# ROS 기초

## 04 ROS 인터페이스

# 목차 ROS 기초

# 1. ROS2 인터페이스

# ROS2 인터페이스

- 데이터의 형태를 ROS2 인터페이스(interface)라고 함
- ROS2에 새롭게 추가된 IDL(Interface Definition Language)
- 단순 자료형 : integer, floating point, Boolean
- 메시지 안에 메시지를 품고 있는 간단한 데이터 구조
  - geometry_msgs/msgs/Twist의 ‘Vector3 linear’
- 메시지들이 나열된 배열과 같은 구조
  - sensor_msgs/msgs/LaserScan의 ‘float32[] ranges'

# ROS2 인터페이스

- field type0이 메시지 자료형, fieldname0이 메시지 이름
  - fieldtype1 fieldname1
  - fieldtype2 fieldname2
  - fieldtype3 fieldname3
- Ventor4.msg는 아래와 같이 float64자료형에 x, y, z 이름으로 선언
  - float64 x
  - float64 y
  - float64 z

# ROS2 기본 자료형과 언어 자료형 매칭



| Type name | C++ | Python | DDS type |
|---|---|---|---|
| bool | bool | builtins.bool | boolean |
| byte | uint8_t | builtins.bytes* | octet |
| char | char | builtins.str* | char |
| float32 | float | builtins.float* | float |
| float64 | double | builtins.float* | double |
| int8 | int8_t | builtins.int* | octet |
| uint8 | uint8_t | builtins.int* | octet |
| int16 | int16_t | builtins.int* | short |
| uint16 | uint16_t | builtins.int* | unsigned short |
| int32 | int32_t | builtins.int* | long |
| uint32 | uint32_t | builtins.int* | unsigned long |
| int64 | int64_t | builtins.int* | long long |
| uint64 | uint64_t | builtins.int* | unsigned long long |
| string | std::string | builtins.str | string |
| wstring | std::u16string | builtins.str | wstring |

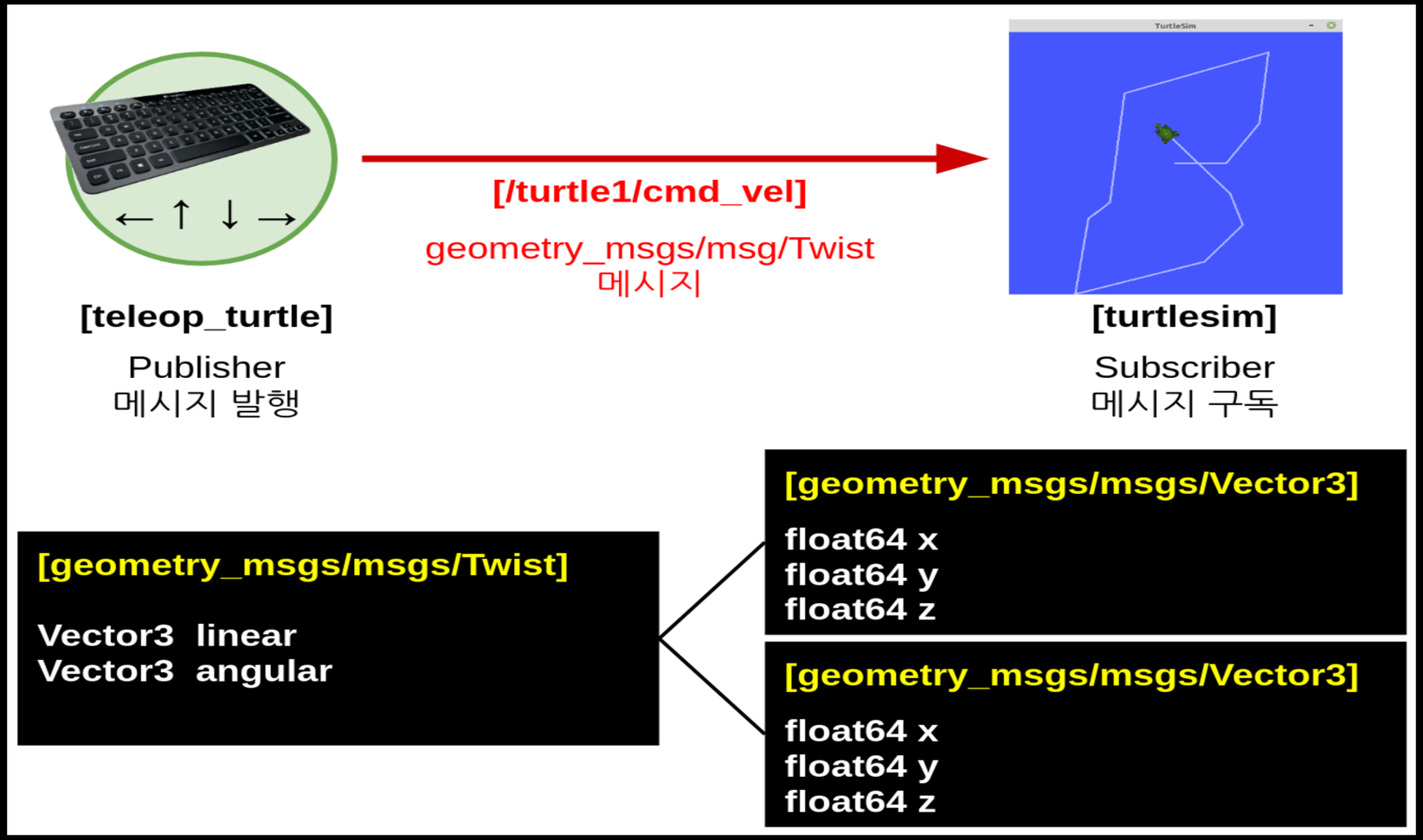
# ROS2 기본 자료형과 언어 자료형 매칭



| Type name | C++ | Python | DDS type |
|---|---|---|---|
| static array | std::array<T, N> | builtins.list* | T[N] |
| unbounded dynamic array | std::vector | builtins.list | sequence |
| bounded dynamic array | custom_class<T, N> | builtins.list* | sequence<T, N> |
| bounded string | std::string | builtins.str* | string |

# 메시지 인터페이스

- 메시지를 통해 병진 속도 3개, 회전 속도 3개 표현



# ROS2 인터페이스

- ros2 interface show

```
$ ros2 interface show geometry_msgs/msg/Twist
Vector3 linear
Vector3 angular
```

```
$ ros2 interface show geometry_msgs/msg/Vector3
float64 x
float64 y
float64 z
```

# ROS2 인터페이스

- list는 현재 개발 환경의 모든 msg, srv, action 출력

```
$ ros2 interface list
Messages:
    action_msgs/msg/GoalInfo
    action_msgs/msg/GoalStatus
    action_msgs/msg/GoalStatusArray
(생략)
Services:
    action_msgs/srv/CancelGoal
    composition_interfaces/srv/ListNodes
(생략)
Actions:
    action_tutorials_interfaces/action/Fibonacci
    example_interfaces/action/Fibonacc
(생략)
```

```
$ ros2 interface packages
action_msgs
action_tutorials_interfaces
actionlib_msgs
builtin_interfaces
(생략)
```

# ROS2 인터페이스

- package는 msr, srv, action 인터페이스를 담고 있는 패키지 목록 출력

```
$ ros2 interface package turtlesim
turtlesim/srv/TeleportAbsolute
turtlesim/srv/SetPen
turtlesim/msg/Color
turtlesim/action/RotateAbsolute
turtlesim/msg/Pose
turtlesim/srv/Spawn
turtlesim/srv/TeleportRelative
turtlesim/srv/Kill
```

```
$ ros2 interface proto geometry_msgs/msg/Twist
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
"
```

# 서비스 인터페이스

- 메시지 인터페이스의 확장형
- 서비스 요청/응답 형태로 구분되는데 '---' 구분자 사용

```
$ ros2 interface show turtlesim/srv/Spawn.srv
float32 x
float32 y
float32 theta
string name
---
string name
```

# 액션 인터페이스

- '---' 구분자를 사용하여 액션 목표, 액션 결과, 피드백으로 나누어 사용

```
$ ros2 interface show turtlesim/action/RotateAbsolute.action
float32 theta
---
float32 delta
---
float32 remaining
```

# 2. ROS2 물리량 표준 단위

# ROS2 표준 단위의 필요성



- 단위의 불일치는 개발자와 사용자간의 불편을 야기하고 치명적인 버그로 이어짐
- 변환 프로그램을 추가로 넣는 방법도 있지만 불필요한 계산 발생
- 개발 초기부터 표준 단위에 관한 규칙을 정립

# ROS2 표준 단위

- 국제단위계인 SI 단위와 일곱 개의 기본 단위를 조합해 만들어진 SI 유도 단위(SI derived unit)을 표준 단위로 정함

| 물리량 | 단위 (SI unit) | 물리량 | 단위 (SI derived unit) |
|---|---|---|---|
| length (길이) | meter (m) | angle (평면각) | radian (rad) |
| mass (질량) | kilogram (kg) | frequency (주파수) | hertz (Hz) |
| time (시간) | second (s) | force (힘) | newton (N) |
| current (전류) | ampere (A) | power (일률) | watt (W) |
| | | voltage (전압) | volt (V) |
| | | temperature (온도) | celsius (°C) |
| | | magnetism (자기장) | tesla (T) |

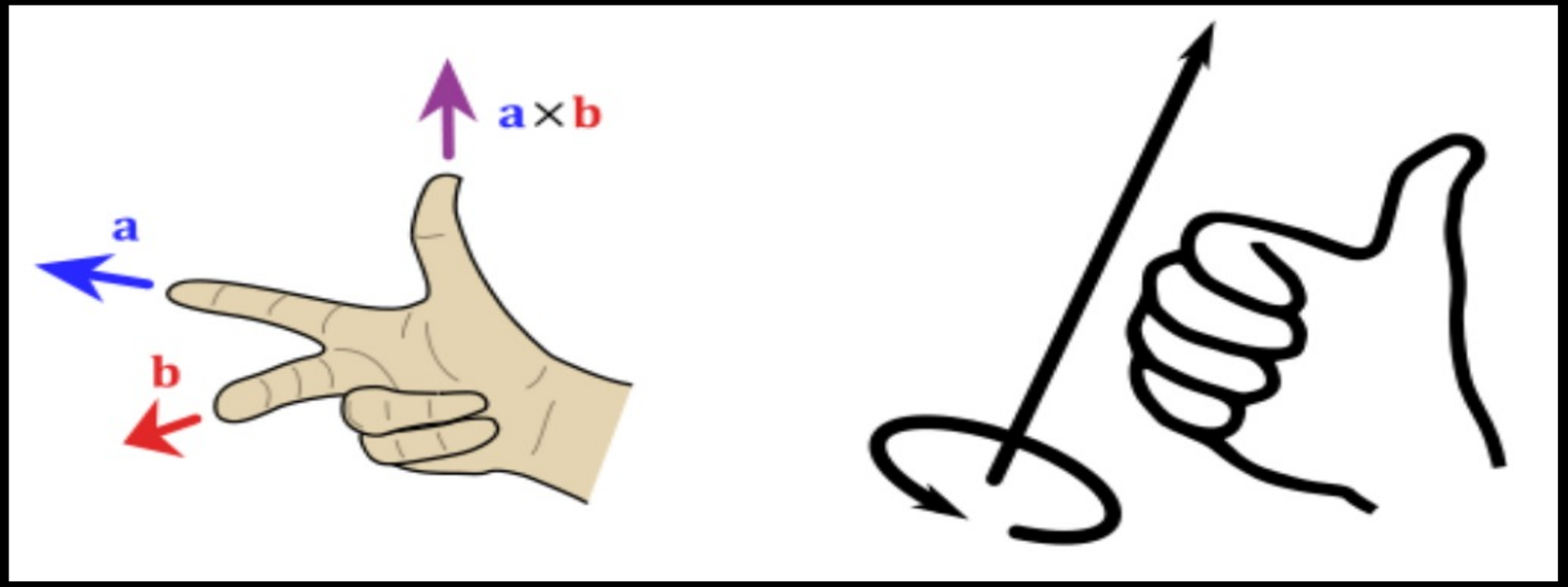
# ROS2의 좌표



- 좌표계(coordinate system) 불일치를 사전에 막기위한 규칙
- 컴퓨터 비전에서는 z forward, x right, y down를 기본 좌표로 사용
- 로봇은 x forward, y left, z up을 기본 좌표계로 사용
- Lidar 및 IMU, Torque 센서들도 제조사 별로 다른 좌표계를 사용할 수 있고 변환 API 있더라도 기본 출력을 미리 알아볼 필요가 있음
- ex) IMU가 NED 타입(x north, y east, z down, relative to magnetic north) 이냐 ENU(x-east, y-north, z-up, relative to magnetic north)
- 각 센서들의 좌표를 로봇 좌표에 맞추어 적절한 좌표 변환 필요

# ROS2 좌표 기본 규칙

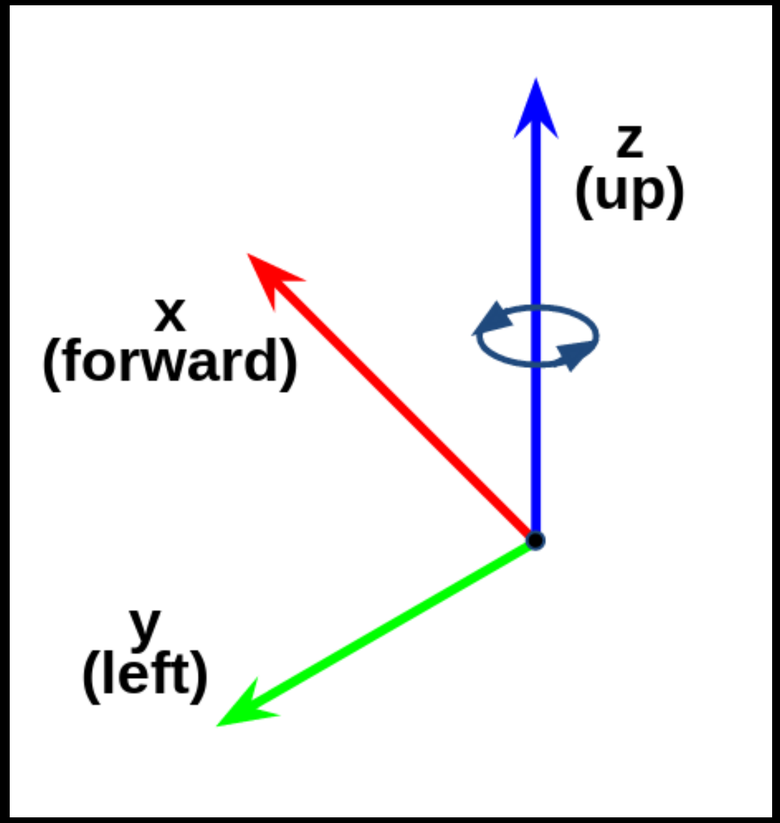
- 모든 좌표계를 삼차원 벡터 표기관습을 이해하기 위한 오른손 법칙에 따라 표현
- 회전의 경우 검지/중지/엄지를 축을 사용, 오른손의 손가락을 감는 방향이 정회전 + 방향




- 로봇이 제자리에서 12시 방향 -> 9시 방향으로 회전했다면 로봇은 정방향 + 1.6708 rad 만큼 회전했다고 말할 수 있음

# ROS2 축 방향(Axis Orientation) 규칙

- 축 방향(Axis Orientation)으로 x forward, y left, z up을 사용
- RvizLh Gazebo에서 기본 3축 표현에 헷갈리지 않도록 RGB 원색으로 표현



# ROS2 축 방향(Axis Orientation) 규칙



- 지리적 위치(geographic locations)의 단거리 데카르트 표현의 경우 ENU(east north up 규칙을 사용(큰 맵을 다루는 드론, 실외 자율주행 로봇에서 사용하는 좌표)
- 접미사 프레임 사용(Suffix Fames) : ENU 좌표에서 벗어나는 경우 접미사 프레임을 사용하여 기본 좌표계와 구별하여 사용(_optical, _ned)
  - 카메라 센서 메시지에 '_optical' 접미사를 붙여 구분(카메라 좌표계와 로봇 좌표계 간의 TF(transform) 필요
  - 실외 동작 시스템의 경우 센서 및 지도에 따라 NED 사용하고 '_ned' 접미사 사용

# ROS2 회전 표현(Rotation Representation) 규칙

- 쿼터니언(quaternion) : 간결한 표현 방식으로 가장 널리 사용(x, y, z, w), 특이점 없음(No singularities)
- 회전 매트릭스(rotation matrix) : 특이점 없음(No singularities)
- 고정축 roll, pitch, yaw((fixed axis roll, pitch, yaw about X, Y, Z axes respectively)
- 오일러 각도 yaw, pitch, roll(euler angles yaw, pitch, and roll about Z, Y, X axes respectively) : 한 축 회전이 다른 축 회전과 겹치는 문제(일명 짐벌락)로 인해 사용을 권장하지 않음

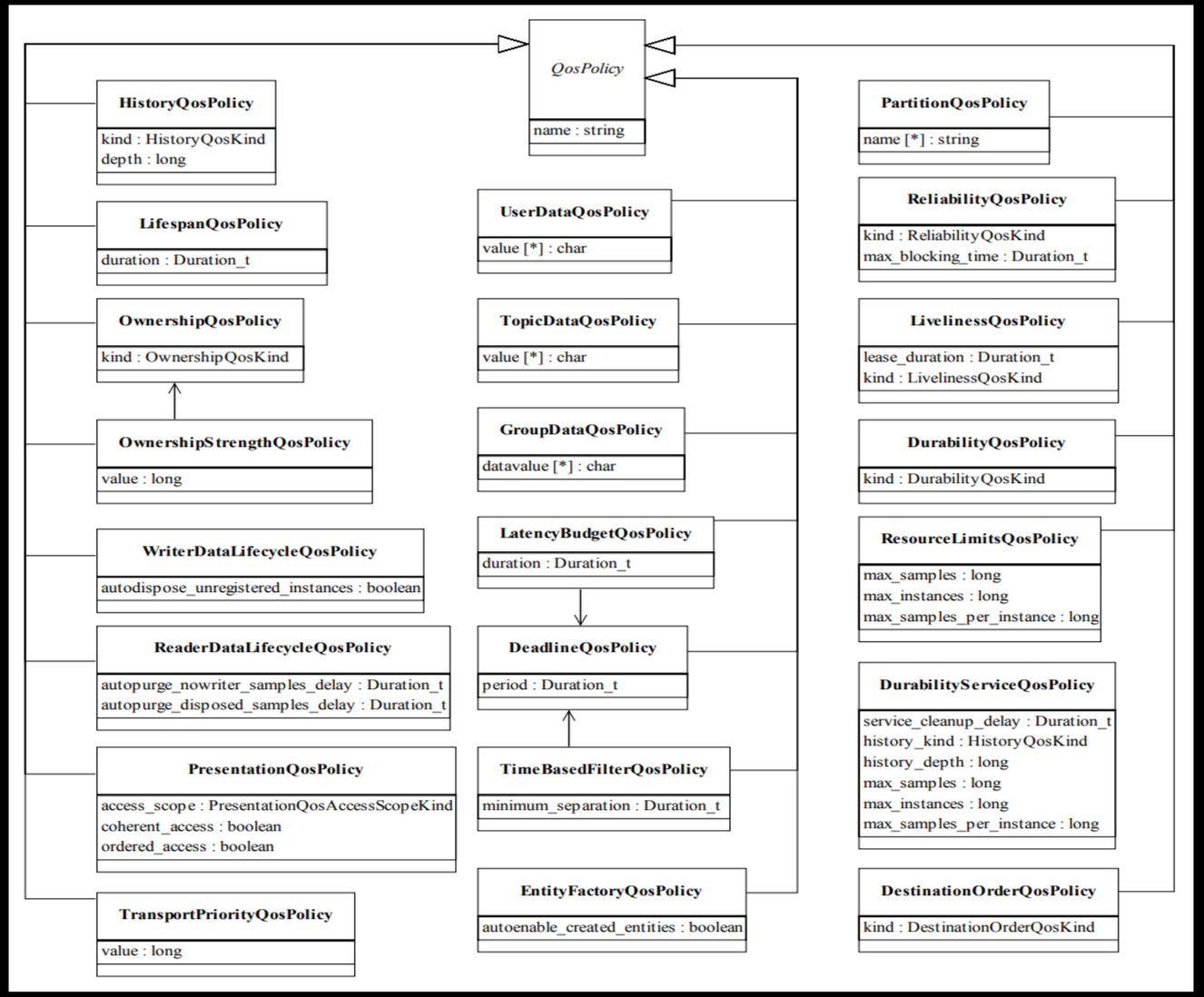
# 3. DDS의 QOS(Quality of Service)

# DDS의 서비스 품질

- Qos는 데이터 통신의 옵션
- ROS2 는 TCP 및 UCP를 선택적으로 사용가능하고 여러가지 통신 설정을 유저가 직접 할 수 있음
- 실시간성 성정 및 대역폭, 지속성/중복성과 관련된 옵션 등
- 어떤 상황에서 어떻게 설정하느냐가 중요

# QoS 종류




QosPolicy
name : string
HistoryQosPolicy
kind : HistoryQosKind
depth : long
LifespanQosPolicy
duration : Duration_t
OwnershipQosPolicy
kind : OwnershipQosKind
OwnershipStrengthQosPolicy
value : long
WriterDataLifecycleQosPolicy
autodispose_unregistered_instances : boolean
ReaderDataLifecycleQosPolicy
autopurge_nowriter_samples_delay : Duration_t
autopurge_disposed_samples_delay : Duration_t
PresentationQosPolicy
access_scope : PresentationQosAccessScopeKind
coherent_access : boolean
ordered_access : boolean
TransportPriorityQosPolicy
value : long
UserDataQosPolicy
value [*] : char
TopicDataQosPolicy
value [*] : char
GroupDataQosPolicy
datavalue [*] : char
LatencyBudgetQosPolicy
duration : Duration_t
DeadlineQosPolicy
period : Duration_t
TimeBasedFilterQosPolicy
minimum_separation : Duration_t
EntityFactoryQosPolicy
autoenable_created_entities : boolean
PartitionQosPolicy
name [*] : string
ReliabilityQosPolicy
kind : ReliabilityQosKind
max_blocking_time : Duration_t
LivelinessQosPolicy
lease_duration : Duration_t
kind : LivelinessQosKind
DurabilityQosPolicy
kind : DurabilityQosKind
ResourceLimitsQosPolicy
max_samples : long
max_instances : long
max_samples_per_instance : long
DurabilityServiceQosPolicy
service_cleanup_delay : Duration_t
history_kind : HistoryQosKind
history_depth : long
max_samples : long
max_instances : long
max_samples_per_instance : long
DestinationOrderQosPolicy
kind : DestinationOrderQosKind

# QoS 종류

- Reliability : TCP처럼 데이터 손실을 방지함으로써 신뢰도를 우선시하거나 (reliable), UDP처럼 통신 속도를 최우선시하여 사용(best effort)
- History : 통신 상태에 따라 정해진 사이즈만큼의 데이터를 보관
- Durability : 데이터를 수신하는 서브스크라이버가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 설정
- Deadline : 정해진 주기 안에 데이터가 발신 및 수신되지 않을 경우 이벤트 함수를 실행
- Lifespan : 정해진 주기 안에서 수신되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제
- Liveliness : 정해진 주기 안에서 노드 혹은 토픽의 생사 확인

# ROS2에서 QoS 옵션



- History – values

| History | 데이터를 몇 개나 보관할지를 결정하는 QoS 옵션 |
|---|---|
| KEEP_LAST | 정해진 메시지 큐 사이즈 만큼의 데이터를 보관<br>* depth: 메시지 큐의 사이즈 (KEEP_LAST 설정일 경우에만 유효) |
| KEEP_ALL | 모든 데이터를 보관 (메시지 큐의 사이즈는 DDS 벤더마다 다름) |

- RxO(requested by offered) – 해당사항 없음

```
[RCLCPP]
rclcpp::QoS(rclcpp::KeepLast(10)).reliable().durability_volatile();
```

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

# ROS2에서 QoS 옵션

- Reliability – values

| Reliability | 데이터 전송에 있어 속도를 우선시 하는지 신뢰성을 우선시 하는지를 결정하는 QoS 옵션 |
|---|---|
| BEST_EFFORT | 데이터 송신에 집중. 전송 속도를 중시하며 네트워크 상태에 따라 유실이 발생할 수 있음 |
| RELIABLE | 데이터 수신에 집중. 신뢰성을 중시하며 유실이 발생하면 재전송을 통해 수신을 보장함 |

- RxO(requested by offered)

| Pub(↓) / Sub (→) | BEST_EFFORT | RELIABLE |
|---|---|---|
| BEST_EFFORT | BEST_EFFORT | 불가 |
| RELIABLE | BEST_EFFORT | RELIABLE |

- examples

```
[RCLCPP]
rclcpp::QoS(rclcpp::KeepAll).best_effort().transient_local();
```

```
[RCLPY]
qos_profile = QoSProfile(reliability=QoSReliabilityPolicy.BEST_EFFORT)
```

# DDS의 서비스 품질

- Durability

| Durability | 데이터를 수신하는 서브스크라이버가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 QoS 옵션 |
|---|---|
| TRANSIENT_LOCAL | Subscription이 생성되기 전의 데이터도 보관 (Publisher에만 적용 가능) |
| VOLATILE | Subscription이 생성되기 전의 데이터는 무효 |

- RxO(requested by offered)

| Pub(↓) / Sub (→) | TRANSIENT_LOCAL | VOLATILE |
|---|---|---|
| TRANSIENT_LOCAL | TRANSIENT_LOCAL | VOLATILE |
| VOLATILE | 불가 | VOLATILE |

- ex

```
[RCLCPP]
rclcpp::QoS(rclcpp::KeepAll).best_effort().transient_local()
```

```
[RCLPY]
qos_profile = QoSProfile(durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)
```

# DDS의 서비스 품질

- Deadline

| Deadline | 정해진 주기 안에 데이터가 발신 및 수신되지 않을 경우 EventCallback를 실행시키는 QoS 옵션 |
|---|---|
| deadline_duration | Deadline을 확인하는 주기 |

- RxO(requested by offered)

| Pub(↓) / Sub (→) | 1000ms | 2000ms |
|---|---|---|
| 1000ms | 가능 | 가능 |
| 2000ms | 불가 | 가능 |

- Ex

```
[RCLCPP]
rclcpp::QoS(10).deadline(100ms);
```

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

# DDS의 서비스 품질

- Lifespan

| Lifespan | 정해진 주기 안에서 수신되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제하는 QoS 옵션 |
|---|---|
| lifespan_duration | Lifespan을 확인하는 주기 |

- RxO(requested by offered) – 해당사항 없음
- ex)

```
[RCLCPP]
rclcpp::QoS(10).reliable().transient_local().lifespan(10ms);
```

```
[RCLPY]
qos_profile = QoSProfile(lifespan=Duration(0.01))
```

# DDS의 서비스 품질

- Liveliness

| Liveliness | 정해진 주기 안에서 노드 혹은 토픽의 생사 확인하는 QoS 옵션 |
|---|---|
| liveliness | 자동 또는 매뉴얼로 확인할지를 지정하는 옵션, 하기 3가지 중 선택<br>* AUTOMATIC, MANUAL_BY_NODE, MANUAL_BY_TOPIC 중 선택 |
| lease_duration | Liveliness을 확인하는 주기 |

- RxO(requested by offered)

| Pub(↓) / Sub (→) | AUTOMATIC | MANUAL_BY_NODE | MANUAL_BY_TOPIC |
|---|---|---|---|
| AUTOMATIC | 가능 | 불가 | 불가 |
| MANUAL_BY_NODE | 가능 | 가능 | 불가 |
| MANUAL_BY_TOPIC | 가능 | 가능 | 가능 |

# DDS의 서비스 품질

- automatic : 생존성을 관리하는 주체가 DDS 미들웨어
- Manual : 생존성을 관리하는 주체가 사용자

```
[RCLCPP]
rclcpp::QoS qos_profile(10);
  qos_profile
  .liveliness(RMW_QOS_POLICY_LIVELINESS_AUTOMATIC)
  .liveliness_lease_duration(1000ms);
```

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

# rmw_qos_profile 사용

- rmw_qos_profile : 가장 많이 사용하는 QoS 설정을 세트로 표현해둔 것으로 목적에 따라 6가지로 분류

| | 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 Depth) | 10 | 5 | 10 | 1 | 1,000 | 1,000 |
| Durability | VOLATILE | VOLATILE | VOLATILE | TRANSIENT LOCAL | VOLATILE | VOLATILE |

# rmw_qos_profile 사용



- 센서와 같이 지속성이 높으며 데이터 유실보다는 순간 데이터를 가장 빠르게 전달해야 하는 sensor data의 경우는 아래와 같이 설정되어 있음

```
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
};
```

# rmw_qos_profile 사용



- 실제 코드에서 사용

```
from rclpy.qos import qos_profile_sensor_data
```

- 퍼블리셔를 선언하는 'create_publisher' 함수에서 '밴_profile_sensor_data'를 매개변수로 사용

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

# 유저 QoS 프로파일

- 유저가 직접 설정하여 새로운 프로파일을 생성 하여 사용

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

- QoSProfile을 선언하여 아래와 같이 원하는 옵션으로 커스텀

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

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