolicy.AUTOMATIC:
            executor.remove_node(talker)
            talker.destroy_node()
        elif liveliness_policy == QoSLivelinessPolicy.MANUAL_BY_TOPIC:
            talker.stop()
        kill_timer.cancel()

    if parsed_args.kill_publisher_after > 0:
        kill_timer = listener.create_timer(  # noqa: F841
            parsed_args.kill_publisher_after / 1000.0,
            kill_talker)

    executor.add_node(listener)
    executor.add_node(talker)
    executor.spin()

    rclpy.shutdown()

```

[Liveliness의 역할]

Liveliness는 다음과 같은 상황에서 중요하게 사용됩니다.

1. **실시간 시스템에서 중요 정보 감지**
   - 예를 들어, 자율 주행 자동차에서 센서 데이터를 발행하는 노드가 멈추면 즉시 이를 감지하고 적절한 조치를 취할 수 있음.
2. **발행자의 상태 모니터링**
   - 발행자가 정상적으로 데이터를 제공하고 있는지 확인하는 역할
3. **데이터 갱신 주기 보장**
   - 특정 시간 내에 발행자가 데이터를 보내지 않으면 구독자는 이를 감지하고 대체 데이터 소스를 사용할 수도 있음.

✓liveliness.py 분석

- **주기 설정**
  - 'parsed_args.liveliness_lease_duration'을 통해 사용자가 정의한 주기를 설정할 수 있음
  - 'Duration' 객체로 주기 정보를 갖는 변수 ' lifespan'의 선언
  - 'POLICY_MAP' 딕셔너리에서 사용자가 입력한 정책(parsed_args.policy)을 가져옴

```
liveliness_lease_duration = Duration(seconds=parsed_args.liveliness_lease_duration / 1000.0)
liveliness_policy = POLICY_MAP[parsed_args.policy]
```

- **QoS 프로파일 설정**
  - 동일한 qos_profile을 사용
  - depth=10, Liveliness는 명령어로 실행 시 설정할 수 있도록 함

```
qos_profile = QoSProfile(
    depth=10,
    liveliness=liveliness_policy,
    liveliness_lease_duration=liveliness_lease_duration)

subscription_callbacks = SubscriptionEventCallbacks(
    liveliness=lambda event: get_logger('Listener').info(str(event)))

listener = Listener(topic, qos_profile, event_callbacks=subscription_callbacks)

publisher_callbacks = PublisherEventCallbacks(
    liveliness=lambda event: get_logger('Talker').info(str(event)))

talker = Talker(
    topic, qos_profile,
    event_callbacks=publisher_callbacks,
    assert_topic_period=parsed_args.topic_assert_period / 1000.0)
```

✓터미널에서 Liveliness 데모 실행

- Liveliness 1초 설정, 실행 2초 후 퍼블리셔 노드 종료, 자동으로 확인

```
ros2 run quality_of_service_demo_py liveliness 1000 --kill-publisher-after 2000 --policy AUTOMATIC
```

- 실행 결과

```
victor@victor-VirtualBox:~$ ros2 run quality_of_service_demo_py liveliness 1000 --kill-publisher-after 2000 --policy AUTOMATIC
[INFO] [1743478003.094778676] [qos_listener]: Subscription created
[INFO] [1743478003.098157683] [qos_talker]: Talker starting up
[INFO] [1743478003.600229111] [qos_talker]: Publishing: Talker says 0
[INFO] [1743478003.602301301] [Listener]: <rclpy._rclpy_pybind11.rmw_liveliness_changed_status_t object at 0x75a97b3230b0>
[INFO] [1743478003.603255849] [qos_listener]: I heard: Talker says 0
[INFO] [1743478004.099515275] [qos_talker]: Publishing: Talker says 1
[INFO] [1743478004.100265316] [qos_listener]: I heard: Talker says 1
[INFO] [1743478004.599417555] [qos_talker]: Publishing: Talker says 2
[INFO] [1743478004.600227397] [qos_listener]: I heard: Talker says 2
[INFO] [1743478005.099176900] [qos_talker]: Publishing: Talker says 3
[INFO] [1743478005.101406609] [Listener]: <rclpy._rclpy_pybind11.rmw_liveliness_changed_status_t object at 0x75a97b3236b0>
[INFO] [1743478005.101650086] [qos_listener]: I heard: Talker says 3
```

- Publisher가 1000ms마다 메시지를 Publishing
- 2초가 지나면 퍼블리셔 노드인 qos_talker가 종료
- 이때 Listener는 liveliness로 설정한 1초 주기 동안 노드가 죽었다는 것을 자동으로 확인

✓터미널에서 Liveliness 데모 실행

- Liveliness 1초 설정, 실행 2초 후 퍼블리셔 노드 종료, 수동으로 확인

```
ros2 run quality_of_service_demo_py liveliness 1000 --kill-publisher-after 2000 --policy MANUAL_BY_TOPIC
```

- 실행 결과

```
[INFO] [1728986123.146376915] [qos_listener]: Subscription created
[INFO] [1728986123.147483087] [qos_talker]: Talker starting up
[INFO] [1728986123.648214225] [qos_talker]: Publishing: Talker says 0
[INFO] [1728986123.648669166] [Listener]: <rclpy._rclpy_pybind11.rmw_liveliness_changed_status_t object at 0x7d1897bd12f0>
[INFO] [1728986123.648861774] [qos_listener]: I heard: Talker says 0
[INFO] [1728986124.148209272] [qos_talker]: Publishing: Talker says 1
[INFO] [1728986124.148612804] [qos_listener]: I heard: Talker says 1
[INFO] [1728986124.648202674] [qos_talker]: Publishing: Talker says 2
[INFO] [1728986124.648788037] [qos_listener]: I heard: Talker says 2
[INFO] [1728986125.648243245] [Listener]: <rclpy._rclpy_pybind11.rmw_liveliness_changed_status_t object at 0x7d1897bd2030>
[INFO] [1728986125.648473114] [Talker]: <rclpy._rclpy_pybind11.rmw_liveliness_lost_status_t object at 0x7d1897bd21b0>
```

- AUTOMATIC 때와 비슷하나, 노드를 죽이지 않고 퍼블리시만 되지 않도록 설정했을 때의 결과가 위와 같음

QoS Programming
QoS 정리

[Duration 객체]

Duration은 ROS2에서 시간 간격(time span)을 표현하는 전용 타입

1. 로봇처럼 실시간성이 중요한 시스템에서는 1ms 미만의 정밀한 시간 제어 필요
2. 단순한 시간의 길이. 내부적으로는 nanoseconds 단위로 관리
3. 명확성 : float 데이터 타입을 사용하면 단위에 대한 혼란(sec/ms) 발생
4. 일관성 : QoS설정은 모두 Duration 사용
5. 성능 최적화 : RMW는 nano초 기반 시간 계산을 빠르게 수행

```python
from rclpy.duration import Duration

d = Duration(seconds=3, nanoseconds=500000000)  # 3.5초
print(d.nanoseconds)  # 3.5초를 나노초로 환산
```

deadline.py | lifespan.py

opt > ros > humble > lib > python3.10 > site-packages > quality_of_service_demo_py > deadline.py > main

```python
import argparse
import sys

from quality_of_service_demo_py.common_nodes import Listener
from quality_of_service_demo_py.common_nodes import Talker

import rclpy
from rclpy.duration import Duration
from rclpy.executors import ExternalShutdownException
from rclpy.executors import SingleThreadedExecutor
from rclpy.logging import get_logger
from rclpy.qos import QoSProfile
from rclpy.qos_event import PublisherEventCallbacks
from rclpy.qos_event import SubscriptionEventCallbacks
```

```python
deadline = Duration(seconds=parsed_args.deadline / 1000.0)
```

deadline.py | lifespan.py

opt > ros > humble > lib > python3.10 > site-packages > quality_of_service_demo_py > lifespan.py >

```python
import argparse

from quality_of_service_demo_py.common_nodes import Listener
from quality_of_service_demo_py.common_nodes import Talker

import rclpy
from rclpy.duration import Duration
from rclpy.executors import SingleThreadedExecutor
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy
```

```python
lifespan = Duration(seconds=parsed_args.lifespan / 1000.0)
```

QoS Programming
QoS 정리

QoS	설명	옵션
Reliability	UDP처럼 통신 속도를 최우선 할지 TCP처럼 데이터 손실 방지하며 신뢰도를 우선할지	▪ BEST_EFFORT(속도 우선) ▪ RELIABLE(신뢰도 우선)
History	통신 상태에 따라 정해진 사이즈 만큼의 데이터를 보관	▪ KEEP_LAST ▪ KEEP_ALL
Durability	데이터를 수신하는 Subscriber가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 설정	▪ TRANSIENT_LOCAL ▪ VOLATILE(휘발성)
Deadline	정해진 주기 안데 데이터가 발신 및 수신되지 않을 경우 이벤트 함수를 실행시킴	deadline_duration(단위:ms) (700, 1000등)
Lifespan	정해진 주기 안에서 수신되는 데이터만 유효판정하고 그렇지 않은 데이터는 삭제	lifespan_duration(단위:ms) (700, 1000 등)
Liveliness	정해진 주기 안에서 노드 혹은 토픽의 생사를 확인	Liveliness(AUTOMATIC, MANUAL_BY_TOPIC)

ros2 component

- **Component는 실행 중인 컨테이너와 컴포넌트 목록을 확인하거나 실행 및 중지를 할 수 있는 명령어**
  - 실행 중인 컨테이너와 컴포넌트 목록 출력

```
$ ros2 component list
```

  - 지정 컨테이너 노드의 특정 컴포넌트 실행

```
$ ros2 component load /my_container composition composition::Talker
```

  - 표준 컨테이너 노드로 특정 컴포넌트 실행

```
$ ros2 component standalone composition composition::Talker
```

  - 사용 가능한 컴포넌트들의 목록 출력

```
$ ros2 component types
```

  - 지정 컴포넌트의 실행 중지

```
$ ros2 component unload /my_container 1
```

- 컴포넌트 노드는 여러 노드를 하나의 프로세스 내에서 실행할 수 있도록 설계된 ROS2의 기능
- 이를 통해 노드 간의 통신 오버헤드를 줄이고, 시스템의 자원 활용을 최적화할 수 있음
- 노드 수가 많아서 프로세스 수를 줄이고 싶을때, 통신 성능이 매우 중요한 경우
- p39 참조

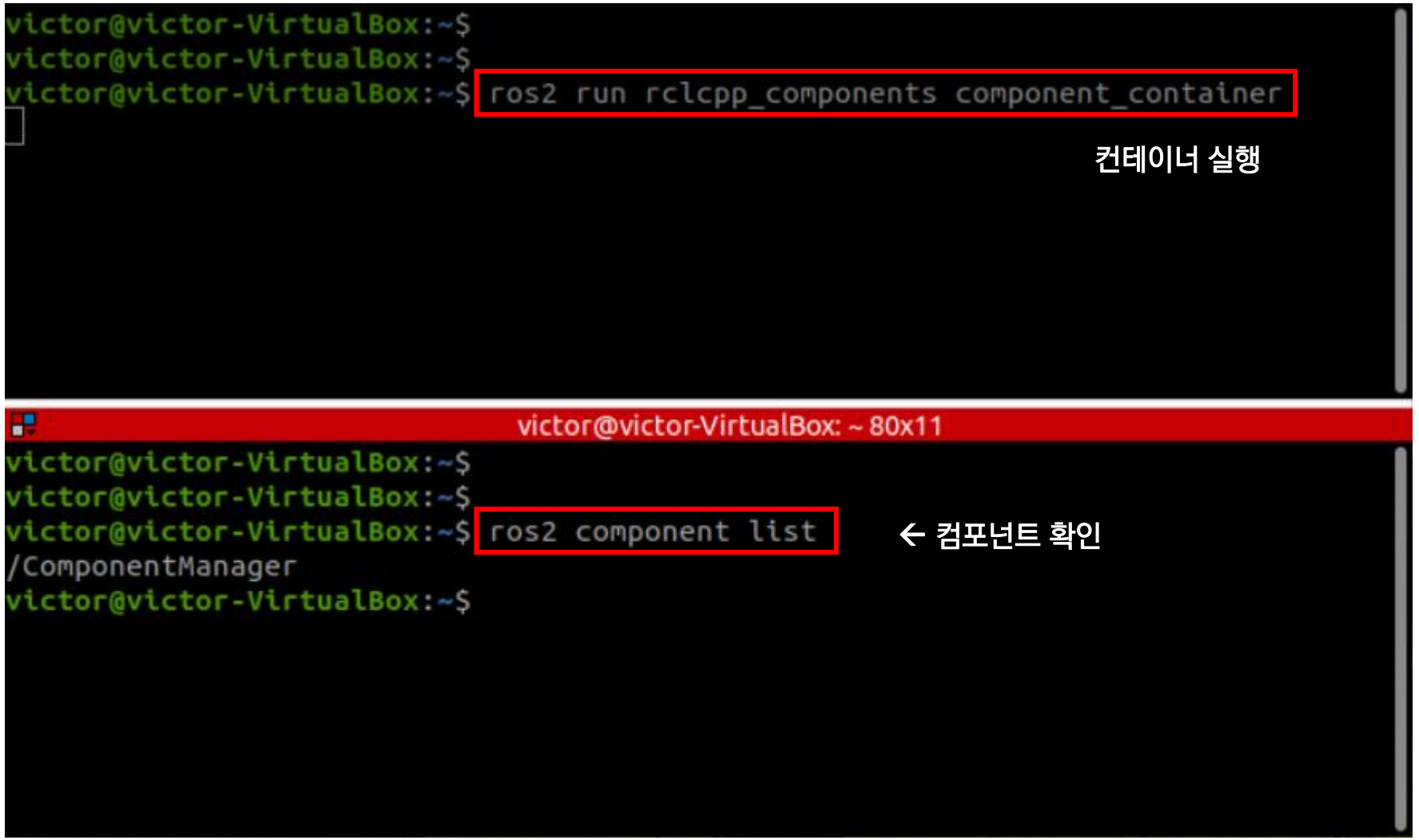
- 예제를 위해 다음 명령어로 component 예제 패키지의 런치 파일 실행
- ros2 launch composition composition_demo.launch.py

ROS2 CLI

ROS2 CLI 실습

교재 : ROS2로 시작하는 로봇 프로그래밍 (p533)

ros2 component



ROS2 CLI

ROS2 CLI 실습

```
victor@victor-VirtualBox:~$ ros2 run rclcpp_components component_container
[INFO] [1743338846.016664223] [ComponentManager]: Load Library: /opt/ros/humble/lib/libtalker_component.so
[INFO] [1743338846.018912454] [ComponentManager]: Found class: rclcpp_components::NodeFactoryTemplate<composition::Talke
r>
[INFO] [1743338846.018967075] [ComponentManager]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition:
:Talker>
[INFO] [1743338847.577330923] [talker]: Publishing: 'Hello World: 1'
[INFO] [1743338848.280206971] [talker]: Publishing: 'Hello World: 2'
[INFO] [1743338849.275903837] [talker]: Publishing: 'Hello World: 3'
[INFO] [1743338850.346059190] [talker]: Publishing: 'Hello World: 4'
[INFO] [1743338851.708013633] [talker]: Publishing: 'Hello World: 5'
[INFO] [1743338852.029732080] [talker]: Publishing: 'Hello World: 6'
[INFO] [1743338853.180381306] [talker]: Publishing: 'Hello World: 7'
[INFO] [1743338854.462108913] [talker]: Publishing: 'Hello World: 8'
[INFO] [1743338855.027892827] [talker]: Publishing: 'Hello World: 9'
[INFO] [1743338856.233668117] [talker]: Publishing: 'Hello World: 10'
[INFO] [1743338857.081915740] [talker]: Publishing: 'Hello World: 11'
[INFO] [1743338858.788361404] [talker]: Publishing: 'Hello World: 12'
```

```
victor@victor-VirtualBox: ~ 120x9
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$ ros2 component list
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'
victor@victor-VirtualBox:~$
```

- Talker 컴포넌트를 컨테이너에 적재
- 적재가 완료되면 컨테이너를 실행시켰던 터미널창에 Publishing 됨

ROS2 CLI

ROS2 CLI 실습

```
[INFO] [1743339312.323152684] [ComponentManager]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition::Listener>
[INFO] [1743339312.405975194] [talker]: Publishing: 'Hello World: 6'
[INFO] [1743339312.406127393] [listener]: I heard: [Hello World: 6]
[INFO] [1743339313.409002298] [talker]: Publishing: 'Hello World: 7'
[INFO] [1743339313.409160747] [listener]: I heard: [Hello World: 7]
[INFO] [1743339314.406949710] [talker]: Publishing: 'Hello World: 8'
[INFO] [1743339314.407381248] [listener]: I heard: [Hello World: 8]
[INFO] [1743339315.406509305] [talker]: Publishing: 'Hello World: 9'
[INFO] [1743339315.406805908] [listener]: I heard: [Hello World: 9]
[INFO] [1743339316.406694343] [talker]: Publishing: 'Hello World: 10'
[INFO] [1743339316.406839701] [listener]: I heard: [Hello World: 10]
[INFO] [1743339317.406446281] [talker]: Publishing: 'Hello World: 11'
[INFO] [1743339317.408092513] [listener]: I heard: [Hello World: 11]
[INFO] [1743339318.691158891] [talker]: Publishing: 'Hello World: 12'
[INFO] [1743339318.691418092] [listener]: I heard: [Hello World: 12]
```

```
victor@victor-VirtualBox: ~ 120x15
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$ ros2 component list
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'
victor@victor-VirtualBox:~$ ros2 component list
/ComponentManager
  1  /talker
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Listener
Loaded component 1 into '/ComponentManager' container node as '/listener'
victor@victor-VirtualBox:~$ ^C
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Listener
Loaded component 2 into '/ComponentManager' container node as '/listener'
victor@victor-VirtualBox:~$
```

- Listener 컴포넌트도 컨테이너에 적재
- 적재가 완료되면 컨테이너를 실행시켰던 터미널창에 로그 확인

ROS2 CLI

ROS2 CLI 실습

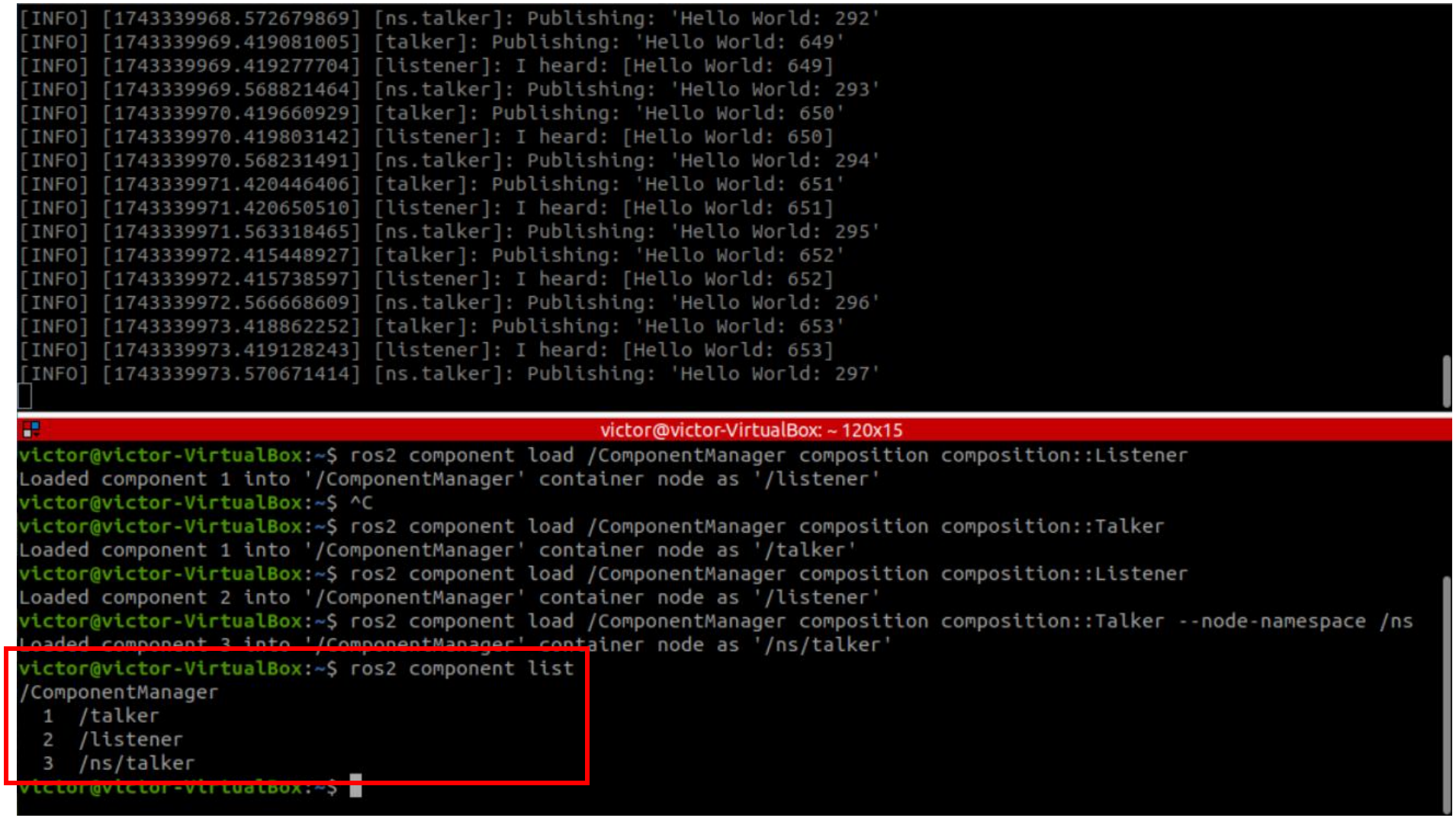
```
[INFO] [1743339694.594585627] [listener]: I heard: [Hello World: 379]
[INFO] [1743339695.411423653] [talker]: Publishing: 'Hello World: 380'
[INFO] [1743339695.411605305] [listener]: I heard: [Hello World: 380]
[INFO] [1743339695.559120754] [ns.talker]: Publishing: 'Hello World: 22'
[INFO] [1743339696.425814029] [talker]: Publishing: 'Hello World: 381'
[INFO] [1743339696.426038884] [listener]: I heard: [Hello World: 381]
[INFO] [1743339696.593665073] [ns.talker]: Publishing: 'Hello World: 23'
[INFO] [1743339697.779835331] [talker]: Publishing: 'Hello World: 382'
[INFO] [1743339697.780013164] [ns.talker]: Publishing: 'Hello World: 24'
[INFO] [1743339697.780144951] [listener]: I heard: [Hello World: 382]
[INFO] [1743339698.635144808] [talker]: Publishing: 'Hello World: 383'
[INFO] [1743339698.635326449] [ns.talker]: Publishing: 'Hello World: 25'
[INFO] [1743339698.635427918] [listener]: I heard: [Hello World: 383]
[INFO] [1743339699.419633977] [talker]: Publishing: 'Hello World: 384'
[INFO] [1743339699.419967580] [listener]: I heard: [Hello World: 384]
[INFO] [1743339699.577088143] [ns.talker]: Publishing: 'Hello World: 26'
```

- Talker 컴포넌트에 namespace를 붙여서 실행
- 라이브러리를 불러오지 않음
- 이미 talker 컴포넌트가 공유 라이브러리로 메모리에 적재되어 있어서
- 해당 메모리에 접근할 수 있고(Zero-copy)
- 네임스페이스 옵션만 변경하여 실행 시킴

```
victor@victor-VirtualBox: ~ 120x15
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'
victor@victor-VirtualBox:~$ ros2 component list
/ComponentManager
  1  /talker
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Listener
Loaded component 1 into '/ComponentManager' container node as '/listener'
victor@victor-VirtualBox:~$ ^C
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Listener
Loaded component 2 into '/ComponentManager' container node as '/listener'
victor@victor-VirtualBox:~$ ros2 component load /ComponentManager composition composition::Talker --node-namespace /ns
Loaded component 3 into '/ComponentManager' container node as '/ns/talker'
victor@victor-VirtualBox:~$
```

ROS2 CLI

ROS2 CLI 실습



ROS2 CLI

ROS2 CLI 실습

```
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$
victor@victor-VirtualBox:~$ ros2 launch composition composition_demo.launch.py
[INFO] [launch]: All log files can be found below /home/victor/.ros/log/2025-03-30-22-09-04-216288-victor-VirtualBox-5439
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [component_container-1]: process started with pid [5451]
[component_container-1] [INFO] [1743340144.558707664] [my_container]: Load Library: /opt/ros/humble/lib/libtalker_component.so
[component_container-1] [INFO] [1743340144.559265315] [my_container]: Found class: rclcpp_components::NodeFactoryTemplate<composition::Talke
r>
[component_container-1] [INFO] [1743340144.559280275] [my_container]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition:
:Talker>
[INFO] [launch_ros.actions.load_composable_nodes]: Loaded node '/talker' in container '/my_container'
[component_container-1] [INFO] [1743340144.568030131] [my_container]: Load Library: /opt/ros/humble/lib/liblistener_component.so
[component_container-1] [INFO] [1743340144.568413464] [my_container]: Found class: rclcpp_components::NodeFactoryTemplate<composition::Liste
ner>
[component_container-1] [INFO] [1743340144.568424123] [my_container]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition:
:Listener>
[INFO] [launch_ros.actions.load_composable_nodes]: Loaded node '/listener' in container '/my_container'
[component_container-1] [INFO] [1743340145.811137905] [talker]: Publishing: 'Hello World: 1'
[component_container-1] [INFO] [1743340145.812154488] [listener]: I heard: [Hello World: 1]
[component_container-1] [INFO] [1743340146.657925961] [talker]: Publishing: 'Hello World: 2'
[component_container-1] [INFO] [1743340146.658414017] [listener]: I heard: [Hello World: 2]
[component_container-1] [INFO] [1743340147.599856142] [talker]: Publishing: 'Hello World: 3'
[component_container-1] [INFO] [1743340147.600329917] [listener]: I heard: [Hello World: 3]
[component_container-1] [INFO] [1743340148.738544555] [talker]: Publishing: 'Hello World: 4'
[component_container-1] [INFO] [1743340148.738929013] [listener]: I heard: [Hello World: 4]
[component_container-1] [INFO] [1743340149.740613503] [talker]: Publishing: 'Hello World: 5'
[component_container-1] [INFO] [1743340149.744736082] [listener]: I heard: [Hello World: 5]
[component_container-1] [INFO] [1743340151.092563188] [talker]: Publishing: 'Hello World: 6'
[component_container-1] [INFO] [1743340151.093815215] [listener]: I heard: [Hello World: 6]
[component_container-1] [INFO] [1743340151.583109462] [talker]: Publishing: 'Hello World: 7'
[component_container-1] [INFO] [1743340151.593742263] [listener]: I heard: [Hello World: 7]
[component_container-1] [INFO] [1743340152.600559517] [talker]: Publishing: 'Hello World: 8'
[component_container-1] [INFO] [1743340152.601008870] [listener]: I heard: [Hello World: 8]
```

ROS2 CLI

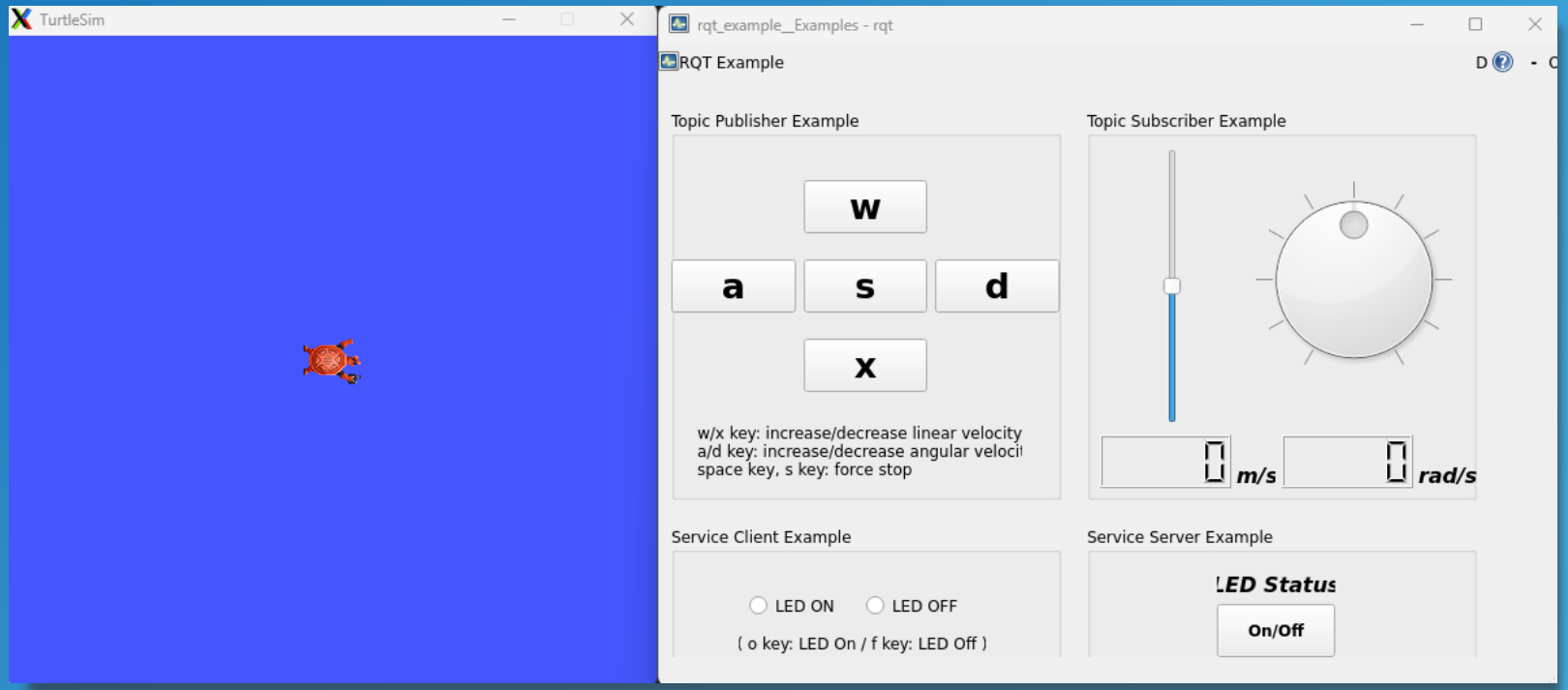
ROS2 CLI 실습

composition_demo.launch.py

opt > ros > humble > share > composition > launch > composition_demo.launch.py > ...

```python
import launch
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode


def generate_launch_description():
    """Generate launch description with multiple components."""
    container = ComposableNodeContainer(
            name='my_container',
            namespace='',
            package='rclcpp_components',
            executable='component_container',
            composable_node_descriptions=[
                ComposableNode(
                    package='composition',
                    plugin='composition::Talker',
                    name='talker'),
                ComposableNode(
                    package='composition',
                    plugin='composition::Listener',
                    name='listener')
            ],
            output='screen',
    )

    return launch.LaunchDescription([container])
```

RQt



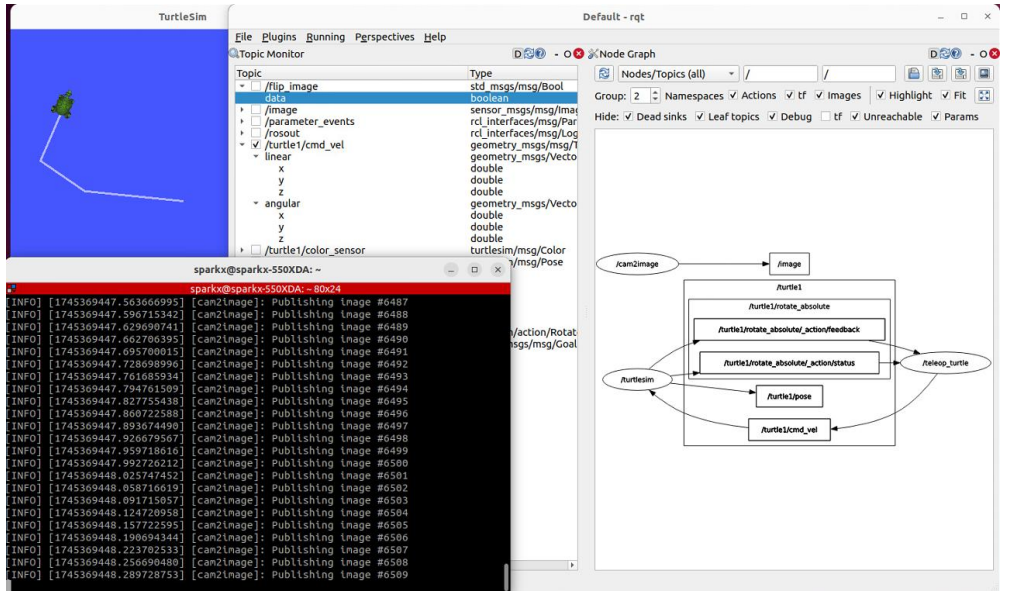
※ 필요한 파일 : rqt_example.zip

RQt 플러그인
RQt

RQt

RQt는 ROS2 시스템을 GUI로 “보고”, “조작하고”, “디버깅 ” 하는데 필수적인 도구 모음

- **RQt**
  - 플러그인 형태로 다양한 도구 및 인터페이스를 구현할 수 있는 ROS의 GUI(Graphical User Interface) 프레임워크
  - 토픽, 서비스, 액션 같은 ROS2 통신을 시각적으로 보고 조작(디버깅, 모니터링, 개발)
  - ROS + Qt의 합성어
  - 여러 Plugin을 통해 다양한 기능 제공
  - 크로스 플랫폼 지원



크로스플랫폼의 장점

- 운영 체제에 구애 받지 않고 개발 가능
- 한 번 개발하면 여러 OS에서 실행 가능 (코드 수정 최소화)
- ROS 2가 지원하는 다양한 환경에서 사용 가능

RQt 플러그인

RQt 플러그인 (RQt Plugin)

RQt 플러그인 스타일의 장점

- **표준화된 GUI 절차 제공**
  - GUI 시작 및 종료 처리 용이함
  - 다양한 옵션 저장 및 복원 가능
- **API 제공**
  - RQt 플러그인 API 사용 시 위 기능들을 비교적 쉽게 구현 가능

ros2_ws/rqt_example/package.xml

```
22    <export>
23      <build_type>ament_cmake</build_type>
24      <rqt_gui plugin="${prefix}/plugin.xml"/>
25    </export>
26  </package>
```

[활용]

- 노드/토픽 연결 상태를 rqt_graph로 점검
- 센서 데이터(LiDAR 거리값)을 rqt_plot을 실시간 확인
- 서비스 호출 실습시 rqt_service_caller 사용
- 디버깅시 rqt_consol로 에러 메시지 모니터링
- 파라미터 튜닝할 때 rqt_reconfigure로 실시간 수정

RQt 플러그인

RQt 플러그인 (RQt Plugin)

RQt 패키지

- 기본적으로 사용할 RQt 플러그인 ROS2 패키지는 다음과 같음

패키지 이름	설명
RQt 패키지	‘rqt_gui’, ‘rqt_gui_cpp’, ‘rqt_gui_py’, ‘rqt_py_common’ 패키지 포함
rqt_gui	여러 rqt 위젯을 단일 창에 도킹할 수 있는 위젯 패키지
rqt_gui_cpp	C++ 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
rqt_gui_py	Python 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
rqt_py_common	Python으로 작성된 RQt 플러그인에서 공용으로 사용되는 기능을 모듈로 제공하는 패키지
rqt_common_plugins	rqt_action, rqt_bag 등 20여개의 RQt 플러그인을 포함하는 메타패키지
qt_gui_core	qt_gui, qt_gui_cpp, qt_gui_py_common, qt_gui_app, qt_dotgraph 등을 담은 메타패키지
python_qt_binding	QtCore, QtGui, QtWidgets 등을 사용할 때 Python 언어 기반의 Qt API를 제공하는 바인딩 패키지

RQt 플러그인

RQt 플러그인 (RQt Plugin)

python_qt_binding

- **Qt Python API 사용**
  - Python으로 Qt API 사용 시 Qt C++ API 대신, Python으로 바인딩된 API 사용
  - 대표적 Qt Python API : PyQt, PySide

- **Python_qt_binding 패키지의 장점**
  - PyQt와 PySide를 구분 없이 사용 가능
  - 필요 시 두 바인딩 API 간 전환 가능

- PyQt와 PySide는 C++ 기반 Qt 라이브러리를 Python에서 사용할 수 있도록 바인딩한 것이다.
- 바인딩(binding)이란 Python과 C++ 사이에서 데이터를 변환하고 호출할 수 있도록 해주는 기술

- **RQt 플러그인 패키지 사용 순서**
  1. rqt_gui_py.plugin 모듈의 Plugin 클래스 상속
  2. qt_gui.plugin 모듈의 Plugin 클래스 상속
  3. python_qt_binding.QtCore 모듈의 Qobject 클래스 상속

RQt 개발 환경

- **RQt 플러그인 개발 환경**
  - Ubuntu 22.04 LTS, ROS2 Humble 기준
  - `ros-humble-desktop`을 설치하였다면, RQt 개발 환경은 설치 되어있음
  - 만약 설치가 되어있지 않다면, 다음 명령어를 통해 설치

```
$ sudo apt install qtcreator
```

RQt 플러그인 작성 순서
Python Style

RQt 플러그인 작성 순서

해당 섹션에서는 플러그인 작성 순서를 소개하고 있다. 제공된 코드를 받은 후 파일의 존재 유무만 확인해보자.

1. RQt 플러그인 패키지 생성

1.
```
$ cd ~/ros2_ws/src
```
2.
```
$ ros2 pkg create rqt_example –build-type ament_cmake –dependencies rclpy rqt_gui rtq_ggi_py python_qt_binding
```
3. 일반적인 패키지 생성과 다르지 않지만, RQt 플러그인의 기본 기능 관련 및 GUI 관련 패키지는 의존성 패키지로 포함
4. 특히, Python 언어로 작성하지만, RQt 플러그인의 일부로 작성하기 때문에, 빌드 형태는 `ament_cmake`로 설정

2. 패키지 설정 파일 수정

```
~/ros2_ws/src/rqt_example/package.xml
```

RQt 플러그인 작성 순서
Python Style

RQt 플러그인 작성 순서

3. 플러그인 파일 생성

~/ros2_ws/src/rqt_example/plugin.xml

4. 빌드 설정 파일 수정

~/ros2_ws/src/rqt_example/package.xml

5. 스크립트 폴더 및 파일 생성

~/ros2_ws/src/rqt_example/scripts/rqt_example

6. 리소스 폴더 및 UI 파일 생성

~/ros2_ws/src/rqt_example/resource/rqt_example.ui

7. 소스 폴더 및 파일 생성

~/ros2_ws/src/rqt_example/src/rqt_example/__init__.py

~/ros2_ws/src/rqt_example/src/rqt_example/examples.py

~/ros2_ws/src/rqt_example/src/rqt_example/examples_widget.py

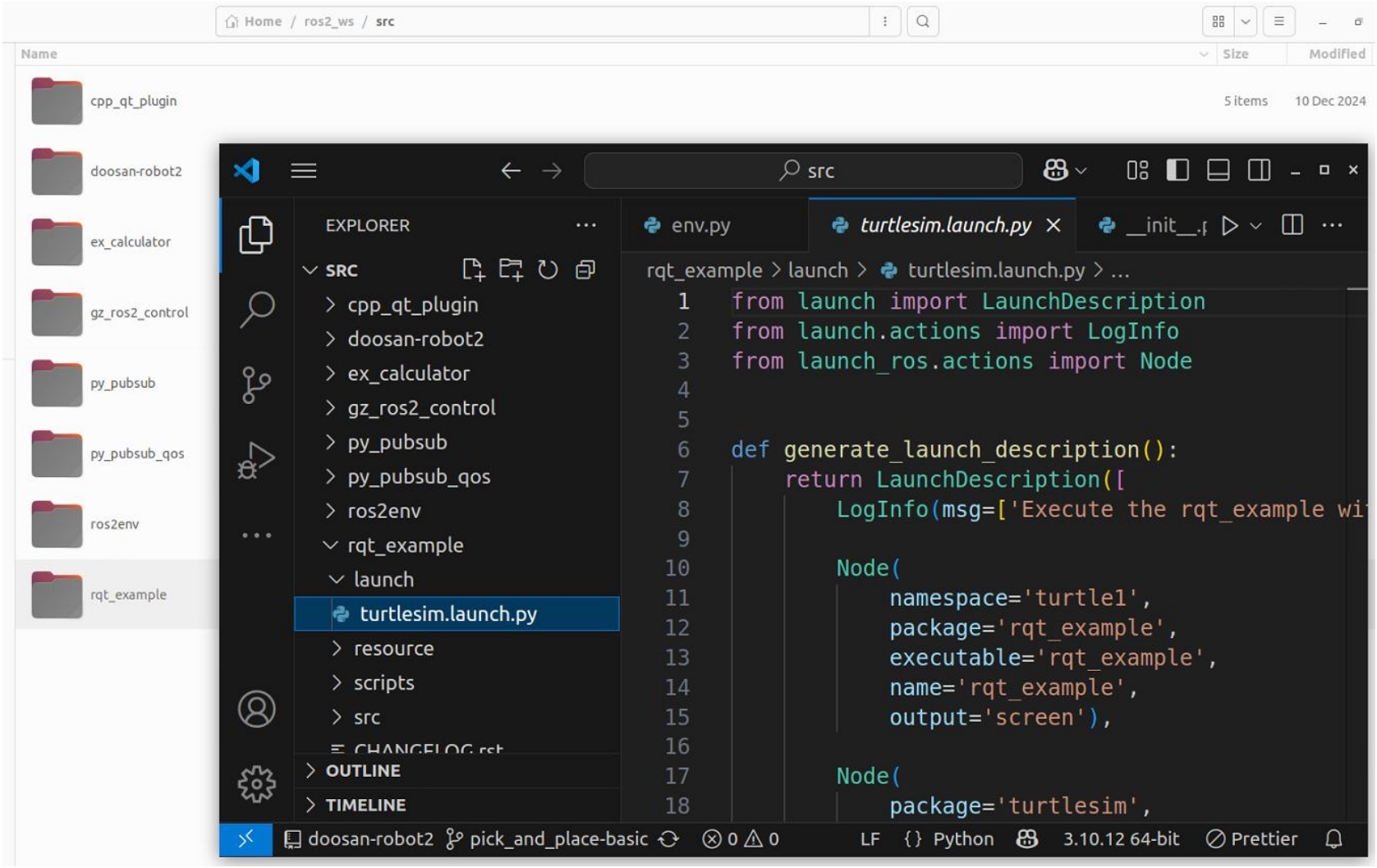
8. 런치 폴더 및 런치 파일 생성

~/ros2_ws/src/rqt_example/launch/rqt_plugin.launch.py

RQt 예제 구성

RQt example

rqt_example

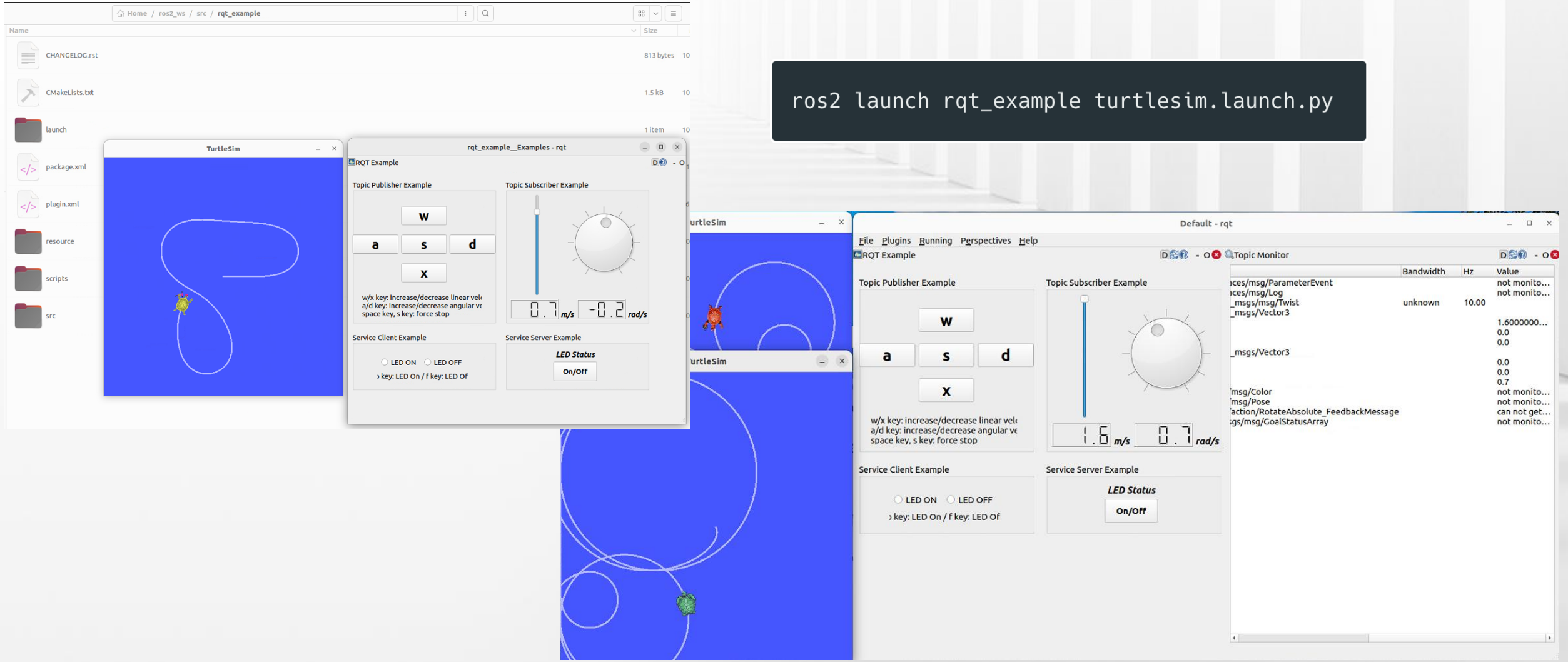


RQt 예제 구성

RQt example

rqt_example 실행 화면

```
ros2 launch rqt_example turtlesim.launch.py
```

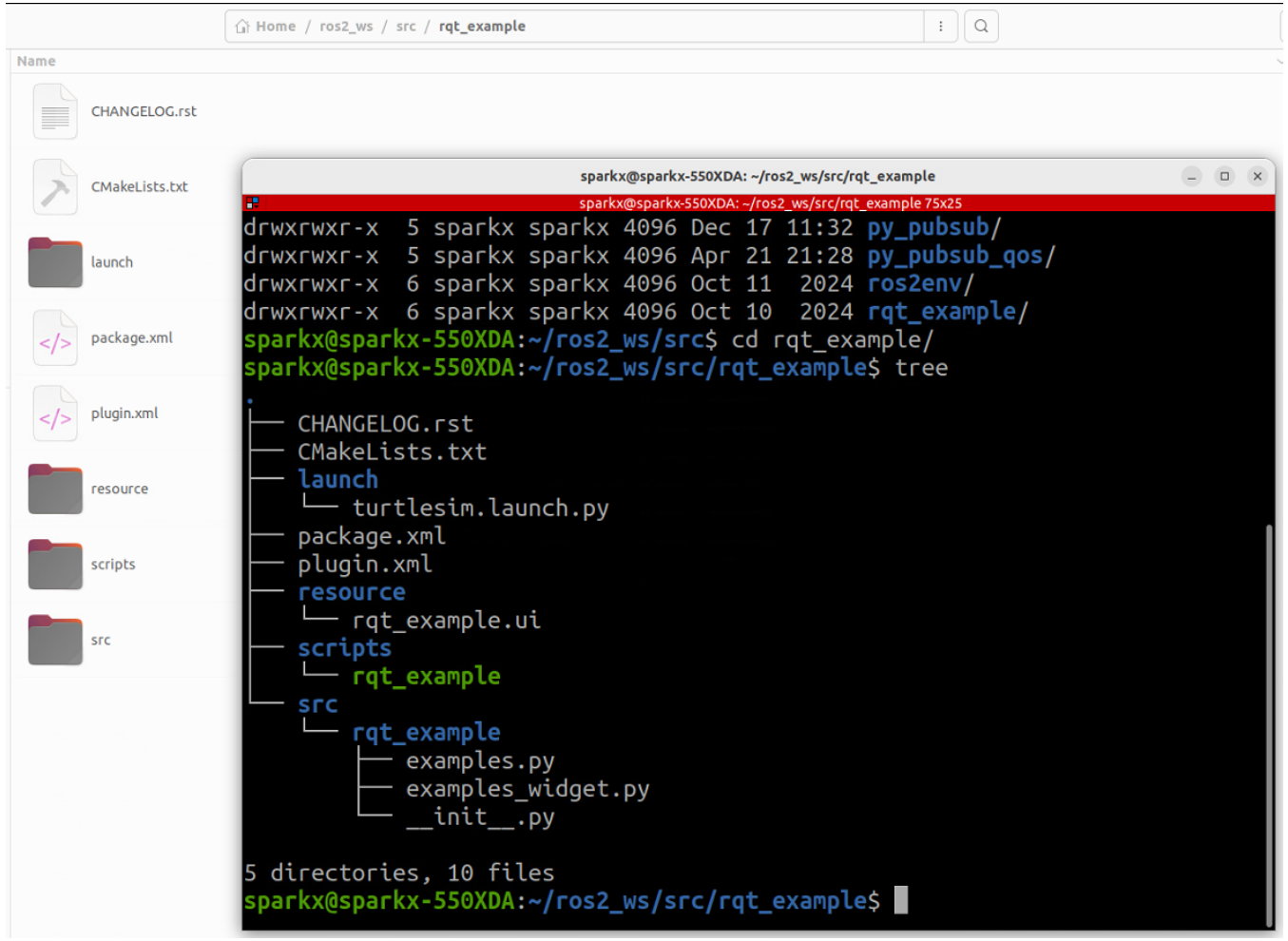


RQt 예제 구성

RQt example

rqt_example

- RQt 기본 GUI 위젯 사용
  - Push button, Radio button, Slider, Dial, LCD 숫자, Label
- ROS2 기반 빌드
  - ROS2의 토픽 Publisher와 Subscriber, 서비스 서버와 클라이언트를 함께 사용

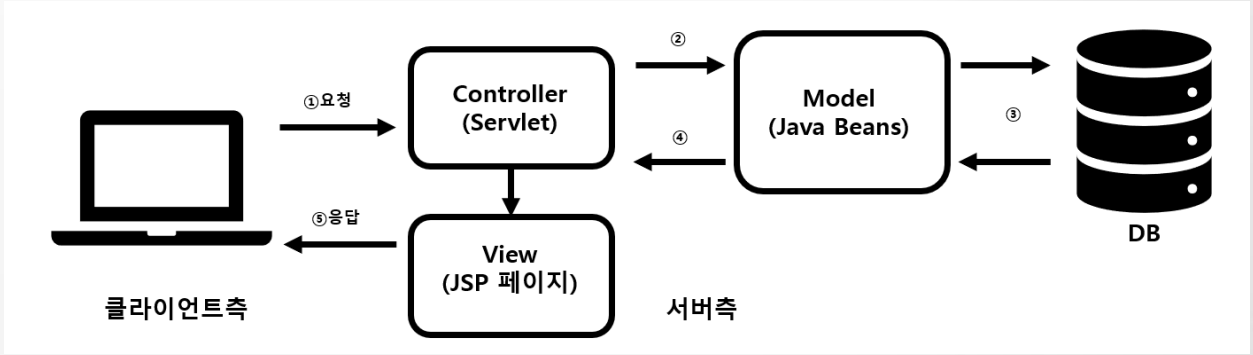


RQt 예제 구성

RQt example

파일 트리 구조

교재 : p547 GUI 개발 MVC(Model, View, Controller)



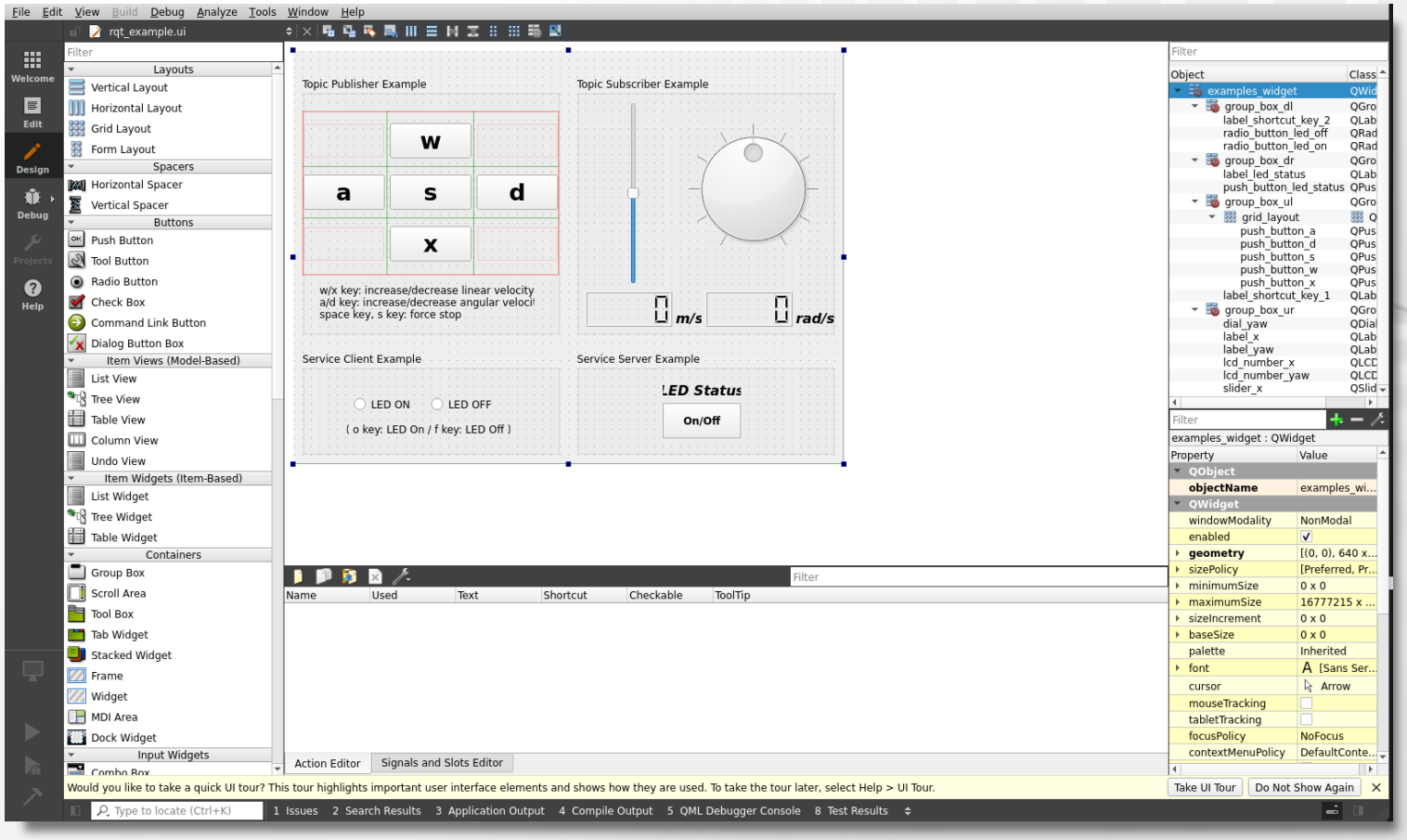

```
rqt_example
├── CHANGELOG.rst
├── CMakeLists.txt
├── launch
│   └── turtlesim.launch.py
├── package.xml
├── plugin.xml
├── resource
│   └── rqt_example.ui
├── scripts
│   └── rqt_example
└── src
    └── rqt_example
        ├── examples.py
        ├── examples_widget.py
        └── __init__.py
```

RQt 예제 구성

RQt 예제 UI 살펴보기

```
$ qtcreator ~/ros2_ws/src/rqt_example/resource/rqt_example.ui
```

- 또는 직접 Ubuntu 파일 탐색기에서 더블 클릭하여 실행



- 설치된 모든 플러그인을 강제로 다시 검색

```
victor@victor-VirtualBox:~/rqt_ws$ rqt --force-discover
```

```
rm ~/.config/ros.org/rqt_gui.ini
```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

패키지 설정 파일 수정

~/ros2_ws/src/rqt_example/package.xml

rqt_example > package.xml

```
1   <?xml version="1.0"?>
2   <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
3   <package format="3">
4     <name>rqt_example</name>
5     <version>0.6.0</version>
6     <description>ROS 2 example for RQt plugin</description>
7     <maintainer email="user@email.com">Juwan</maintainer>
8     <license>Apache 2.0</license>
9     <author email="user@email.com">Juwan</author>
10    <buildtool_depend>ament_cmake</buildtool_depend>
11    <exec_depend>geometry_msgs</exec_depend>
12    <exec_depend>python_qt_binding</exec_depend>
13    <exec_depend>python3-catkin-pkg-modules</exec_depend>
14    <exec_depend>qt_gui_py_common</exec_depend>
15    <exec_depend>rclpy</exec_depend>
16    <exec_depend>rqt_gui</exec_depend>
17    <exec_depend>rqt_gui_py</exec_depend>
18    <exec_depend>rqt_py_common</exec_depend>
19    <exec_depend>std_srvs</exec_depend>
20    <test_depend>ament_lint_auto</test_depend>
21    <test_depend>ament_lint_common</test_depend>
22    <export>
23      <build_type>ament_cmake</build_type>
24      <rqt_gui plugin="${prefix}/plugin.xml"/>
25    </export>
26  </package>
```

← RQt에 이 패키지에서 제공하려는 플러그인을 추가하는 기능

RQt 예제 구성

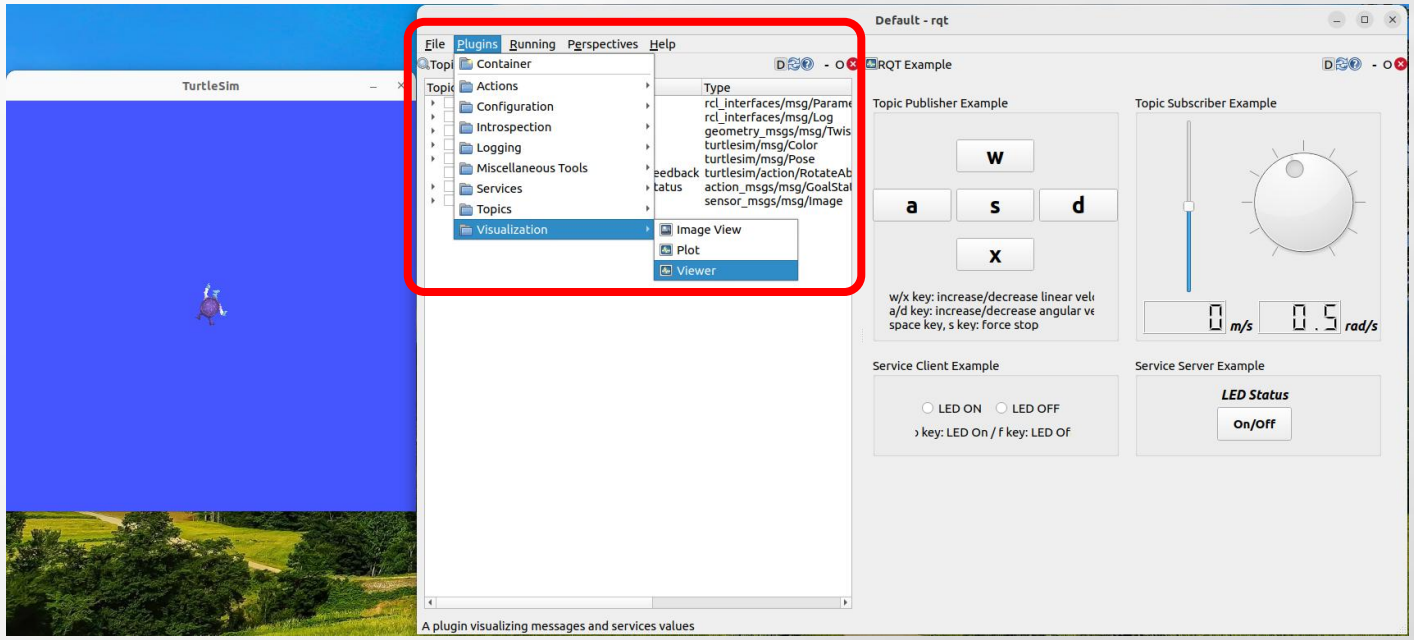
RQt 예제 설정 파일 살펴보기

RQt 플러그인 파일 생성

- 터미널 창에 `rqt`라고 입력하여, RQt를 실행
- 메뉴 옵션에서 Plugins 〉 Actions, Configuration, Introspection 등 세부 항목 실행 가능한 RQt 플러그인들 확인 가능
- 각 플러그인을 마우스 클릭만으로 실행할 수 있음

✓ rqt_example 패키지의 RQt 플러그인 또한 그림과 같이 Plugins 메뉴에 포함 및 실행 가능

✓ 이를 위해선 RQt 플러그인 파일 plugin.xml을 생성하고 알맞은 태그를 작성



RQt 예제 구성

RQt 예제 설정 파일 살펴보기

RQt 플러그인 파일 생성

~/ros2_ws/src/rqt_example/plugin.xml

- Group 태그가 메뉴의 세부 항목이 되며 〈label〉, 〈icon〉, 〈statustip〉이 해당 RQt 플러그인의 속성이 됨

rqt_example > plugin.xml

```
1  <library path="src">
2    <class name="Examples" type="rqt_example.examples.Examples" base_class_type="rqt_gui_py::Plugin">
3      <description>
4        A plugin visualizing messages and services values
5      </description>
6      <qtgui>
7        <group>
8          <label>Visualization</label>
9          <icon type="theme">folder</icon>
10         <statustip>Plugins related to visualization</statustip>
11       </group>
12       <label>Viewer</label>
13       <icon type="theme">utilities-system-monitor</icon>
14       <statustip>A plugin visualizing messages and services values</statustip>
15     </qtgui>
16   </class>
17 </library>
```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

빌드 설정 파일 수정

- 빌드설정파일 CMakeLists.txt 도 일반적인 ROS 패키지와 유사
- plugin.xml, resource, launch 폴더 및 하위 파일들을 share 폴더에 설치
- Scripts 폴더의 rqt_example 파일을 lib 폴더에 설치

~/ ros2_ws/src/rqt_example/CMakeLists.txt

rqt_example > CMakeLists.txt

```
1  ##############################################################################
2  # Set minimum required version of cmake, project name and compile options
3  ##############################################################################
4  cmake_minimum_required(VERSION 3.5)
5  project(rqt_example)
6
7  ##############################################################################
8  # Find and load build settings from external packages
9  ##############################################################################
10 find_package(ament_cmake REQUIRED)
11 find_package(ament_cmake_python REQUIRED)
12
13 ##############################################################################
14 # Install
15 ##############################################################################
16 ament_python_install_package(${PROJECT_NAME}
17   PACKAGE_DIR src/${PROJECT_NAME})
18
19 install(FILES
20   plugin.xml
21   DESTINATION share/${PROJECT_NAME}
22 )
23
24 install(DIRECTORY
25   resource
26   launch
27   DESTINATION share/${PROJECT_NAME}
28 )
29
30 install(PROGRAMS
31   scripts/rqt_example
32   DESTINATION lib/${PROJECT_NAME}
33 )
34
35 ##############################################################################
36 # Test
37 ##############################################################################
38 if(BUILD_TESTING)
39   find_package(ament_lint_auto REQUIRED)
40   ament_lint_auto_find_test_dependencies()
41 endif()
42
43 ##############################################################################
44 # Macro for ament package
45 ##############################################################################
46 ament_package()
```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

스크립트 폴더 및 파일 생성

- 스크립트 폴더에는 RQt 플러그인을 지정하고 종료하는 코드를 기술
- RQt의 진입코드라고 볼 수 있으며, rqt_gui 중 main module의 Main 클래스를 이용하여 RQt 플러그인 기능 사용
- 메인 코드인 examples 모듈의 Examples 클래스 호출

```
~/ros2_ws/src/rqt_example/scripts/rqt_example
```

rqt_example > scripts > rqt_example

```python
import sys

from rqt_gui.main import Main

from rqt_example.examples import Examples

plugin = 'rqt_example.examples.Examples'
main = Main(filename=plugin)
sys.exit(main.main(standalone=plugin))

```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

리소스 폴더 및 UI 파일 생성

- Qt의 ui 파일은 XML 태그를 이용
- 수작업으로 작업하지는 않고 qtcreator에서 손쉽게 구성할 수 있다.

```
~/ros2_ws /src/rqt_example/resource/rqt_example.ui
```

rqt_example > resource > rqt_example.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>examples_widget</class>
 <widget class="QWidget" name="examples_widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>640</width>
    <height>480</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>RQT Example</string>
  </property>
  <widget class="QGroupBox" name="group_box_ur">
   <property name="geometry">
    <rect>
     <x>330</x>
     <y>30</y>
     <width>300</width>
     <height>300</height>
    </rect>
   </property>
   <property name="title">
    <string>Topic Subscriber Example</string>
   </property>
   <widget class="QSlider" name="slider_x">
    <property name="geometry">
     <rect>
      <x>50</x>
      <y>30</y>
      <width>31</width>
      <height>211</height>
     </rect>
    </property>
    ... skipped below
```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

소스 폴더 및 파일 생성

- rqt_example의 메인 소스 코드에 해당하는 파일들
- 다음 섹션에서 상세한 설명

```
~/ros2_ws/src/rqt_example/__init__.py
```

```
~/ros2_ws/src/rqt_example/examples.py
```

```
~/ros2_ws/src/rqt_example/examples_widget.py
```

RQt 예제 구성

RQt 예제 설정 파일 살펴보기

런치 폴더 및 런치 파일 생성

- 런치 파일은 turtlesim 패키지의 turtlesim_node 노드와 함께 연동하여 테스트 가능하도록 구성
- turtlesim_node 노드의 토픽과 맞추기 위해 namespace를 `turtle1`으로 설정

~/ ros2_ws /src/rqt_example/launch/turtlesim.launch.py

rqt_example > launch > turtlesim.launch.py

```python
from launch import LaunchDescription
from launch.actions import LogInfo
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        LogInfo(msg=['Execute the rqt_example with turtlesim node.']),

        Node(
            namespace='turtle1',
            package='rqt_example',
            executable='rqt_example',
            name='rqt_example',
            output='screen'),

        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim',
            output='screen')
    ])

```

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

RQt 메인 소스 코드

```
~/ros2_ws/src/rqt_example/examples.py
```

```
~/ros2_ws/src/rqt_example/examples_widget.py
```

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples.py

rqt_example > src > rqt_example > examples.py

```python
from rqt_example.examples_widget import ExamplesWidget

from rqt_gui_py.plugin import Plugin


class Examples(Plugin):

    def __init__(self, context):
        super(Examples, self).__init__(context)
        self.setObjectName('RQt example')
        self.widget = ExamplesWidget(context.node)
        serial_number = context.serial_number()
        if serial_number > 1:
            self.widget.setWindowTitle(self.widget.windowTitle() + ' ({0})'.format(serial_number))
        context.add_widget(self.widget)

    def shutdown_plugin(self):
        print('Shutdown the RQt example.')
        self.widget.shutdown_widget()

```

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples.py

- Examples 클래스는 `rqt_gui_py.plugin` 모듈의 Plugin 클래스를 상속
  - ROS RQt 플러그인 기본 기능 제공
  - 플러그인 관리, 초기화 및 종료와 같은 기능 처리
- def __init__(self, context): super(Examples, self).__init__(context)
  - 부모 클래스(Plugin)의 생성자를 호출하여 초기화
  - Context는 RQt에서 제공하는 실행 컨텍스트로 RQt본체와 함께 도킹되어 사용될 수 있게 해주고 플러그인의 환경 정보를 포함한다.
- Self.setObjectName('RQt example')
  - 플러그인의 객체 이름을 'RQt example'로 설정한다.
  - 이 이름은 RQt에서 플러그인을 관리할 때 사용한다.
- ExamplesWidget 클래스는 작성하고자 하는 UI를 포함한 실제 코드가 담긴 클래스
  - ExamplesWidget 객체를 생성하여 widget에 저장.
  - 이 노드가 ExamplesWidget 클래스 내에서 rclpy의 Node 역할을 하는 것(실제 GUI 요소를 포함하는 위젯 클래스)
  - Context.node 를 전달하여 ROS2노드와 연결

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples.py

- Serial_number = context.serial_number()
  - Context.serial_number()를 호출하여 플러그인의 시리얼 번호를 가져온다.
  - 만약 시리얼 번호가 1보다 크면, 창 제목, windowTitle()에 시리얼 번호를 추가한다.
  - 이 기능은 동일한 플러그인을 여러 개 실행할 때, 각 플러그인 창을 구별할 수 있도록 한다. 예, RQt example, RQt example(2).....
- Context.add_widget(self.widget)
  - Context.add_widget(self.widget)을 호출하여 ExamplesWidget을 RQt 인터페이스에 추가한다.
  - 이렇게 해야 GUI가 RQt창에 표시된다.
- Def shutdown_plugin(self):
  - RQt 플러그인이 종료될때 실행된다.
  - 메서드를 호출하여 위젯의 종료 처리를 수행한다.

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- ExamplesWidget 클래스는 앞서 설명한 GUI 화면 구성을 담당하는 rqt_example.ui 파일을 호출 및 화면에 띄우는 역할
- 다음과 같은 내용들을 포함
- Topic publisher, topic subscriber, service server, service client, timer, push button, radio button 등

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 34 ~ 36

```
34          _, package_path = get_resource('packages', pkg_name)
35          ui_file = os.path.join(package_path, 'share', pkg_name, 'resource', ui_filename)
36          loadUi(ui_file, self)
```

- ament_index_python.resources 모듈의 get_resource 함수를 이용하여, `rqt_example` 패키지의 `rqt_example.ui` 파일을 loadUi 함수로 불러옴(패키지 경로를 가져옴)
- 이를 통해 qtcreator로 미리 만들어둔 UI를 화면에 띄울 수 있는 것(.ui파일을 로드해서 Widget에 적용)

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 48 ~ 52

```
48          qos = QoSProfile(depth=10)
49          self.publisher = self.node.create_publisher(Twist, topic_name, qos)
50          self.subscriber = self.node.create_subscription(Twist, topic_name, self.get_velocity, qos)
51          self.service_server = self.node.create_service(SetBool, service_name, self.set_led_status)
52          self.service_client = self.node.create_client(SetBool, service_name)
```

- 예제에서 사용할 ros 요소들을 선언
- 키보드의 키 w, a, s, d, x, space bar 또는 각 버튼을 클릭하여,
  로봇의 병진 속도(linear) 및 회전 속도(angular)를 변경할 수 있도록 Publishing($ros2 interface show geometry_msgs/msg/Twist)
- Subscriber는 이 속도 값을 수신 받아 slider와 dial과 같은 인디케이터로 표현하거나 LCD 숫자 형태로 값을 표시
- 서비스의 경우, radio button 두개가 있는데, 이들 중 하나를 선택하면 해당 값을 서비스 request 값으로 보냄
- 서비스 response값으로 가상의 LED가 켜지고 꺼짐을 나타낼 수 있는 True, False를 반한

RQt 예제 소스 코드 분석
RQt 메인 소스 코드

examples_widget.py

- Line : 54 ~ 60

```
54          self.publish_timer = QTimer(self)
55          self.publish_timer.timeout.connect(self.send_velocity)
56          self.publish_timer.start(self.PUBLISH_INTERVAL)
57
58          self.update_timer = QTimer(self)
59          self.update_timer.timeout.connect(self.update_indicators)
60          self.update_timer.start(self.REDRAW_INTERVAL)
```

- 해당 코드는 특정 콜백 함수를 정기적으로 실행할 timer류에 대한 선언
- send_velocity : w, a, s, d, x, space bar 키 또는 버튼 클릭에 의해 변경된 속도 값 publishing 함수(100ms마다 호출)
- publish_timer는 send_velocity 함수를 콜백 함수로 설정함으로써 주기적으로 퍼블리시 함
- update_indicators : Subscribing한 속도 값을 처리하는 함수
- update_timer는 update_indicators 함수를 콜백 함수로 설정, 30ms마다 GUI 갱신

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 146 ~ 154

```
146     def send_velocity(self):
147         twist = Twist()
148         twist.linear.x = self.pub_velocity.linear.x
149         twist.linear.y = 0.0
150         twist.linear.z = 0.0
151         twist.angular.x = 0.0
152         twist.angular.y = 0.0
153         twist.angular.z = self.pub_velocity.angular.z
154         self.publisher.publish(twist)
```

- send_velocity 함수는 이와 같이 geometry_msgs 패키지의 Twist 인터페이스를 사용
- 지정된 병진 속도와 회전 속도를 각각 linear.x 와 angular.z로 지정하여 Publishing하는 역할
- 토픽 Publishing 주기는 100ms(0.1sec)

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 156 ~ 161

```
156     def update_indicators(self):
157
158         self.slider_x.setValue(int(self.sub_velocity.linear.x * self.CMD_VEL_X_FACTOR))
159         self.dial_yaw.setValue(int(self.sub_velocity.angular.z * self.CMD_VEL_YAW_FACTOR))
160         self.lcd_number_x.display(self.sub_velocity.linear.x)
161         self.lcd_number_yaw.display(self.sub_velocity.angular.z)
```

- update_indicators 함수는 이와 같이 slider 형태, dial 형태, LCD number 형태의 위젯으로 구성
- 각 위젯의 값으로는 서브스크라이브한 병진(linear) 속도와 회전(angular) 속도를 사용
- 해당 함수의 토픽 값을 GUI 형태로 볼 수 있게 됨

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 85 ~ 86

```
85      def get_velocity(self, msg):
86          self.sub_velocity = msg
```

- update_indicators 함수에서 사용된 병진 속도 및 회전 속도는 이와 같이 get_velocity 함수가 토픽을 수신할 때마다 업데이트
- 이 함수에서 사용된 인터페이스는 토픽 퍼블리셔와 마찬가지로 Twist 인터페이스

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 63 ~ 77

```
63          self.push_button_w.pressed.connect(self.increase_linear_x)
64          self.push_button_x.pressed.connect(self.decrease_linear_x)
65          self.push_button_a.pressed.connect(self.increase_angular_z)
66          self.push_button_d.pressed.connect(self.decrease_angular_z)
67          self.push_button_s.pressed.connect(self.set_stop)
68
69          self.push_button_w.setShortcut('w')
70          self.push_button_x.setShortcut('x')
71          self.push_button_a.setShortcut('a')
72          self.push_button_d.setShortcut('d')
73          self.push_button_s.setShortcut('s')
74
75          self.shortcut_space = QShortcut(QKeySequence(Qt.Key_Space), self)
76          self.shortcut_space.setContext(Qt.ApplicationShortcut)
77          self.shortcut_space.activated.connect(self.push_button_s.pressed)
```

- push_button으로 지정된 w, a, s, d, x 버튼을 마우스를 눌렀을 때 호출되는 함수들을 지정
- 또한, 해당하는 키보드 자판을 눌렀을 때 동일 효과를 주기 위한 short cut 설정

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 103 ~ 117

```
103     def increase_linear_x(self):
104         self.pub_velocity.linear.x += 0.1
105 
106     def decrease_linear_x(self):
107         self.pub_velocity.linear.x -= 0.1
108 
109     def increase_angular_z(self):
110         self.pub_velocity.angular.z += 0.1
111 
112     def decrease_angular_z(self):
113         self.pub_velocity.angular.z -= 0.1
114 
115     def set_stop(self):
116         self.pub_velocity.linear.x = 0.0
117         self.pub_velocity.angular.z = 0.0
```

- 각 push_button을 눌렀을 때 실행되는 함수는 이와 같으며, 현재의 병진 속도와 회전 속도를 변화시킴
- 단위는 SI 단위로, 병진 속도에 m/sec, 회전 속도에는 rad/sec

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 79 ~ 83

```
79          self.radio_button_led_on.clicked.connect(self.call_led_service)
80          self.radio_button_led_off.clicked.connect(self.call_led_service)
81
82          self.radio_button_led_on.setShortcut('o')
83          self.radio_button_led_off.setShortcut('f')
```

- LED ON, LED OFF 버튼을 눌렀을 때 call_led_service라는 서비스 클라이언트의 요청 함수를 지정한 구문
- push_button과 비슷하게 LED ON은 키보드의 `o` 자판, OFF는 `f` 자판을 숏컷으로 설정

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 119 ~ 144

```
119     def call_led_service(self):
120         request = SetBool.Request()
121 
122         if self.radio_button_led_on.isChecked():
123             request.data = True
124         elif self.radio_button_led_off.isChecked():
125             request.data = False
126 
127         wait_count = 1
128         while not self.service_client.wait_for_service(timeout_sec=0.5):
129             if wait_count > 5:
130                 return
131             self.node.get_logger().error('Service not available #{0}'.format(wait_count))
132             wait_count += 1
133 
134         future = self.service_client.call_async(request)
135 
136         while rclpy.ok():
137             if future.done():
138                 if future.result() is not None:
139                     response = future.result()
140                     self.node.get_logger().info(
141                         'Result of service call: {0}'.format(response.message))
142                 else:
143                     self.node.get_logger().error('Error calling service')
144                 break
```

- call_led_service 함수는 radio button의 클릭 상태를 보고 request 값으로 True 또는 False를 지정하여 요청

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 88 ~ 101

```
88      def set_led_status(self, request, response):
89          if request.data:
90              self.push_button_led_status.setText('ON')
91              self.push_button_led_status.setStyleSheet('color: rgb(255, 170, 0);')
92              response.success = True
93              response.message = 'LED ON'
94          elif not request.data:
95              self.push_button_led_status.setText('OFF')
96              self.push_button_led_status.setStyleSheet('')
97              response.success = True
98              response.message = 'LED OFF'
99          else:
100             response.success = False
101         return response
```

- 서비스 클라이언트의 요청을 받아 처리하는 함수
- SetBool 서비스 인터페이스를 사용하며 request.data 값에 따라 버튼의 상태를 변경한 후 클라이언트에게 반환
- 버튼 상태와 색상 변경이 성공적으로 완료되면, 그 결과를 `success`와 `message` 변수에 담아 서비스 클라이언트에 반환

RQt 예제 소스 코드 분석

RQt 메인 소스 코드

examples_widget.py

- Line : 163 ~ 169

```
163     def shutdown_widget(self):
164         self.update_timer.stop()
165         self.publish_timer.stop()
166         self.node.destroy_client(self.service_client)
167         self.node.destroy_service(self.service_server)
168         self.node.destroy_subscription(self.subscriber)
169         self.node.destroy_publisher(self.publisher)
```

- examples.py의 shutdown_plugin 함수가 호출하는 함수
- rqt_example 노드를 실행한 터미널 창에서 ctrl + c (SIGINT) 신호 또는 UI화면에서의 창 닫기 버튼을 눌러 종료할 때 호출됨

RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

패키지 빌드

- cd ~/rqt_example
- colcon build
- source install/setup.bash

RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

패키지 빌드

- 아래 명령어를 통해 기본적인 rqt_example이 실행되는 지 확인

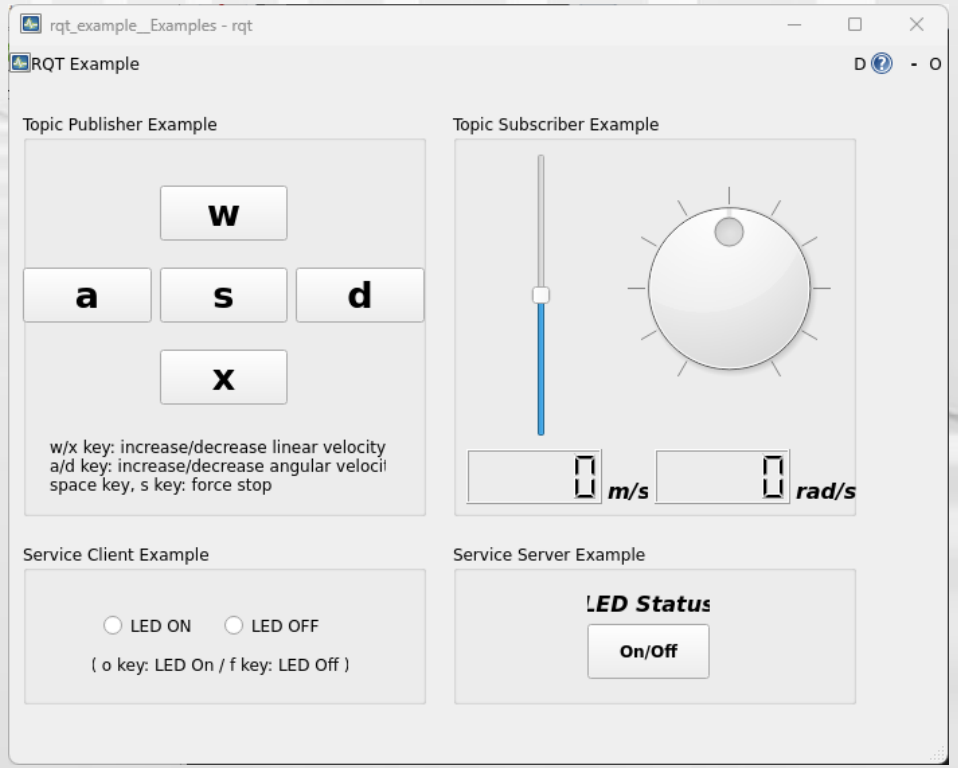
```
ros2 run rqt_example rqt_example
```

- 만약 `qt_gui_main() found no plugin matching~`과 같은 에러 발생 시

→ $ `rqt --force-discover` 명령어 실행

또는

→ `rm ~/.config/ros.org/rqt_gui.ini` 명령어로 설정 파일 삭제

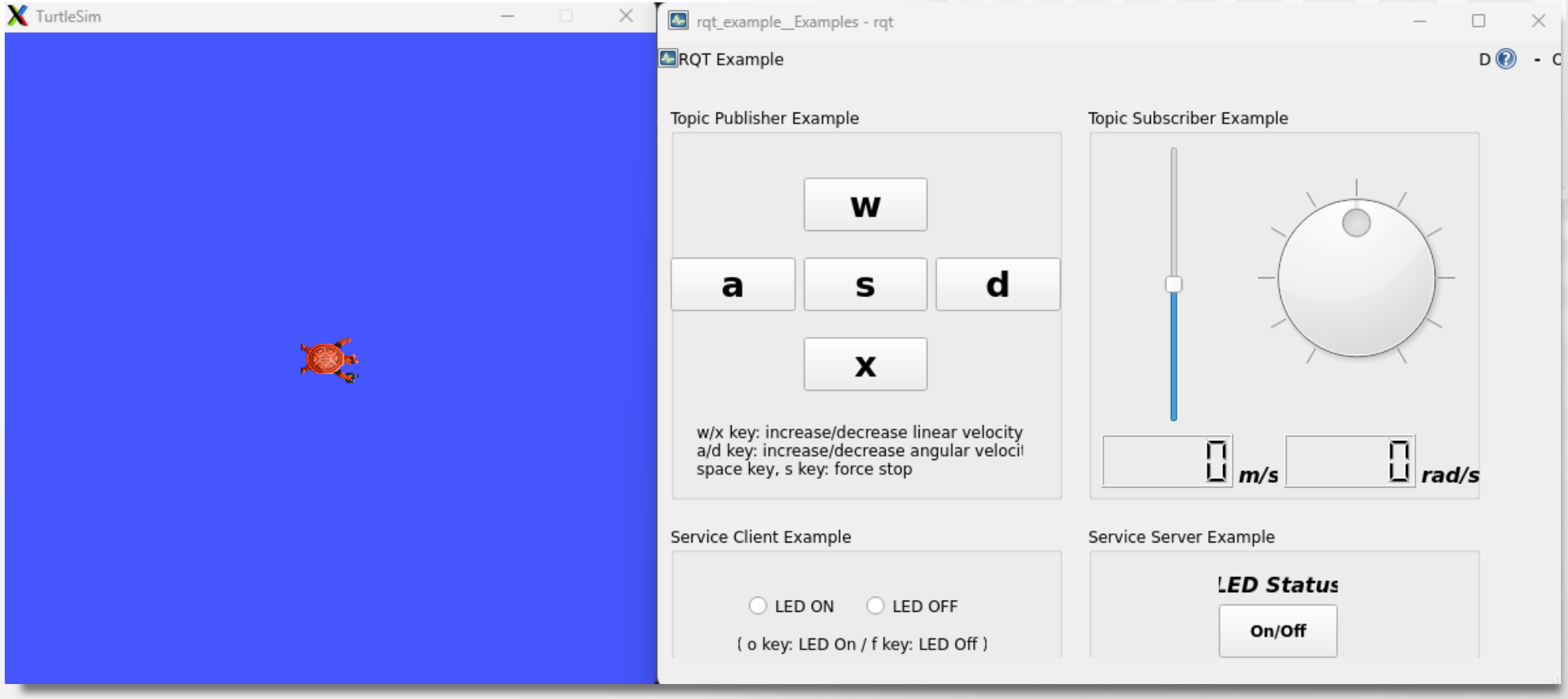


RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

런치 파일 실행

```
ros2 launch rqt_example turtlesim.launch.py
```



RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

런치 파일 실행

```
ros2 launch rqt_example turtlesim.launch.py
```

turtlesim.launch.py

src > rqt_example > launch > turtlesim.launch.py > ...

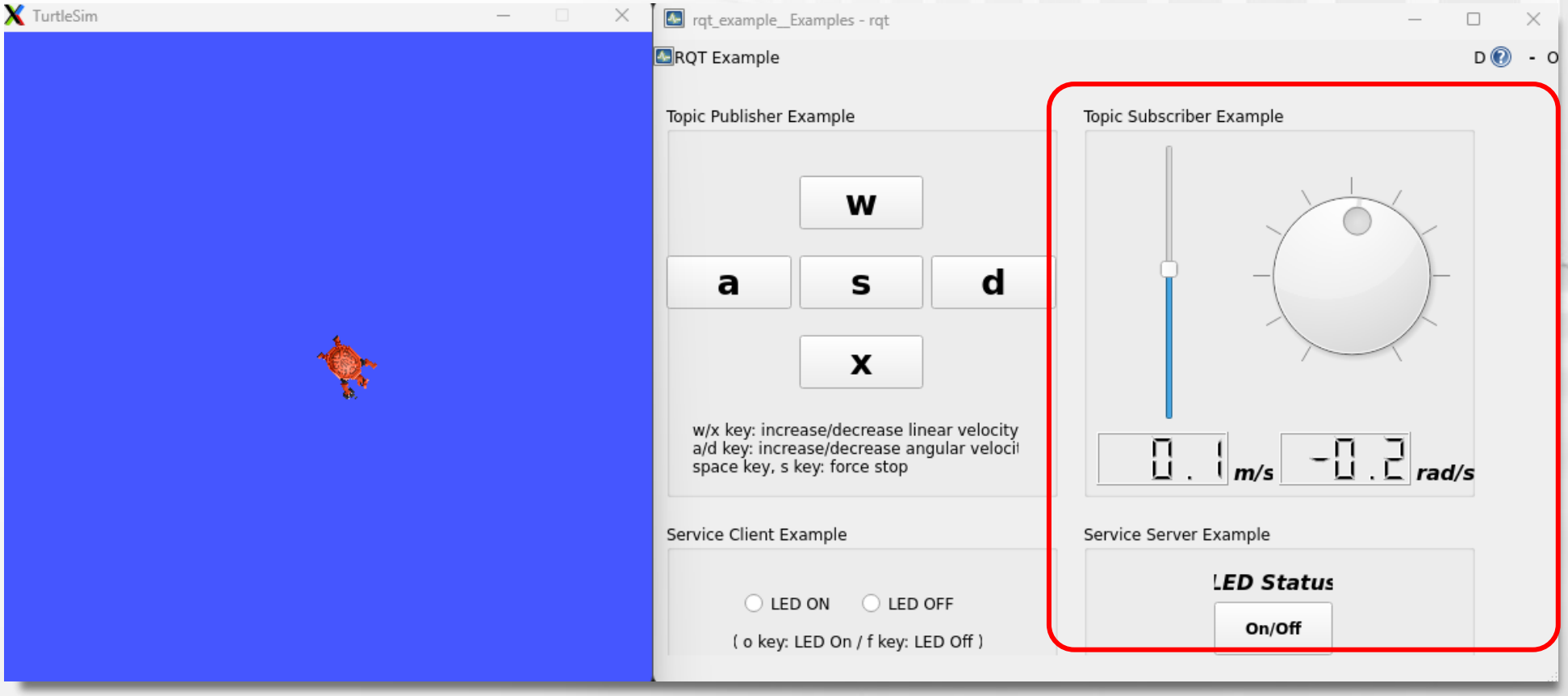
```python
from launch import LaunchDescription
from launch.actions import LogInfo
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        LogInfo(msg=['Execute the rqt_example with turtlesim node.']),

        Node(
            namespace='turtle1',
            package='rqt_example',
            executable='rqt_example',
            name='rqt_example',
            output='screen'),

        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim',
            output='screen')
    ])

```

RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

런치 파일 실행

- 키보드 조작 후 값이 변경되는 지 확인

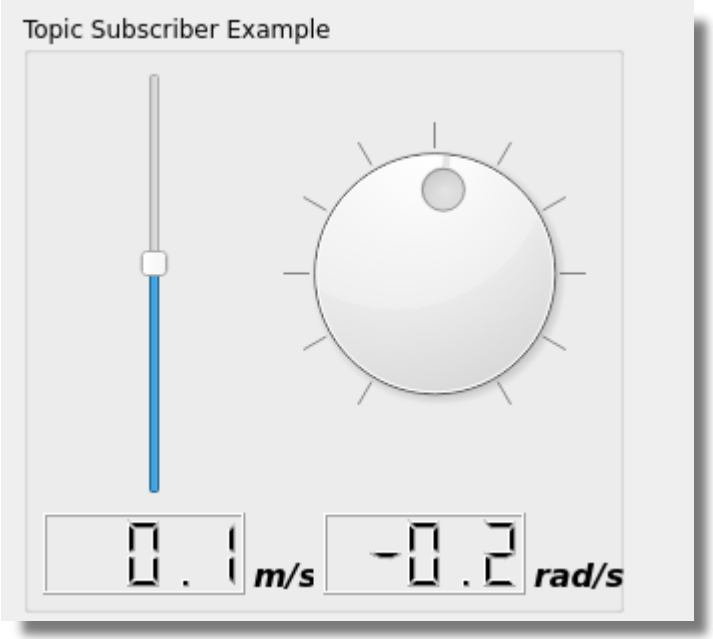


RQt 플러그인 예제 실행

Turtlesim Node 연동 예제

런치 파일 실행

- 새로운 터미널을 열고, source install/setup.bash 후,
- ros2 topic echo /turtle1/cmd_vel 을 통해, 토픽에 발행되는 값 확인




```
linear:
  x: 0.1
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.2
---
```

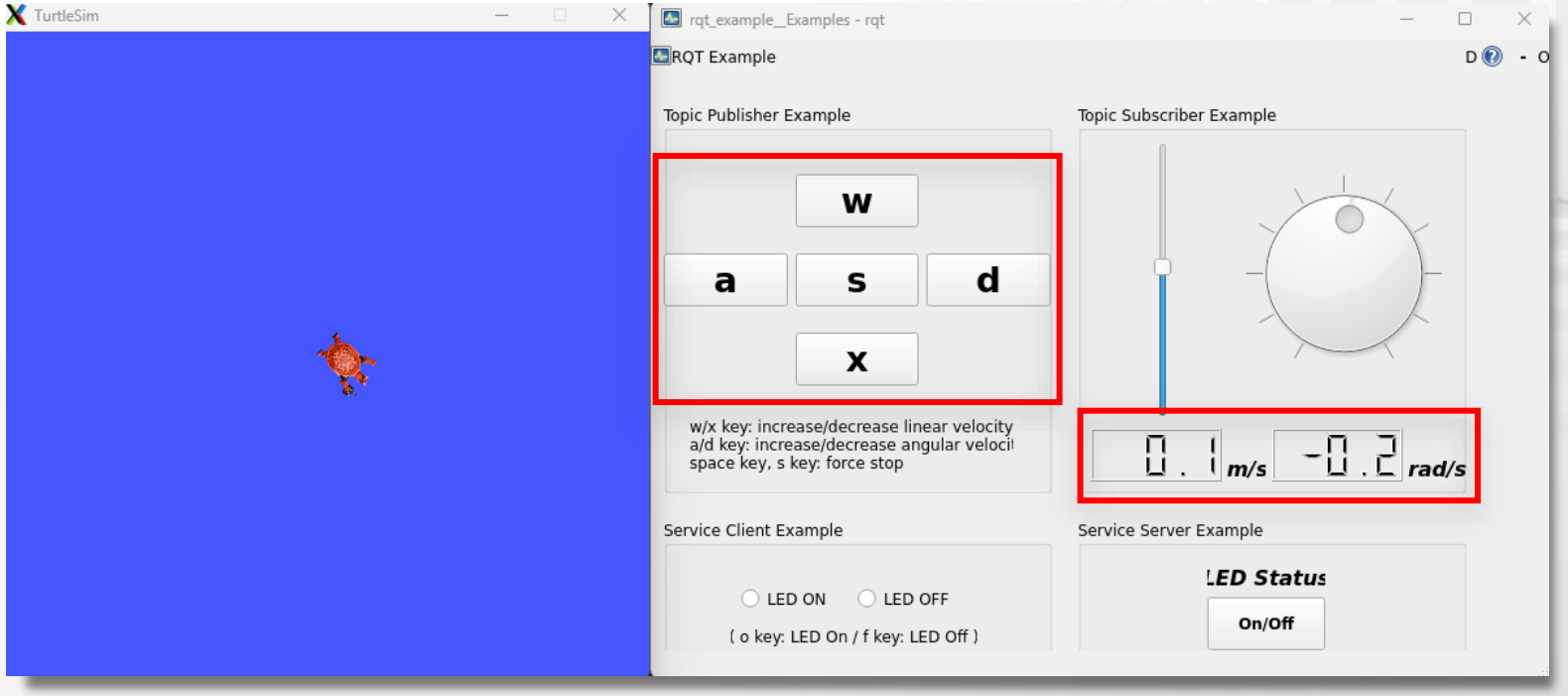
RQt 플러그인 예제 실행

Turtlesim Node 연동 실습

GitHub https://github.com/ros-visualization

코드 수정해서 Build 후 실행해보기

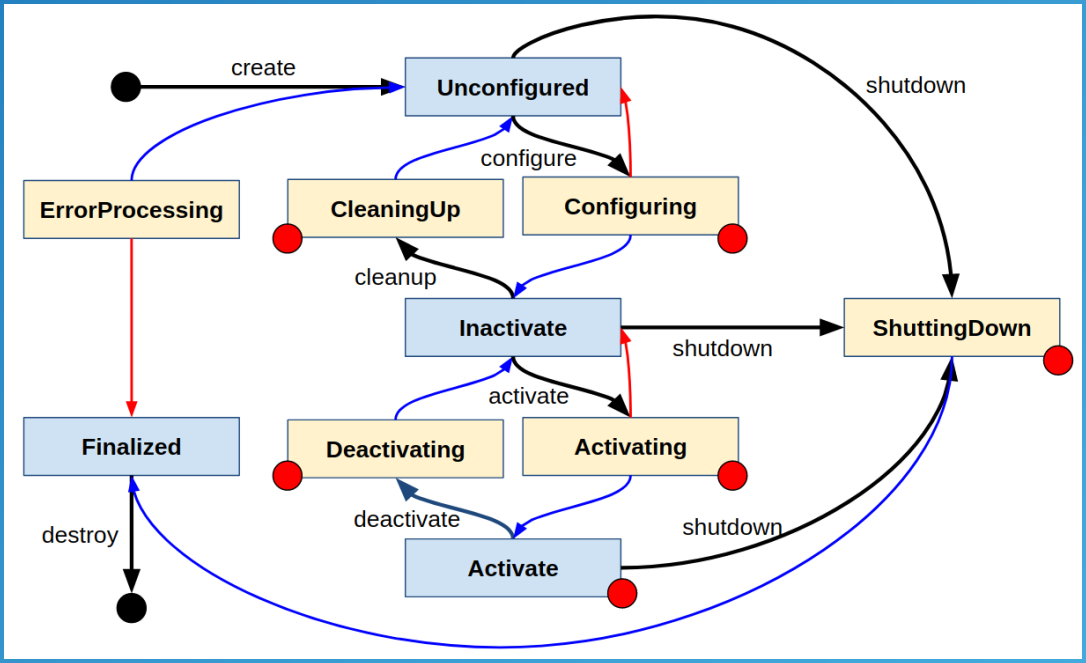
- 키보드, 병진속도, 회전속도 증가/감소 단위 값 수정해서 Build 해보기




```
Finished <<< dsr_controller2 [1min 24s]

Summary: 34 packages finished [3min 35s]
  3 packages had stderr output: dsr_controller2 dsr_hardware2 realtime_control
sparkx@sparkx-550XDA:~/ros2_ws$ . install/setup.bash
sparkx@sparkx-550XDA:~/ros2_ws$ rqt
RosPluginProvider._parse_plugin_xml() plugin file "/home/sparkx/ros2_ws/install/rqt_example/share/rqt_example/plugin.xml" in package "rqt_example" not found
RosPluginProvider._parse_plugin_xml() plugin file "/home/sparkx/ros2_ws/install/rqt_example/share/rqt_example/plugin.xml" in package "rqt_example" not found
```

```
sparkx@sparkx-550XDA:~/ros2_ws$ ros2 run rqt_example rqt_example --force-discover
Shutdown the RQt example.
```

Lifecycle


create
Unconfigured
shutdown
configure
ErrorProcessing
CleaningUp
Configuring
cleanup
Inactivate
shutdown
ShuttingDown
activate
Finalized
Deactivating
Activating
deactivate
shutdown
destroy
Activate

Lifecycle

Lifecycle

- **Lifecycle**
  - ROS2에서는 노드의 상태 관리를 위해 Lifecycle 인터페이스 제공
  - 노드는 주요 상태(Unconfigured, Inactive, Active, Finalized)와 전환 상태(Configuring, CleaningUp 등)를 가짐
  - 노드를 체계적으로 관리 및 상태 전환을 통해 노드를 구성, 활성화, 비활성화, 정리 가능

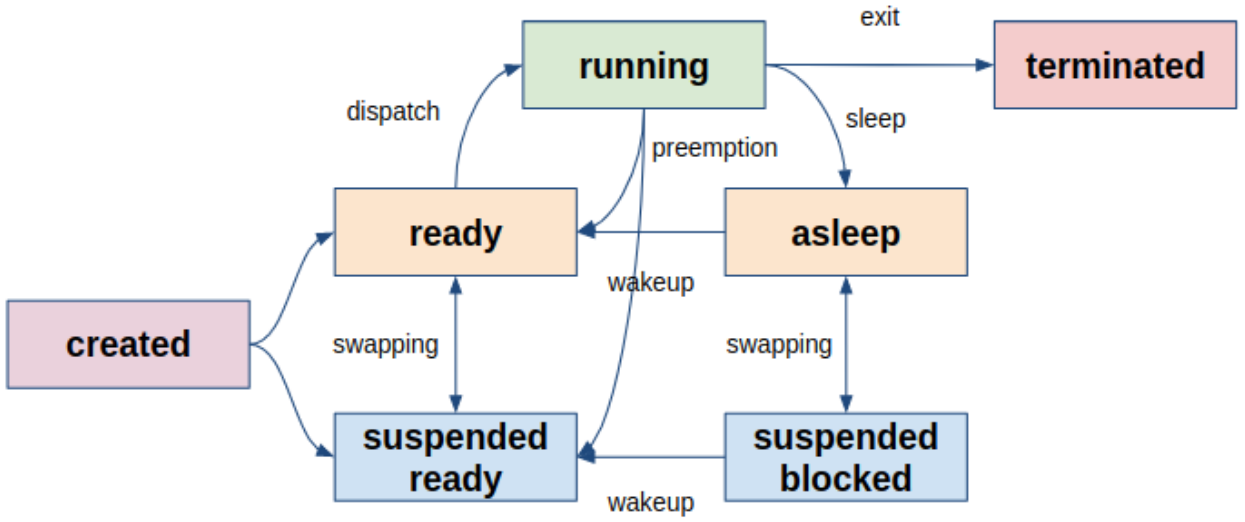
[사용 목적]

- **노드가 준비 중인지**
- **아직 데이터 수집 안한 상태인지**
- **안전하게 작동 가능한 상태인지**
- **위와 같이 노드의 상태 변화를 명확하게 관리**

Lifecycle

Lifecycle

- OS는 복수개의 프로세스를 효율적으로 관리하기 위해 프로세스의 상태를 정의하고, 상태의 전환을 조율함
- 프로세스의 상태는 프로세서, 메모리와 같은 자원의 할당 여부에 따라 정의됨
- 프로세스의 상태는 처리 순서, 교착 상태, 메모리 할당 등에 의해 전환될 수 있음



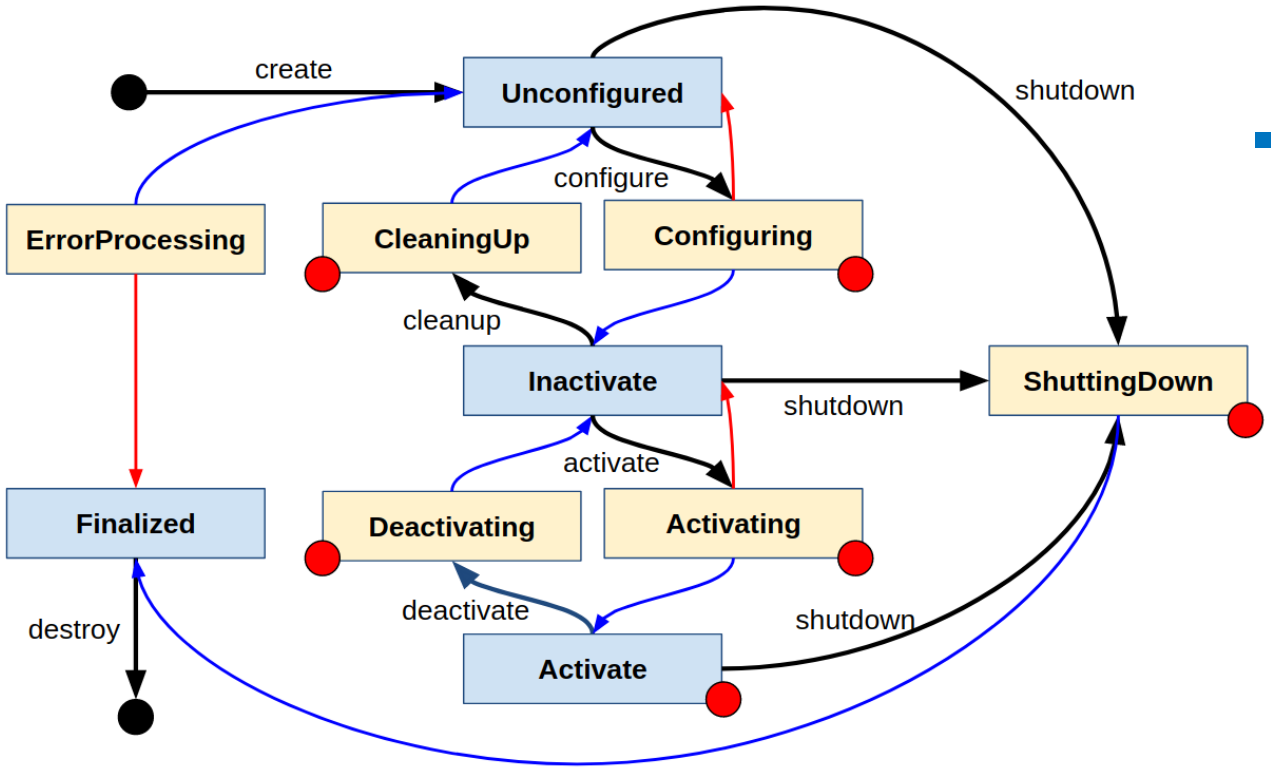
Lifecycle

Lifecycle

- **ROS2에서는 Lifecycle 인터페이스를 통해 노드의 상태 확인이나 재실행, 교체가 가능**
- **예시: 카메라 센서를 통해 받은 이미지 정보를 발간하는 노드**
  - 먼저 노드를 동작시키기 전에 카메라와의 통신을 위한 포트가 제대로 잡혔는지 확인
  - 만약 노드가 동작되는 도중에 에러가 발생 하였다면 잠시 그 동작을 멈추고 에러를 해결한 다음 재시작
  - 주변 환경의 변화로 인해 에러를 해결할 수 없다면 해당 노드는 종료시키고 준비된 다른 노드를 동작

Lifecycle

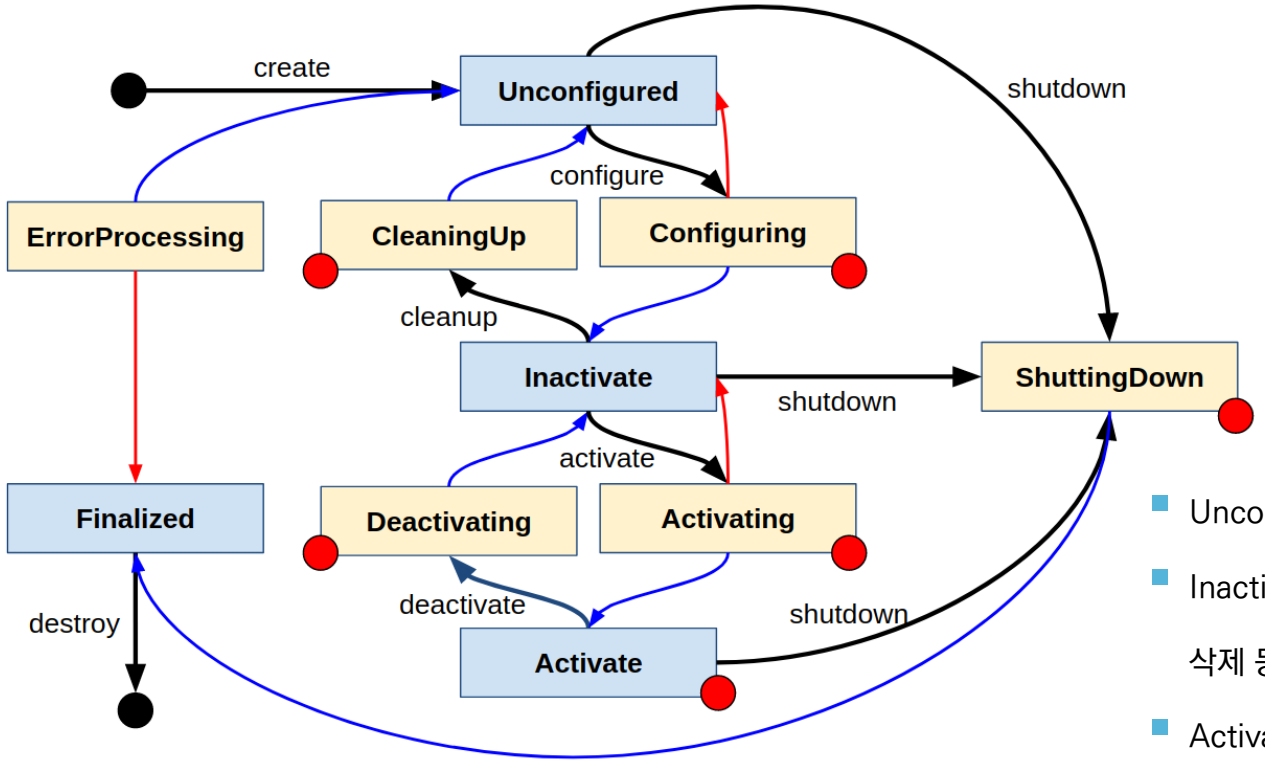
Lifecycle



- 노드의 상태와 상태 전환(Transition)
  - 파란 박스: 주요 상태
  - 노란 박스: 전환 상태
  - 검정색 화살표: 전환을 나타냄
  - 파란색 화살표: 전환 성공 시 주요 상태의 변화
  - 빨간색 화살표: 전환 실패 시 주요 상태의 변화
  - 빨간색 작은 원: 에러가 발생할 수 있는 상태

Lifecycle

Lifecycle

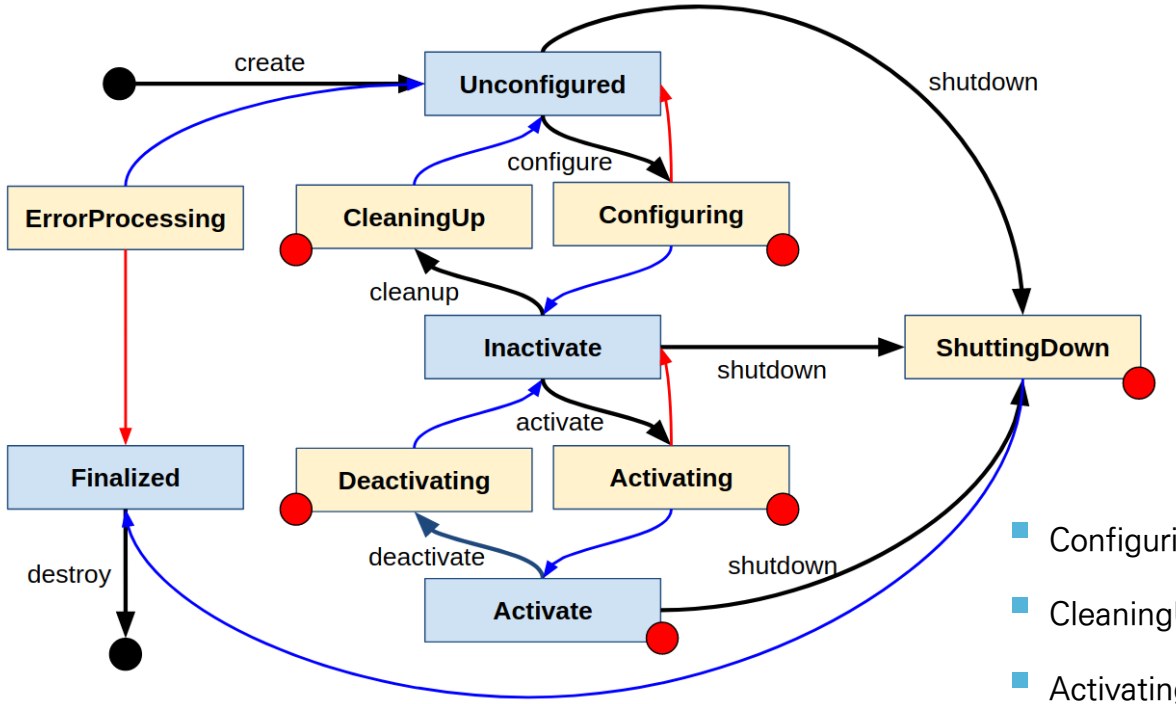


주요 상태

- Unconfigured: 노드가 생성된 직후의 상태, 에러 발생 이후 다시 조정될 수 있는 상태
- Inactivate: 노드가 동작을 수행하지 않는 상태. 파라미터 등록, 토픽 발간과 구독 추가 삭제 등을 (재)구성 할 수 있는 상태
- Activate: 노드가 동작을 수행하는 상태.
- Finalized: 노드가 메모리에서 해제되기 직전 상태 노드가 파괴되기 전 디버깅이나 내부 검사를 진행할 수 있는 상태

Lifecycle

Lifecycle

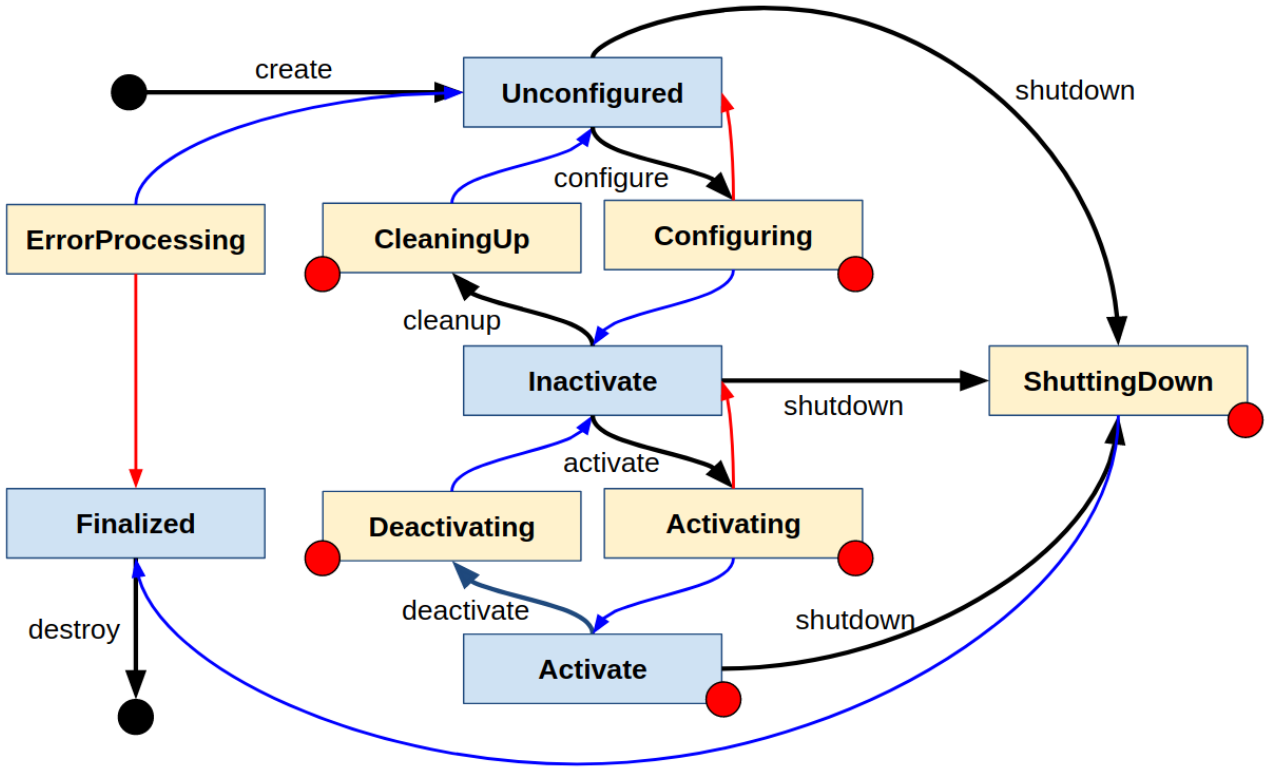


전환 상태

- Configuring: 노드를 구성하기 위해 필요한 설정 수행
- CleaningUp: 노드가 처음 생성되었을 때 상태와 동일하게 만드는 과정 수행
- Activating: 노드가 동작을 수행하기 전 마지막 준비 과정 수행
- Deactivating : 노드가 동작을 수행하기 전으로 돌아가는 과정 수행
- ShuttingDown: 노드가 파괴되기 전 필요한 과정 수행
- ErrorProcessing: 사용자 코드가 동작되는 상태에서 발생하는 에러를 해결하기 위한 과정 수행

Lifecycle

Lifecycle



- **전환**
  - Create: 노드를 생성하고 초기 상태로 설정
  - Configure: 노드를 구성하여 준비 상태로 만듦
  - Cleanup: 노드를 초기화하여 이전 상태로 되돌림
  - Activate: 노드를 활성화하여 기능을 수행할 수 있게 함
  - Deactivate: 노드를 비활성화하여 동작을 멈춤
  - Shutdown: 노드를 안전하게 종료하는 과정
  - Destroy: 노드를 메모리에서 완전히 제거

Lifecycle

Lifecycle

- 다음 명령어로 노드들을 실행 (각기 다른 터미널에서 실행)

```
$ ros2 run lifecycle lifecycle_talker
```

```
$ ros2 run lifecycle lifecycle_listener
```

```
$ ros2 run lifecycle lifecycle_service_client
```

Lifecycle

Lifecycle

- lc_client를 실행시킬 시 lc_talker의 상태가 configure → Inactive → Activate → Inactive
  Active → Inactivate → Finalized의 순서로 전환되는 것을 확인 가능

```
[INFO] [1734957585.584383029] [lc_talker]: on_configure() is called.
[INFO] [1734957586.584679885] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN] [1734957586.584834264] [LifecyclePublisher]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is not activated
[INFO] [1734957587.584579847] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
[INFO] [1734957588.584537043] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
...
[INFO] [1734957595.584612628] [lc_talker]: on_activate() is called.
[INFO] [1734957597.585062244] [lc_talker]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #11]
[INFO] [1734957598.584647883] [lc_talker]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #12]
...
[INFO] [1734957605.585311565] [lc_talker]: on_deactivate() is called.
[INFO] [1734957606.584640621] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN] [1734957606.584859103] [LifecyclePublisher]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is not activated
[INFO] [1734957607.584641216] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
...
[INFO] [1734957615.585283846] [lc_talker]: on_activate() is called.
[INFO] [1734957617.585802035] [lc_talker]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #30]
...
[INFO] [1734957625.585313310] [lc_talker]: on_deactivate() is called.
[INFO] [1734957626.584621985] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN] [1734957626.584858428] [LifecyclePublisher]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is not activated
[INFO] [1734957627.584537851] [lc_talker]: Lifecycle publisher is currently inactive. Messages are not published.
..
[INFO] [1734957635.586532316] [lc_talker]: on cleanup is called.
[INFO] [1734957645.584812617] [lc_talker]: on shutdown is called from state unconfigured
```

ros2 security

- Security는 SROS의 유틸리티로, DDS-Security를 ROS2에서 사용하기 위해 필요한 도구를 모아둔 것
  - 보안키 저장소 생성

```
$ mkdir ~/sros2_demo
$ cd ~/sros2_demo
$ ros2 security create_keystore demo_keystore
```

← ~/sros2_demo/demo_keystore/enclaves 디렉토리 확인해보기

  - 보안키 생성

```
$ ros2 security create_enclave demo_keystore /talker_listener/talker
$ ros2 security create_enclave demo_keystore /talker_listener/listener
```

  - 환경변수 구성

```
$ export ROS_SECURITY_KEYSTORE=~/sros2_demo/demo_keystore
$ export ROS_SECURITY_ENABLE=true
$ export ROS_SECURITY_STRATEGY=Enforce
```

← Terminator 2개 실행해서 진행 : menu → broadast all

- **환경변수 3가지**
  1. ROS_SECURITY_KEYSTORE : 보안설정 파일을 보관하는 폴더를 지정. demo_keystore
  2. ROS_SECURITY_ENABLE : 보안 설정의 On/Off 기능으로 true/false 형태로 설정. Default는 false
  3. ROS_SECURITY_STRATEGY : 보안 설정 방법. Enforce로 설정하면 보안 설정 파일이 없는 메시지 통신은 금지, Permissive의 경우 비보안 참여자로 참석시킴.

※ 위 환경변수 3가지는 노드를 실행할 때마다 매번 각 터미널에서 선언해야 함.
ROS2 보안 기능을 지속적으로 사용할 예정이라면 ~/.bashrc에 추가해야 함

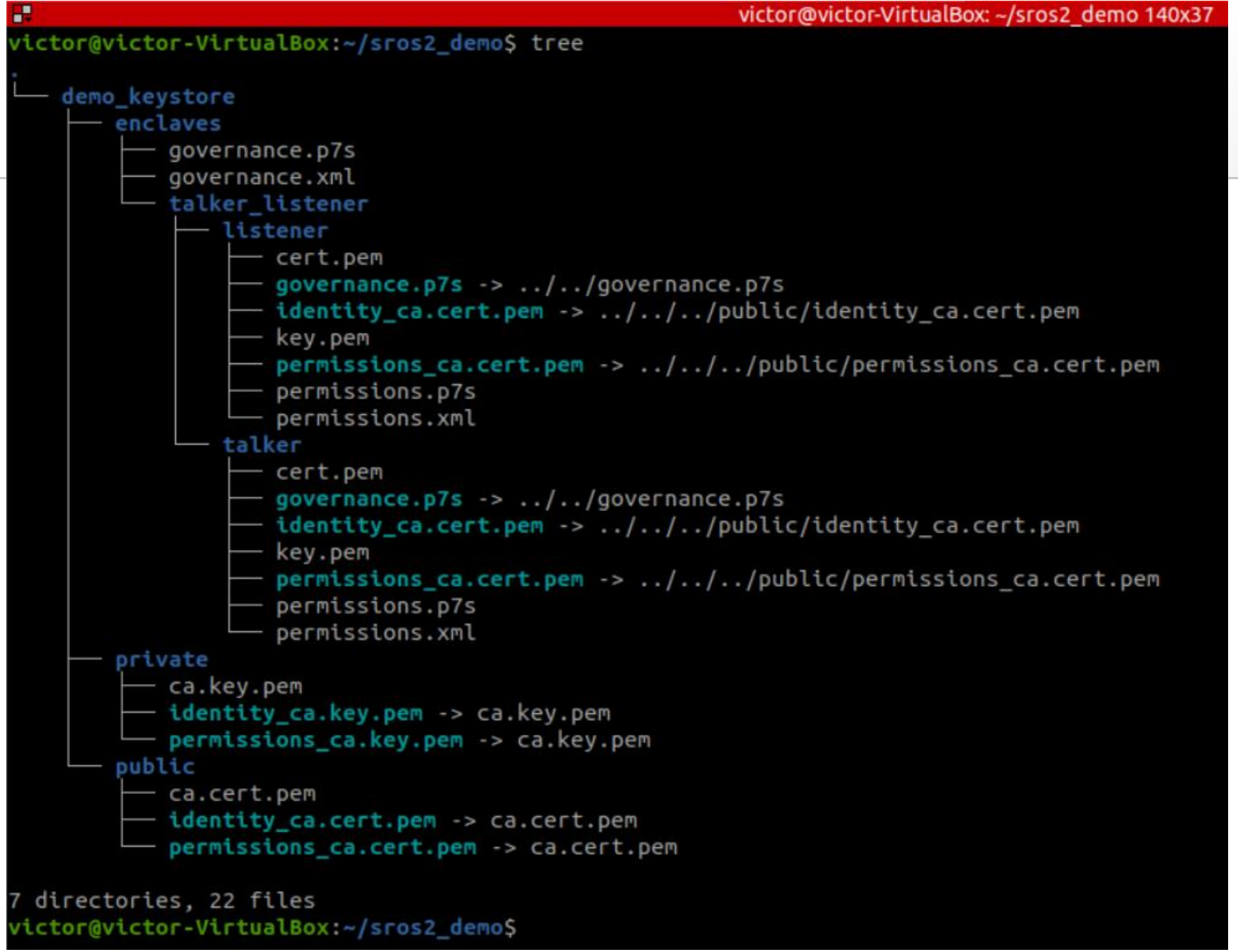
- **Security는 SROS의 유틸리티로, DDS-Security를 ROS2에서 사용하기 위해 필요한 도구를 모아둔 것**
  - talker node 실행하여 데모를 시작

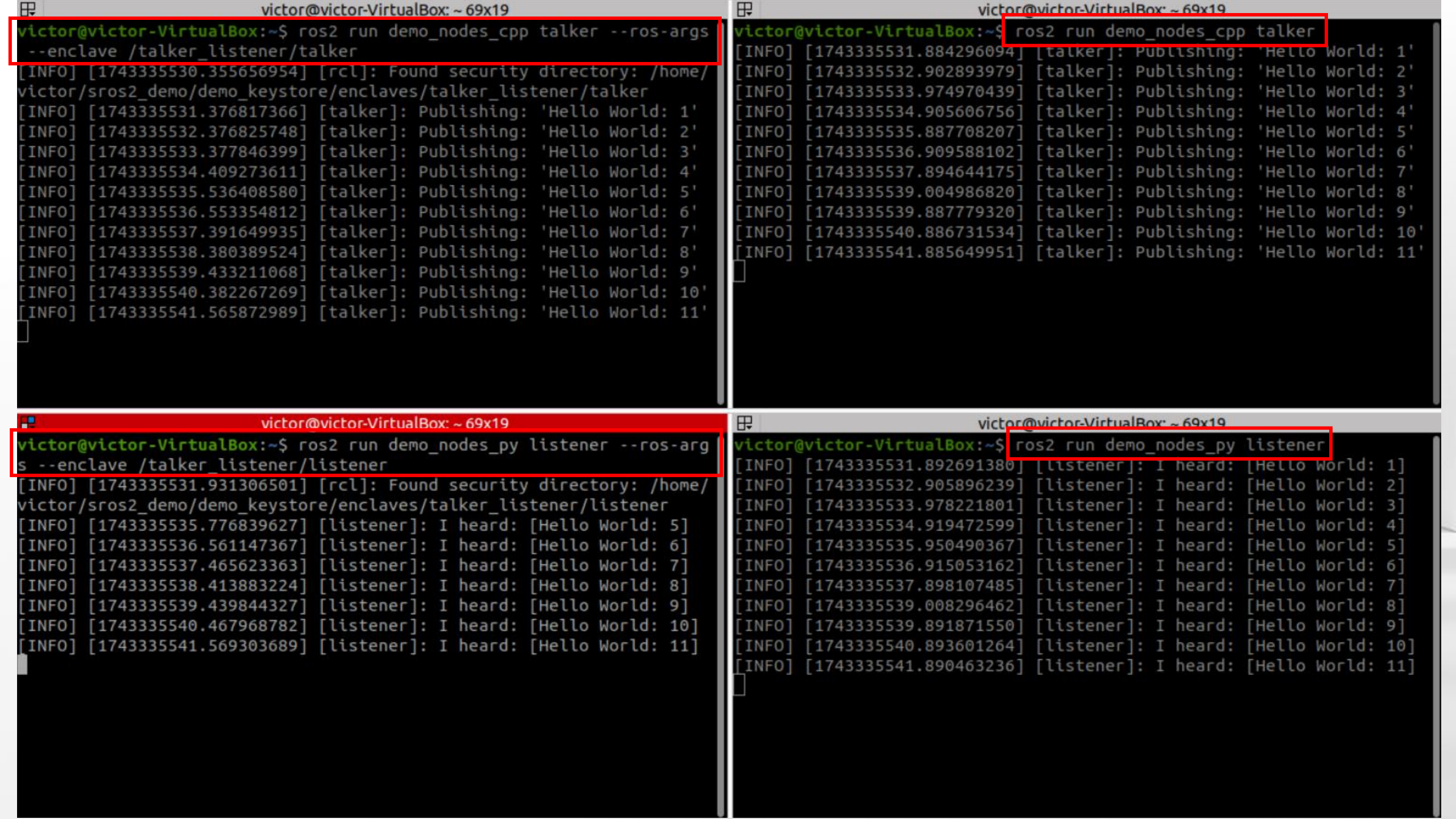
```
$ ros2 run demo_nodes_cpp talker --ros-args --enclave /talker_listener/talker
```

  - 새로운 터미널에서 listener node 실행

```
$ ros2 run demo_nodes_py listener --ros-args --enclave /talker_listener/listener
```

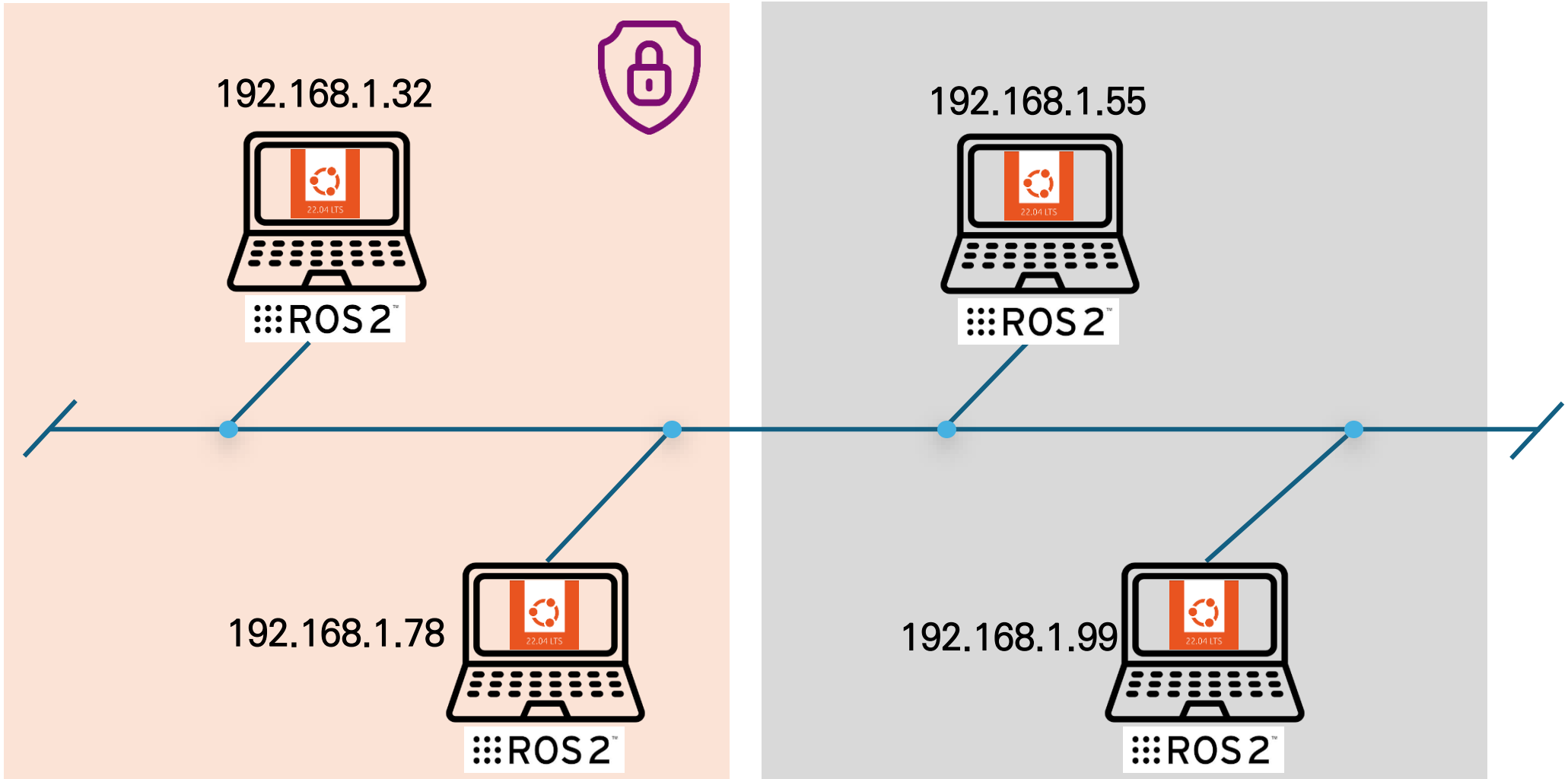
- **이 노드들은 인증 및 암호화를 사용하여 통신 함**
  (해당 노드들은 적절한 키와 인증서를 생성 하였으므로 통신이 가능)


victor@victor-VirtualBox: ~/sros2_demo 140x37
victor@victor-VirtualBox:~/sros2_demo$ tree
.
└── demo_keystore
    ├── enclaves
    │   ├── governance.p7s
    │   ├── governance.xml
    │   └── talker_listener
    │   ├── listener
    │   │   ├── cert.pem
    │   │   ├── governance.p7s -> ../../governance.p7s
    │   │   ├── identity_ca.cert.pem -> ../../../public/identity_ca.cert.pem
    │   │   ├── key.pem
    │   │   ├── permissions_ca.cert.pem -> ../../../public/permissions_ca.cert.pem
    │   │   ├── permissions.p7s
    │   │   └── permissions.xml
    │   └── talker
    │   ├── cert.pem
    │   ├── governance.p7s -> ../../governance.p7s
    │   ├── identity_ca.cert.pem -> ../../../public/identity_ca.cert.pem
    │   ├── key.pem
    │   ├── permissions_ca.cert.pem -> ../../../public/permissions_ca.cert.pem
    │   ├── permissions.p7s
    │   └── permissions.xml
    ├── private
    │   ├── ca.key.pem
    │   ├── identity_ca.key.pem -> ca.key.pem
    │   └── permissions_ca.key.pem -> ca.key.pem
    └── public
        ├── ca.cert.pem
        ├── identity_ca.cert.pem -> ca.cert.pem
        └── permissions_ca.cert.pem -> ca.cert.pem
7 directories, 22 files
victor@victor-VirtualBox:~/sros2_demo$


victor@victor-VirtualBox: ~ 69x19
victor@victor-VirtualBox:~$ ros2 run demo_nodes_cpp talker --ros-args --enclave /talker_listener/talker
[INFO] [1743335530.355656954] [rcl]: Found security directory: /home/victor/sros2_demo/demo_keystore/enclaves/talker_listener/talker
[INFO] [1743335531.376817366] [talker]: Publishing: 'Hello World: 1'
[INFO] [1743335532.376825748] [talker]: Publishing: 'Hello World: 2'
[INFO] [1743335533.377846399] [talker]: Publishing: 'Hello World: 3'
[INFO] [1743335534.409273611] [talker]: Publishing: 'Hello World: 4'
[INFO] [1743335535.536408580] [talker]: Publishing: 'Hello World: 5'
[INFO] [1743335536.553354812] [talker]: Publishing: 'Hello World: 6'
[INFO] [1743335537.391649935] [talker]: Publishing: 'Hello World: 7'
[INFO] [1743335538.380389524] [talker]: Publishing: 'Hello World: 8'
[INFO] [1743335539.433211068] [talker]: Publishing: 'Hello World: 9'
[INFO] [1743335540.382267269] [talker]: Publishing: 'Hello World: 10'
[INFO] [1743335541.565872989] [talker]: Publishing: 'Hello World: 11'
victor@victor-VirtualBox: ~ 69x19
victor@victor-VirtualBox:~$ ros2 run demo_nodes_cpp talker
[INFO] [1743335531.884296094] [talker]: Publishing: 'Hello World: 1'
[INFO] [1743335532.902893979] [talker]: Publishing: 'Hello World: 2'
[INFO] [1743335533.974970439] [talker]: Publishing: 'Hello World: 3'
[INFO] [1743335534.905606756] [talker]: Publishing: 'Hello World: 4'
[INFO] [1743335535.887708207] [talker]: Publishing: 'Hello World: 5'
[INFO] [1743335536.909588102] [talker]: Publishing: 'Hello World: 6'
[INFO] [1743335537.894644175] [talker]: Publishing: 'Hello World: 7'
[INFO] [1743335539.004986820] [talker]: Publishing: 'Hello World: 8'
[INFO] [1743335539.887779320] [talker]: Publishing: 'Hello World: 9'
[INFO] [1743335540.886731534] [talker]: Publishing: 'Hello World: 10'
[INFO] [1743335541.885649951] [talker]: Publishing: 'Hello World: 11'
victor@victor-VirtualBox: ~ 69x19
victor@victor-VirtualBox:~$ ros2 run demo_nodes_py listener --ros-args --enclave /talker_listener/listener
[INFO] [1743335531.931306501] [rcl]: Found security directory: /home/victor/sros2_demo/demo_keystore/enclaves/talker_listener/listener
[INFO] [1743335535.776839627] [listener]: I heard: [Hello World: 5]
[INFO] [1743335536.561147367] [listener]: I heard: [Hello World: 6]
[INFO] [1743335537.465623363] [listener]: I heard: [Hello World: 7]
[INFO] [1743335538.413883224] [listener]: I heard: [Hello World: 8]
[INFO] [1743335539.439844327] [listener]: I heard: [Hello World: 9]
[INFO] [1743335540.467968782] [listener]: I heard: [Hello World: 10]
[INFO] [1743335541.569303689] [listener]: I heard: [Hello World: 11]
victor@victor-VirtualBox: ~ 69x19
victor@victor-VirtualBox:~$ ros2 run demo_nodes_py listener
[INFO] [1743335531.892691380] [listener]: I heard: [Hello World: 1]
[INFO] [1743335532.905896239] [listener]: I heard: [Hello World: 2]
[INFO] [1743335533.978221801] [listener]: I heard: [Hello World: 3]
[INFO] [1743335534.919472599] [listener]: I heard: [Hello World: 4]
[INFO] [1743335535.950490367] [listener]: I heard: [Hello World: 5]
[INFO] [1743335536.915053162] [listener]: I heard: [Hello World: 6]
[INFO] [1743335537.898107485] [listener]: I heard: [Hello World: 7]
[INFO] [1743335539.008296462] [listener]: I heard: [Hello World: 8]
[INFO] [1743335539.891871550] [listener]: I heard: [Hello World: 9]
[INFO] [1743335540.893601264] [listener]: I heard: [Hello World: 10]
[INFO] [1743335541.890463236] [listener]: I heard: [Hello World: 11]

ROS2 CLI

ROS2 CLI 실습



ROKEY BOOT CAMP

수고하셨습니다.