

# ROS2\_day1-1

## Day\_1-1

```
#include "geometry_msgs/msg/twist.hpp"
#include "rclcpp/parameter_client.hpp"
#include "turtlesim/srv/set_pen.hpp"
#include "std_srvs/srv/empty.hpp"
#include "turtlesim/srv/spawn.hpp"
#include "turtlesim/srv/kill.hpp"
```

- geometry\_msgs/msg/twist.hpp  
: twist 메세지 타입 포함, 터틀의 선속도, 각속도 제어  
3개의 floated 변수 ⇒ 묶음 두개(linear묶음, angular 묶음)

```
rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr control_publisher_ = this->create_publisher<geomtr...~>("/tur
```

```
void TurtlesimCliNode::control_mode() {
    std::cout << "w,a,s,d 를 눌러 조종화세요, q 입력시 종료\n";
    geometry_msgs::msg::Twist twist;
    char input;
    while ((std::cin >> input) && (input != 'q')) {
        if (input == 'w')
            twist.linear.x = 1.0;
        else if (input == 's')
            twist.linear.x = -1.0;
        else if (input == 'a')
            twist.angular.z = 1.0;
        else if (input == 'd')
            twist.angular.z = -1.0;

        control_publisher_->publish(twist);
        twist.linear.x = 0.0;
        twist.angular.z = 0.0;
    }
}
```

```

    }
}

```

control\_publisher\_ : 퍼블리셔 선언, turtle1/cmd\_vel 토픽 pub

Twist타입의 twist 객체 생성, twist에 명령 담아서 퍼블리시

조건문으로 각각 기능 구현 (선속도, 각속도)

ex) 전진 (1.0, 0.0), 제자리 회전(0.0, 1.0), 자동차처럼 회전하면서 전진(1.0, 1.0)

&& (!'q') 해줌으로써 q 입력전까지는 제어하도록 구현

cli입력이므로 입력후 (0.0, 0.0)으로 다시 정리 초기화

- rclcpp/parameter\_client.hpp

: ros2에서 파라미터 관리를 위한 클라이언트 관련 툴 제공(터틀심 터틀 노드 ⇒ 배경색 설정)

'AsyncParametersClient' 클래스를 제공 - 비동기 방식

비동기 방식 : 요청을 보낸 후 결과를 기다리지 않고 다른 작업을 수행하게 해주는 방식

```

void TurtlesimCliNode::set_background_color() {
    std::string color_input;
    std::cout << "r,g,b 셋 중 하나를 입력하세요: ";
    std::cin >> color_input;

    int r = 0, g = 0, b = 0;
    if (color_input == "r")
        r = 255;
    else if (color_input == "g")
        g = 255;
    else if (color_input == "b")
        b = 255;

    auto parameter_client = std::make_shared<rclcpp::AsyncParam
    if (parameter_client->wait_for_service(std::chrono::second
        parameter_client->set_parameters({
            rclcpp::Parameter("background_r", r),
            rclcpp::Parameter("background_g", g),
            rclcpp::Parameter("background_b", b)
        }));
}

```

```

        // Empty 서비스 요청을 통해 배경 초기화
        auto request = std::make_shared<std_srvs::srv::Empty>();
        auto result = clear_client_->async_send_request(request);

        if (rclcpp::spin_until_future_complete(this->shared_future, result))
            std::cout << "배경 색 변경\n";
        }
    }
}

```

'AsyncParametersClient' 객체인 parameter\_client를 통해 'turtlesim'의 배경색 파라미터(background r,g,b) 설정

parameter\_client->set\_parameters() 함수를 사용해 배경색 값 설정 - 비동기방식  
여기서 Empty 서비스 요청을 통해 배경을 초기화 해줌

서비스 : ros2 토픽과는 다른 통신 방식으로 요청,응답 구조

```

#include "std_srvs/srv/empty.hpp"

rclcpp::Client<std_srvs::srv::Empty>::SharedPtr clear_client_

auto request = std::make_shared<std_srvs::srv::Empty::Request>();
auto result = clear_client_->async_send_request(request);

```

std\_srvs::srv::Empty::Request 객체를 생성, empty이므로 강요청 객체(트리거 역할 느낌)

clear\_client\_ 터틀심 노드에 화면초기화 서비스인 /clear 서비스 존재, 이걸 받아와서 화면 초기화

즉 변경된 파라미터 값(background\_r,g,b)을 설정 해줌

- turtlesim/srv/spawn.hpp, turtlesim/srv/kill.hpp

```

auto spawn_request = std::make_shared<turtlesim::srv::Spawn>();
spawn_request->x = 5.5;
spawn_request->y = 5.5;
spawn_request->theta = 0.0;
spawn_request->name = "turtle1";

```

```

if (spawn_client_->wait_for_service(std::chrono::seconds(1)))
    auto spawn_result = spawn_client_->async_send_request(spawn_request);
    if (rclcpp::spin_until_future_complete(this->shared_from_this(), spawn_result) == FutureReturnCode::SUCCESS)
        std::cout << "새로운 거북이가 생성되었습니다.\n";
}

```

spawn 서비스의 구조

request : 새로운 터틀의 초기 위치, 방향, 이름 지정 가능

response : 생성된 터틀의 이름을 반환

spawn\_request 객체 생성 설정(위치,방향,이름)해주고, spawn\_client\_ 를 통해 비동기 방식으로 Spawn 서비스 요청 전송 ⇒ spin\_until\_future\_complete로 요청이 완료될 때까지 대기

대기 해주는 이유는 비동기 방식임에도 대기가 없으면 요청의 결과가 오기도 전에 다음 코드가 실행됨.

사실상 비동기 방식의 이점은 못 살린다.

kill 서비스의 구조

request : 제거할 터틀의 이름을 지정

response : 별도로 없음

kill\_request 객체 생성

kill\_client\_→async\_send\_request(kill\_request);로 요청을 보내고,

spin\_until\_future\_complete로 요청이 완료될 때까지 대기

요청이 성공하면 메시지를 출력

- turtlesim/srv/set\_pen.hpp  
: 터틀심 거북이가 남기는 pen 색상과 두께 설정(서비스 타입)

```

auto set_pen_request = std::make_shared<turtlesim::srv::SetPen>(set_pen_request);
set_pen_request->r = 255;          // 빨간색 설정
set_pen_request->g = 0;           // 녹색 없음
set_pen_request->b = 0;           // 파란색 없음
set_pen_request->width = 3;       // 펜 두께 설정
set_pen_request->off = 0;        // 펜 활성화

// 비동기 방식으로 서비스 요청 전송
if (set_pen_client_->wait_for_service(std::chrono::seconds(1))

```

```

        auto result = set_pen_client_>async_send_request(set_pen_
        if (rc1cpp::spin_until_future_complete(this->shared_from_
            std::cout << "펜 설정이 완료되었습니다.\n";
    }
}

```

SetPen 서비스는 request와 response로 구성됩니다. 이 서비스는 터틀의 펜 속성을 설정하기 위해 사용된다. 여기에서 요청(Request)만 정의되며, 응답(Response)은 필요하지 않다

## Request

터틀의 펜 색상과 두께, 그리고 펜의 활성화 상태를 설정하는 필드를 포함합니다.

**필드 정의:**

uint8 r: 펜의 **빨간색** 성분 (0-255).

uint8 g: 펜의 **녹색** 성분 (0-255).

uint8 b: 펜의 **파란색** 성분 (0-255).

uint8 width: 펜의 **두께**를 나타냅니다.

uint8 off: 펜의 활성화 상태를 나타내는 필드입니다.

'0'이면 펜이 **활성화**되어 이동할 때 선을 남깁니다.

'1'이면 펜이 **비활성화**되어 이동해도 선이 남지 않습니다.

이 요청 구조를 통해 사용자는 터틀이 이동할 때 남기는 선의 색상과 두께를 설정할 수 있습니다.

set\_pen\_client\_>async\_send\_request(set\_pen\_request);를 사용하여 비동기 방식으로 요청을 전송

spin\_until\_future\_complete로 요청의 완료를 확인하고, 성공 시 메시지를 출력

## 기타

- 스마트 포인터

`std::shared_ptr`

역할 :

여러 개의 소유자가 동일한 자원에 접근할 수 있도록 하는 공유 포인터  
동일한 자원을 참조하는 포인터들이 모두 소멸될 때까지 자원이 유지

ros2 사용 :

노드를 생성하거나 클라이언트와 서비스 객체를 정의 관리, ROS2에서 퍼블리셔와 서브스 크라이버가 메시지를 주고받을 때 관리. 이는 메시지가 여러 곳에서 참조되더라도 참조가 모두 해제될 때까지 메모리가 안전하게 유지

## 피드백

- 통신 방식, 스마트 포인터, 비동기 방식 등 자세하게 공부하지 않고 완성시킨 점(gpt 사용이 원인)
- 거북이 설정 모드에서 랜덤으로 거북이 킬,스폰만 구현함
- 아이디어로 기존 거북이에 기존 거북이 이름 지정⇒킬⇒랜덤(turtle1)스폰 순서의 모드 하나,  
설정된 거북이 이름(example\_fix\_name) 스폰 하는 모드 나누었으면 구현 가능 했을 거 같다.

Default - rqt

File Plugins Running Perspectives Help

Node Graph

Nodes/Topics (active) / /

Group: 2 Namespaces Actions tf Images Highlight Fit

Hide: Dead sinks Leaf topics Debug tf Unreachable Params

Message Publisher

Topic: ter\_events Type: ParameterEvent Freq: 1 Hz

TurtleSim

Diagram showing the node graph:

```

graph LR
    A([/turtlesim_cli_node]) --> B[/turtle1/cmd_vel]
    B --> C([/turtlesim])
    subgraph /turtle1
        B
    end

```

Terminal output (Left):

```

jammay@jammay-Legion-5-16IRX9: ~/robot_ws
jammay@jammay-Legion-5-16IRX9:~/robot_ws$ ros2 run turtlesim turtlesim_node
원하는 모드를 선택하세요:
1: 조종모드
2: 배경색 설정 모드
3: 거북이 모양 설정 모드
4: Pen 설정 모드
5: 종료
1
w,a,s,d 를 눌러 조종하세요, q 입력시 종료
w
a
w
d
s
q
원하는 모드를 선택하세요:
1: 조종모드
2: 배경색 설정 모드
3: 거북이 모양 설정 모드
4: Pen 설정 모드
5: 종료

```

Terminal output (Right):

```

jammay@jammay-Legion-5-16IRX9: ~ 38x11
^C[INFO]: signal_handler(signum=2)
jammay@jammay-Legion-5-16IRX9:~$ source ~/robot_ws/install/setup.bash
jammay@jammay-Legion-5-16IRX9:~$ ros2 run turtlesim turtlesim_node
[INFO]: Starting turtlesim with node name /turtlesim
[INFO]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
jammay@jammay-Legion-5-16IRX9: ~ 38x11
jammay@jammay-Legion-5-16IRX9:~$ rqt

```







