

# Week\_10

## 3부 ROS2 심화 프로그래밍

### 1장 Logging

#### 1.1 로그(Log)

- 프로그래밍 언어에서 로그(Log)는 프로그램을 개발하는 과정에서 개발자가 프로그램을 검토/디버깅하는 도구로도 사용되고, 프로그램 사용자가 프로그램에 대한 정보를 얻는 용도로도 사용된다.
  - 개발자 : 검토 / 디버깅
  - 사용자 : 프로그램에 대한 정보를 얻는 용도로 활용된다.

#### 1.2 로그 설정

- 로그가 저장되는 디렉터리이다.

```
$ ls ~/.ros/log
```

```

kody@desktop:~$ ls ~/.ros/log
2023-05-18-16-18-52-597491-desktop-52483
2023-05-18-16-20-42-921120-desktop-53316
2023-05-18-16-22-38-211576-desktop-54083
2023-05-20-18-15-07-182771-desktop-29049
2023-05-20-18-17-35-528581-desktop-29674
2023-05-20-18-19-09-302907-desktop-30145
2023-05-20-18-21-55-531042-desktop-31141
2023-05-20-18-26-20-709996-desktop-32454
cam2image_63159_1683792288109.log
cam2image_64207_1683792367604.log
cam2image_64343_1683792410084.log
cam2image_65182_1683792734235.log
gzserver_52486_1684394333340.log
helloworld_publisher_100171_1685621252254.log
helloworld_subscriber_100089_1685621230461.log
io_manage_86538_1685882218447.log
kit_17374_1683282563022.log
kit_25006_1683290632810.log
kit_25006_1683290776320.log
kit_25006_1683290788097.log
kit_25006_1683290828464.log
kit_25006_1683290831707.log
kit_25006_1683291399317.log
kit_25006_1683291475294.log
kit_25006_1683291489578.log
kit_25006_1683291712393.log
kit_25006_1683291829171.log
kit_25006_1683292049028.log

```

- ROS2에서의 로그 수준은 총 5가지이다.
  - DEBUG
  - INFO
  - WARN
  - ERROR
  - FATAL
- 로그 수준을 설정하는 3가지 방법을 알아보자

#### 1. 코드에 명시하는 방법

```

// C++
RCLCPP_INFO(this->get_logger(), "Published message: '%s'", msg.data.c_str());

```

```
# 파이썬
self.get_logger().info('Published message: {0}'.format(msg.data))
```

- rclcpp에서는 **매크로 함수**로 로그의 수준을 정하고, ( RCLCPP\_INFO() )  
해당 함수에
  1. logger
  2. 출력할 문자열
  3. 로그를 남기고 싶은 변수
 를 남겨서 **로그를 작성**할 수 있다.
- rclpy에서도 get\_logger().**경고수준** 으로 로그의 수준을 정해서 로그를 작성할 수 있다.
  - get\_logger().**경고수준('출력할 문자열',format(로그를 남기고 싶은 변수))**

- RCLCPP\_INFO는 **RCLCPP\_\${SEVERITY} 형식**을 이용하여 **로그를 작성**한 것이다.

이와 같이, 형식을 이용해서 로그를 작성하는 방법은 다음과 같다.

- \${SEVERITY}에는 로그 수준을 넣는다.
  - RCLCPP\_\${SEVERITY} : Formatting을 지원하는 함수이다.
    - RCLCPP\_\${SEVERITY}\_ONCE : **딱 한번만 출력**되는 함수이다.
    - RCLCPP\_\${SEVERITY}\_EXPRESSION : **Expression이 True**일때 출력
    - RCLCPP\_\${SEVERITY}\_FUNCTION : **Function이 True**일때 출력
    - RCLCPP\_\${SEVERITY}\_SKIPFIRST : 말 그대로, 첫번째 함수는 생략하고 **두 번째 함수부터 출력**함
    - RCLCPP\_\${SEVERITY}\_THROTTLE : 특정 주기마다 출력되는 함수이다.
    - RCLCPP\_\${SEVERITY}\_SKIPFIRST\_THROTTLE : 두 번째 호출부터 특정 주기마다 출력되는 함수이다.

## 2. Externally

- 일반적으로 사용자는 DEBUG 로그 수준까지 확일할 필요는 없지만, 이를 확인하기 위해서 ROS1에서는 `rqt_logger_level` 노드를 통해, 특정 로그의 레벨을 런타임에서 변경하는 기능을 제공하고  
아직까지 ROS2에서는 지원하지는 않지만 곧 지원한다고 한다.

## 3. Command Line

- 노드를 실행시킬때 인자를 전달해서 노드의 로그 수준을 전달할 수 있다.
- 다음과 같이 데모 코드 뒤에 인자를 전달해서 로그 수준을 지정할 수 있다.

```
$ ros2 run logging_demo logging_demo_main --ros-args --log-level debug
```

### 1.2.3 로그 형식 지정

- ROS2는 터미널에서 로그 형식을 변경할 수 있고, 이는 환경변수를 통해 설정이 가능하다.

```
$ export RCUTILS_CONSOLE_OUTPUT_FORMAT="[severity] [time] [{name}]: {message}  
([{function_name}]() at {file_name}:{line_number})"
```

## 로그 관련 설정들

- 로그 색상 설정

default값은 INFO는 흰색, WARN은 노란색, ERROR는 빨간색으로 표시되지만 이를 변경할 수 있다.

```
$ export RCUTILS_COLORIZED_OUTPUT=0 # 1 for force it
```

```
터미널에서 확인하는 명령어  
echo $RCUTILS_COLORIZED_OUTPUT
```

```
kody@desktop:~$ echo $RCUTILS_COLORIZED_OUTPUT
0
kody@desktop:~$
```

- 로그 스트림 설정

- 이전 버전에서는

- DEBUG와 INFO수준의 로그가 **stdout** 스트림으로 설정되어 있고,
    - WARN, ERROR, FATAL 수준의 로그는 **stderr** 스트림으로 설정되어 있어서
    - **stdout 버퍼가 모두 차지 않으면 로그를 확인할 수 없는 문제**가 있었고, 이를 해결하기 위해 메인 함수에 force flush 함수를 사용해야만 했다.

- Foxy 부터는 DEBUG와 INFO에서도 **stderr 스트림**을 사용하도록 **기본 설정**되어 있으며, **stdout 스트림**을 사용하고 싶으면 다음 명령어로 **환경변수를 설정**할 수 있다.

```
$ export RCUTILS_LOGGING_USE_STDOUT = 1
```

- 로그 라인 버퍼링 설정 **INFO, DEBUG 수준**의 로그가 **라인 버퍼링**을 하지 않도록 **기본 설정**되어 있다.
  - 라인 버퍼링을 사용하고 싶으면 다음 변수를 사용해 설정할 수 있다.

```
$ export RCUTILS_LOGGING_BUFFERED_STREAM=1
```

- ROS2 로그 관련 환경변수는 다음과 같이 rc(run command)파일인 ~/.bashrc 파일에 설정하고 사용 가능하다.

```
# export RCUTILS_CONSOLE_OUTPUT_FORMAT="[severity] {time} [{name}]: {message}
({function_name}() at {file_name}:{line_number})"

export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity}]: {message}'
export RCUTILS_COLORIZED_OUTPUT=1
export RCUTILS_LOGGING_USE_STDOUT=0
export RCUTILS_LOGGING_BUFFERED_STREAM=0
```

```
alias ros2="source ~/ros2/install/local_setup.bash;"
echo "ROS2 Humble Activated\""

export PATH=$PATH:/home/kody/.local/bin

alias ros2study="source ~/ros2_study/install/local_setup.bash;"
echo "ros2_study workspace is activated\""

alias ros2example="source ~/ros2_example/install/local_setup.bash;"
echo "ros2_example workspace is activated\""
export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity}]: {message}'
export RCUTILS_COLORIZED_OUTPUT=1
export RCUTILS_LOGGING_USE_STDOUT=0
export RCUTILS_LOGGING_BUFFERED_STREAM=0

kody@desktop:~$ export "export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity}]: {message}'" >> ~/.bashrc
bash: export: `export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity}]: {message}''':
not a valid identifier
kody@desktop:~$ echo "export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity}]: {message}'" >> ~/.bashrc
kody@desktop:~$ echo "export RCUTILS_COLORIZED_OUTPUT=1" >> ~/.bashrc
kody@desktop:~$ echo "export RCUTILS_LOGGING_USE_STDOUT=0" >> ~/.bashrc
kody@desktop:~$ echo "export RCUTILS_LOGGING_BUFFERED_STREAM=0" >> ~/.bashrc
kody@desktop:~$
```

- 터미널 출력으로 ~/.bashrc 파일에 환경변수를 작성해 주었다.

## 2장 ROS2 CLI

### 2.3 ROS2 CLI

- ROS2 CLI의 형식이다.

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

#### 사용 예시

- ros2cli + [verbs] + [arguments]
  - ros2 run <package> <executable>
  - ros2 launch <package> <launch-file>
- ros2cli + [verbs] + [sub-verbs]
  - ros2 pkg [create, executables, list, prefix, xml]
  - ros2 node [info, list]

- ros2 topic [bw, delay ,echo, find, hz, info, list, pub ,type]
- ros2 service [call, find, list, type]
- ros2 action [info, list, send\_goal]
- ros2 interface [list, package, packages, proto, show]
- ros2 param [delete, describe, dump, get, list, set]
- ros2 bag [info, play, record]
- ros2cli + [verbs] + [sub-verbs] + (options)
  - ros2 extensions (-a, -v)
  - ros2 extensions\_point (-a, -v)
  - ros2 daemon [start, statue, stop]
  - ros2 multicase [receive, send]
  - ros2 doctor [hello, (-r), (-rf), (-iw)]
  - ros2 wtf [hello, (-r), (-rf), (-iw)]
  - ros2 lifecycle [get, list, nodes, set]
  - ros2 component [list, load, standalone, types, unload]
  - ros2 security [create\_key, create\_keystore, create\_permission, generate\_artifacts, generate\_policy, list\_keys]

## 2.2 ROS2 CLI 실습

### 2.2.1 ros2 run

- run은 특정 패키지의 특정 노드를 실행하는 명령어이다.

```
$ ros2 run turtlesim turtlesim_node
$ ros2 run turtlesim turtle_teleop_key
```

### 2.2.2 ros2 launch

- launch는 특정 패키지의 런치 파일을 실행하는 명령어이다.
  - 설정만 변경하여 노드를 실행하거나

```
Node(
    package='my_package',
    executable='talker',
    parameters=[
        {'frequency': 10.0}, # Changing the frequency parameter
    ], 와 같이 파라미터 값을 launch파일에서 변경하여 실행할 수 있다.
```

- 복수개의 노드를 실행하거나
- 런치에서 또 다른 패키지의 런치 파일을 실행할 수도 있다.

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

### 2.2.3 ros2 pkg

- 패키지 생성
- ros2 pkg executables turtlesim
  - 패키지에 포함된 실행 파일 목록을 출력한다.

```
kody@desktop:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

- ros2 pkg list
  - 패키지 리스트를 출력한다.



```
kody@desktop:~$ ros2 pkg list
ackermann_msgs
action_msgs
action_tutorials_cpp
action_tutorials_interfaces
action_tutorials_py
actionlib_msgs
admittance_controller
ament_cmake
ament_cmake_auto
ament_cmake_copyright
ament_cmake_core
ament_cmake_cppcheck
ament_cmake_cpplint
ament_cmake_export_definitions
ament_cmake_export_dependencies
ament_cmake_export_include_directories
ament_cmake_export_interfaces
ament_cmake_export_libraries
ament_cmake_export_link_flags
ament_cmake_export_targets
ament_cmake_flake8
```

- ros2 pkg prefix turtlesim
  - 패키지의 저장 위치를 확인한다.

```
kody@desktop:~$ ros2 pkg prefix turtlesim
/opt/ros/humble
```

- ros2 pkg xml turtlesim
  - 패키지의 패키지 정보 파일(package.xml)을 확인한다.

```
kody@desktop:~$ ros2 pkg xml turtlesim
<package format="3">
  <name>turtlesim</name>
  <version>1.4.2</version>
  <description>
    turtlesim is a tool made for teaching ROS and ROS packages.
  </description>

  <maintainer email="audrow@openrobotics.org">Audrow Nash</maintainer>
  <maintainer email="michael.jeronimo@openrobotics.org">Michael Jeronimo</maintainer>

  <license>BSD</license>

  <url type="website">http://www.ros.org/wiki/turtlesim</url>
  <url type="bugtracker">https://github.com/ros/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ros/ros_tutorials</url>
```

## 2.2.4 ros2 node

- 노드의 정보를 얻는 데 사용되는 명령어이다.
  - \$ ros2 node list

```
kody@desktop:~$ ros2 node list
/teleop_turtle
/turtlesim
kody@desktop:~$
```

- \$ ros2 node info /turtlesim

```
kody@desktop:~$ ros2 node info /turtlesim
/turtlesim
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/color_sensor: turtlesim/msg/Color
  /turtle1/pose: turtlesim/msg/Pose
Service Servers:
  /clear: std_srvs/srv/Empty
  /kill: turtlesim/srv/Kill
  /reset: std_srvs/srv/Empty
  /spawn: turtlesim/srv/Spawn
  /turtle1/set_pen: turtlesim/srv/SetPen
  /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
  /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
```

## 2.2.5 ros2 topic

- 토픽의 구성, 대역폭, 지연시간, 인터페이스의 정보를 얻거나
- 토픽을 송신 및 수신하는데 사용되는 명령어이다.

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

```
kody@desktop:~$ ros2 topic bw /turtle1/cmd_vel
Subscribed to [/turtle1/cmd_vel]
68 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
41 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
29 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
23 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
19 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
24 B/s from 3 messages
```

```
$ ros2 topic delay /image
```

- 특정 토픽의 지연시간(delay)을 확인하는 명령어이다.
  - delay명령어는 Header 인터페이스를 포함한 경우에만 사용 가능하다.

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

- 토픽(topic)의 데이터를 출력(echo)하는 명령어이다.

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

- 특정한 인터페이스(geometry\_msgs/msg/Twist등)를 사용하고 있는 토픽명을 확인하는 명령어이다.

---

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

- 토픽의 주기를 확인한다.

---

```
$ ros topic list -t
```

- **동작중인** 모든 노드들의 토픽 이름과 (topic list)
- 모든 노드들의 인터페이스 형태를 함께 출력한다 ( -t 옵션)

```
^Ckody@desktop:~$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
```

/turtle1/color\_sensor : 토픽 이름 부분

[turtlesim/msg/Color] : type( -t옵션) 부분

---

```
$ ros2 topic pub --once /turtle/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z:0.0}, angular:{x:0.0, y:0.0, z:1.8}}"
```

- - -once를 이용하여 topic을 한번만 발행하는 CLI이다
- - -once 대신 - -rate 1을 사용하면 1Hz간격으로 topic을 발행한다.

```
$ ros2 topic type /turtle/cmd_vel
```

해당 토픽(/turtle/cmd\_vel)의 인터페이스 형태 ( geometry\_msgs/msg/Twist)를 확인한다.

## 2.2.6 ros2 service

- 서비스의 정보를 얻거나,
- 직접 서비스 요청을 테스트해 볼 수 있는 명령어이다.

```
$ ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r:255, g:255, b:255, width:10}"
requester: making request: turtlesim.srv.SetPen_Request(r:255, g:255, b:255, width:10, off=0)
response:
turtlesim.srv.SetPen_Response()
```

- /turtle1/set\_pen 서비스를 turtlesim/srv/SetPen 인터페이스를 사용해서 콜한다.

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

- 해당 인터페이스(std\_srvs/srv/Empty)를 사용하는 서비스(service) 명을 확인한다.

```
$ ros2 service list
```

- 현재 **실행중인** 서비스 리스트를 확인한다.

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

- 서비스를 입력하여( /clear ) 인터페이스(std\_srvs/srv/Empty)를 확인한다.
- 

## 2.2.7 ros2 action

- 액션의 정보를 얻거나
- 직접 액션 목표 전달을 테스트해 볼 수 있는 명령어이다.

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

- 해당 액션을 사용하는 Action Server와 Action Client 노드 이름과 갯수를 출력한다.
- 

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

- 액션 이름과 인터페이스 형태를 출력한다.

```
kody@desktop:~$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]
```

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

- 인터페이스 / 액션을 입력한 후 액션 목표값( theta : 1.5708)을 전달한다.

## ros2 interface

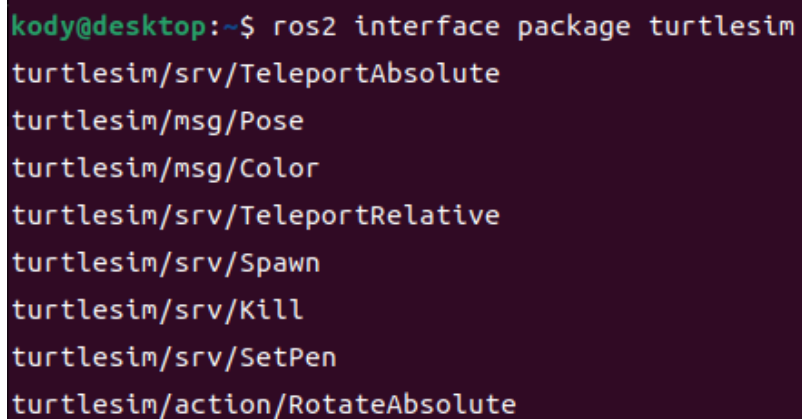
- 인터페이스의 정보를 얻는데 사용되는 명령어이다.

```
$ ros2 interface list
Message:
~~
Service:
~~
Action:
~~
```

- 현재 환경의 모든 msg, srv, action 인터페이스를 확인한다.
- 

```
$ ros2 interface package turtlesim
```

- 해당 패키지에 포함된 인터페이스를 확인한다.



```
kody@desktop:~$ ros2 interface package turtlesim
turtlesim/srv/TeleportAbsolute
turtlesim/msg/Pose
turtlesim/msg/Color
turtlesim/srv/TeleportRelative
turtlesim/srv/Spawn
turtlesim/srv/Kill
turtlesim/srv/SetPen
turtlesim/action/RotateAbsolute
```

---

```
$ ros2 interface proto geometry_msgs/msg/Twist
```

- 해당 패키지의 기본 형태를 확인한다.
- 

```
$ ros2 interface show geometry_msgs/msg/Twist
```

- 메시지 이름 / 인터페이스 이름을 확인한다.