Action 프로그래밍 - Python

해당 실습 자료는 한양대학교 Road Balance - ROS 2 for G Camp와 ROS 2 Documentation: Foxy, 표윤석, 임태훈 <ROS 2로 시작하는 로봇 프로그래밍> 루피페이퍼(2022) 를 참고하여 작성하였습니다.

> 이번 장에서는 `Server Node`와 `Client Node` 간의 메세지 통신 `Action`을 구현해볼 예정입니다.

<1. Action 인터페이스 패키지 만들기>

	msg 인터페이스	srv 인터페이스	action 인터페이스
확장자	*.msg	*.srv	*.action
데이터	토픽 데이터 (data)	서비스 요청 (request) --- 서비스 응답 (response)	액션 목표 (goal) --- 액션 결과 (result) --- 액션 피드백 (feedback)
형식	fieldtype1 fieldname1 fieldtype2 fieldname2 fieldtype3 fieldname3	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4 --- fieldtype5 fieldname5 fieldtype6 fieldname6

Creating custom msg and srv files — ROS 2 Documentation: Foxy documentation

:::2 https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Custom-ROS2-Interfaces.html#create-a-new-package

▼ 방법1: 직접 인터페이스 패키지 만들기

1. 패키지 생성 및 디렉토리 생성

- 인터페이스는 c++을 이용하여 생성합니다.(패키지 이름: custom_action_interface)

```
$ cd ~/ros2_ws/src
$ ros2 pkg create custom_action_interface --build-type amer
$ cd custom_action_interface
$ mkdir action
```

▼ 생성된 디렉토리 구조

```
.
├── action
├── include
│   └── my_first_ros_rclcpp_pkg
├── src
├── CMakeLists.txt
└── package.xml

4 directories, 2 files
```

2. Fibonacci.action 생성

- 경로: `~/ros2_ws/src/custom_action_interface/action`

```
$ cd ~/ros2_ws/src/custom_action_interface/action

## action 예제를 위한 Fibonacci.action 파일 생성
$ gedit Fibonacci.action
```

- `Fibonacci.action` **파일 내용**

```
# Goal
int32 order
---
# Result
int32[] sequence
---
```

```
# Feedback
int32[] partial_sequence
```

3. 관련 설정 파일 수정하기

- 경로: `~/ros2_ws/src/custom_action_interface`

1. package.xml 수정하기
    - 파일 수정하기 위해 이동하기

```
$ cd ~/ros2_ws/src/custom_action_interface
$ gedit package.xml
```

▼ **package.xml 내용**

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/pack
<package format="3">
  <name>custom_action_interface</name>
  <version>0.0.0</version>
  <description> ROS2 example for action interface </d
  <maintainer email="jetson@todo.todo">jetson</mainta
  <license>TODO: License declaration</license>
  <buildtool_depend>ament_cmake</buildtool_depend>

  <buildtool_depend>rosidl_default_generators</buildt
  <exec_depend>builtin_interfaces</exec_depend> ## 실
  <exec_depend>rosidl_default_runtime</exec_depend> #

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <member_of_group>rosidl_interface_packages</member_

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

- `rosidl_default_generators`
  - `rosidl_default_generators`는 ROS 2 인터페이스 정의 언어(IDL) 파일들로부터 코드를 생성하기 위한 기본 도구들을 포함하고 있습니다.
  - 현재 패키지 내에서 메시지, 서비스, 액션 인터페이스를 정의할 때 필요하며, 해당 인터페이스로부터 c++, PYTHON 등의 코드를 생성하는 데 사용됩니다.
- `builtin_interfaces`
  - ROS 2에서 기본 제공하는 메시지 타입들을 포함하고 있는 패키지 입니다.
  - 시간스탬프나 기본적인 메시지 타입 같은 공통적으로 사용되는 인터페이스를 제공하고 있습니다.
- `rosidl_default_runtime`
  - RoS2 IDL로 생성된 인터페이스 코드가 런타임에 필요함을 의미합니다.
  - ROS 2 런타임 중에 메시지, 서비스, 액션 인터페이스와 상호작용하는 데 필요한 라이브러리와 의존성을 포함하고 있습니다.

2. CmakeLists.txt 수정하기 위해 이동

```
$ cd ~/ros2_ws/src/custom_action_interface
$ gedit CMakeLists.txt
```

▼ **CMakeLists.txt 내용**

```
cmake_minimum_required(VERSION 3.5)
project(custom_action_interface)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID
  add_compile_options(-Wall -Wextra -Wpedantic)
```

```
endif()

##================================================
##  Find and load build settings from external packag
##================================================
find_package(ament_cmake REQUIRED)
find_package(builtin_interfaces REQUIRED) ## 추가된 부분
find_package(rosidl_default_generators REQUIRED)## 추
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)
##================================================



##================================================
##  Declare ROS messages, Fibonacci action
##================================================
set(action_files
 "action/Fibonacci.action" ## 정의한 action 파일 경로
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
##================================================



##================================================
## Macro for ament package
##================================================
ament_export_dependencies(rosidl_default_runtime)
ament_package()
##================================================
```

4. Build하기

- 경로: `~/ros2_ws`

```
$ source /opt/ros/foxy/setup.bash
$ cd ~/ros2_ws/
$ colcon build --symlink-install --packages-select custom_a
```

- 확인하기

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

$ ros2 interface show custom_action_interface/action/Fibona
##================== 출력 결과 ==================
# Goal
int32 order
---
# Result
int32[] sequence
---
# Feedback
int32[] partial_sequence
##================================================
```

▼ 방법2: 깃클론해서 패키지 빌드하기

```
$ cd ~/ros2_ws/src
$ git clone https://github.com/2seung0708/ros2_example.git
$ mv ./ros2_example/src/custom_action_interface ~/ros2_ws/s
```

<2. Action 프로그래밍>

Action 예제 작성

1. 패키지 생성

- 작성한 워크스페이스인 `ros2_ws/src` 디렉토리에 이동하신 다음 새로운 패키지를 생성합니다.

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

- 새로운 패키지 이름은 `fibonacci_action` 으로 동명의 디렉토리에 패키지 기본 구성이 생성된 것을 확인 할 수 있을 겁니다.
- 추가로 `--dependencies` 인수를 통해 패키지 환경 설정 파일 `package.xml` 에 필요한 종속성 패키지인 `rclpy` , `custom_action_interface`가 추가됩니다.

Action Server Node 작성

- Action 기능: `Goal Response` , `Feedback` , `Result Response`를 구현
- Action Server Node의 파이썬 스크립트는 `~/ros2_ws/src/fibonacci_action/fibonacci_action /` 폴더에 `fibonacci_action_server.py`라는 이름으로 소스 코드 파일을 저장하시면 됩니다.

```
$ cd ~/ros2_ws/src/fibonacci_action/fibonacci_action
$ gedit fibonacci_action_server.py
```

- **fibonacci_action_server.py** 코드

```
import rclpy # Python ROS2 프로그래밍을 위한 rclpy
from rclpy.node import Node# rclpy 의 Node 클래스
from rclpy.action import ActionServer, GoalResponse # ActionS

from custom_action_interface.action import Fibonacci
# 사전에 정의한 인터페이스 custom_action_interface import

import time

class FibonacciActionServer(Node):# Node 클래스를 상속
    def __init__(self):
        super().__init__('fibonacci_action_server')
                # 부모 클래스(Node)의 생성자를 호출하고 이름을 fibona
```

```
        ##==================== Action Server 정의 ============
        self.action_server = ActionServer(
            self, #  실행 노드
            Fibonacci, # 메시지 타입
            'fibonacci',# 액션 이름(client도 동일하게 받아야함)
            ##### 콜백 함수#####
                        # Goal Request가 오면, 우선 goal_callbac
                        # execute_callback으로 넘어가게 됩니다.
                        ####################
            self.execute_callback,
            goal_callback=self.goal_callback)
        ##=====================================================

        self.get_logger().info("=== Fibonacci Action Server S

    ##================ Goal Request 이후의 콜백함수 ============
    def execute_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')

        feedback_msg = Fibonacci.Feedback()
        feedback_msg.partial_sequence = [0, 1]

        for i in range(1, goal_handle.request.order):
            feedback_msg.partial_sequence.append(
                feedback_msg.partial_sequence[i] + feedback_m
            )

            print(f"Feedback: {feedback_msg.partial_sequence}
            goal_handle.publish_feedback(feedback_msg) # 피드백
            time.sleep(1)

        goal_handle.succeed() # 액션 client에 현재 액션 상태(성공)
        self.get_logger().warn("==== Succeed ====")

        result = Fibonacci.Result()# Result로 선언
        result.sequence = feedback_msg.partial_sequence # fee
```

```
        return result # 결과값을 반환
    ##=======================================================

    ##=============== Goal Request시  콜백함수 ===============
    def goal_callback(self, goal_request):
        """Accept or reject a client request to begin an acti
        self.get_logger().info('Received goal request')

        return GoalResponse.ACCEPT
    ##=======================================================


def main(args=None):
    rclpy.init(args=args) # 초기화
    node = FibonacciActionServer() # FibonacciActionServer를 r
    try:
        rclpy.spin(node) # rclpy에게 이 Node를 반복해서 실행 (=spi
    except KeyboardInterrupt: # `Ctrl + c`가 동작했을 때
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


if __name__ == '__main__':
    main()
```

- class 내부
  - ▼ `def __init__(self)`

```
class FibonacciActionServer(Node):# Node 클래스를 상속
    def __init__(self):
        super().__init__('fibonacci_action_server') # 부

        ##==================== Action Server 정의 ======
        self.action_server = ActionServer(
```

```
            self, #  실행 노드
            Fibonacci, # 메시지 타입
            'fibonacci',# 액션 이름(client도 동일하게 받아야

                        ##----- 콜백 함수 -----
                        # Goal Response가 오면, 우선 goal_
                        # execute_callback으로 넘어가게 됨
                        ##--------------------
            self.execute_callback,
            goal_callback=self.goal_callback)
        ##=============================================

        self.get_logger().info("=== Fibonacci Action Se
```

▼ `def execute_callback(self, goal_handle)`

```
    ##==================== goal_callback 이후의 콜백함수 ==
    ##  =>  Feedback과 Result를 처리
    ## goal_handle: rclpy.action 모듈의 ServerGoalHandle
    ##              execute, succeed, canceled 등 액션 상
    ##              & 피드백을 publish가능
    ##=============================================
    def execute_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')# 터미

                # Feedback action을 준비
        feedback_msg = Fibonacci.Feedback() # Feedback으
        feedback_msg.partial_sequence = [0, 1]

                # Request 숫자만큼의 피보나치 수열을 계산
        for i in range(1, goal_handle.request.order):

                        # 피보나치 로직
            feedback_msg.partial_sequence.append( # 연산
                feedback_msg.partial_sequence[i] + feed
            )
```

```
                        # feedback publish가 이루어지는 부
            print(f"Feedback: {feedback_msg.partial_seq
            goal_handle.publish_feedback(feedback_msg)
            time.sleep(1)

        goal_handle.succeed() # 액션 client에 현재 액션 상태
        self.get_logger().warn("==== Succeed ====")

                # 모든 계산을 마치고, result를 되돌려주는 부분
        result = Fibonacci.Result()# Result로 선언
        result.sequence = feedback_msg.partial_sequence

        return result # 결과값을 반환
    ##=================================================
```

▼ `def goal_callback(self, goal_request)`

```
    ##==================== goal_Request시 사용되는 콜백함수
    ##=================================================
        # Goal Request 시 가장 처음 진입하게 되는 callback입니
    def goal_callback(self, goal_request):
        """Accept or reject a client request to begin a
        self.get_logger().info('Received goal request')

                # 도저히 불가능한 Request가 왔다면, 여기에서 판
                # GoalResponse의 REJECT=1, ACCEPT=2로 상태
                # 아래 ACCEPT => REJECT로 바꾼 뒤, 다시 실행
        return GoalResponse.ACCEPT
    ##=================================================
```

- main 부분

```
def main(args=None):
    rclpy.init(args=args) # 초기화
    node = FibonacciActionServer() # FibonacciActionServer
```

```
    try:
        rclpy.spin(node) # rclpy에게 이 Node를 반복해서 실행 (=
    except KeyboardInterrupt: # `Ctrl + c`가 동작했을 때
        node.get_logger().info('Keyboard Interrupt (SIGINT
    finally:
        node.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


if __name__ == '__main__':
    main()
```

▼ 전체 코드 (with 주석)

```
import rclpy # Python ROS2 프로그래밍을 위한 rclpy
from rclpy.node import Node# rclpy 의 Node 클래스
from rclpy.action import ActionServer, GoalResponse # A

from custom_action_interface.action import Fibonacci #

class FibonacciActionServer(Node):# Node 클래스를 상속
    def __init__(self):
        super().__init__('fibonacci_action_server') # 부
        qos_profile = QoSProfile(depth=10) # 통신상태가 원

        ##==================== Action Server 정의 ======
        self._action_server = ActionServer(
            self, #  실행 노드
            Fibonacci, # 메시지 타입
            'fibonacci',# 액션 이름(client도 동일하게 받아야

                        ##### 콜백 함수#####
                        # Goal Response가 오면, 우선 goal_
                        # execute_callback으로 넘어가게 됩
                        ####################
            self.execute_callback,
            goal_callback=self.goal_callback)
        ##============================================
```

```
        self.get_logger().info("=== Fibonacci Action Se


    ##==================== goal_callback 이후의 콜백함수 ==
    ##  =>  Feedback과 Result를 처리
    ## goal_handle: rclpy.action 모듈의 ServerGoalHandle
    ##              execute, succeed, canceled 등 액션 상
    ##              & 피드백을 publish가능
    ##=================================================
    def execute_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')# 터미

                # Feedback action을 준비
        feedback_msg = Fibonacci.Feedback() # Feedback으
        feedback_msg.partial_sequence = [0, 1]

                # Request 숫자만큼의 피보나치 수열을 계산
        for i in range(1, goal_handle.request.order):

                        # 피보나치 로직
            feedback_msg.partial_sequence.append( # 연산
                feedback_msg.partial_sequence[i] + feed
            )

                        # feedback publish가 이루어지는 부
            print(f"Feedback: {feedback_msg.partial_seq
            goal_handle.publish_feedback(feedback_msg) =
            time.sleep(1)

        goal_handle.succeed() # 액션 client에 현재 액션 상태
        self.get_logger().warn("==== Succeed ====")

                # 모든 계산을 마치고, result를 되돌려주는 부분
        result = Fibonacci.Result()# Result로 선언
        result.sequence = feedback_msg.partial_sequence

        return result # 결과값을 반환
```

```
    ##=================================================

    ##==================== goal_Request시 사용되는 콜백함수
    ##  =>  Feedback과 Result를 처리
    ## goal_handle: rclpy.action 모듈의 ServerGoalHandle
    ##              execute, succeed, canceled 등 액션 상
    ##              & 피드백을 publish가능
    ##=================================================
        # Goal Request 시 가장 처음 진입하게 되는 callback입
    def goal_callback(self, goal_request):
        """Accept or reject a client request to begin a
        self.get_logger().info('Received goal request')

                # 도저히 불가능한 Request가 왔다면, 여기에서 판
                # 아래 ACCEPT => REJECT로 바꾼 뒤, 다시 실행
        return GoalResponse.ACCEPT
    ##=================================================


def main(args=None):
    rclpy.init(args=args) # 초기화
    node = FibonacciActionServer() # FibonacciActionSer
    try:
        rclpy.spin(node) # rclpy에게 이 Node를 반복해서 실행
    except KeyboardInterrupt: # `Ctrl + c`가 동작했을 때
        node.get_logger().info('Keyboard Interrupt (SIG
    finally:
        node.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


if __name__ == '__main__':
    main()
```

Action Client Node작성

- Action Client Node의 파이썬 스크립트는 `~/ros2_ws/src/fibonacci_action/fibonacci_action /` 폴더에 `fibonacci_action_client.py.py` 라는 이름으로 소스 코드 파일을 저장하시면 됩니다.

```
$ cd ~/ros2_ws/src/fibonacci_action/fibonacci_action
$ gedit fibonacci_action_client.py
```

- **fibonacci_action_client.py** 코드

```
import rclpy # Python ROS2 프로그래밍을 위한 rclpy
from rclpy.node import Node# rclpy 의 Node 클래스
from rclpy.action import ActionClient, GoalResponse
# ActionServer와 GoalResponse import

from custom_action_interface.action import Fibonacci
# 사전에 정의한 인터페이스 custom_action_interface import

class FibonacciActionClient(Node):# Node 클래스를 상속
    def __init__(self):
        super().__init__('fibonacci_action_client')
                # 부모 클래스(Node)의 생성자를 호출하고 이름을 fib

        ##==================== Action Client 정의 =========
        self.action_client= ActionClient(
            self, #  실행 노드
            Fibonacci, # 메시지 타입
            'fibonacci')# 액션 이름(action server에서 정한 이름
        ##================================================

        self.get_logger().info("=== Fibonacci Action Clien

    ##=============== send Goal ===============
    def send_goal(self, order):
        goal_msg = Fibonacci.Goal()
        goal_msg.order = order
```

```
        # 10초간 server를 기다리다가 응답이 없으면 에러를 출력
        if self.action_client.wait_for_server(10) is False
          self.get_logger().error("Server Not exists")

        # goal request가 제대로 보내졌는지 알기 위해 future가 사용
        # 더불어, feedback_callback을 묶어 feedback 발생 시 해당
        self._send_goal_future = self.action_client.send_g
            goal_msg, feedback_callback=self.feedback_call
        )


        # server가 존재한다면, Goal Request의 성공 유무,
        # 최종 Result에 대한 callback도 필요
        self._send_goal_future.add_done_callback(self.goal_
    ##=====================================================

    ##=============== feedback을 받아오는 함수 ==============
    def feedback_callback(self, feedback_msg):#send_goal에
        feedback = feedback_msg.feedback
        print(f"Received feedback: {feedback.partial_seque
    ##=====================================================


    ##=============== Goal Request에 대한 응답으로 실행되는 cal
    def goal_response_callback(self, future):
        goal_handle = future.result()

        # Goal type에 따라 성공 유무를 판단합니다.
        if not goal_handle.accepted:
            self.get_logger().info("Goal rejected")
            return

        self.get_logger().info("Goal accepted")

        # 최종 Result 데이터를 다룰 callback을 연동합니다.
        self._get_result_future = goal_handle.get_result_a
        self._get_result_future.add_done_callback(self.get_
    ##=====================================================
```

```
    ##===============  Result callback 함수 ===============
    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().warn(f"Action Done !! Result: {r
        rclpy.shutdown()
    ##=====================================================

def main(args=None):
    rclpy.init(args=args)

    fibonacci_action_client = FibonacciActionClient()

    # Client Node 생성 이후 직접 send_goal을 해줍니다. (Service
        # Goal Request에 대한 future를 반환받음
    future = fibonacci_action_client.send_goal(5)

    rclpy.spin(fibonacci_action_client)


if __name__ == '__main__':
    main()
```

- class 내부

  ▼ `def __init__(self)`

```
class FibonacciActionClient(Node):# Node 클래스를 상속
    def __init__(self):
        super().__init__('fibonacci_action_client') # 누

        ##==================== Action Server 정의 ======
        self.action_client= ActionClient(
            self, #  실행 노드
            Fibonacci, # 메시지 타입
            'fibonacci')# 액션 이름(action server에서 정한
        ##=============================================
```

```
        self.get_logger().info("=== Fibonacci Action Cl
```

▼ `def send_goal(self, order)`

```
    ##=============== send Gaol ===============
    def send_goal(self, order):
        goal_msg = Fibonacci.Goal()
        goal_msg.order = order

        # 10초간 server를 기다리다가 응답이 없으면 에러를 출력
        if self.action_client.wait_for_server(10) is Fa
          self.get_logger().error("Server Not exists")

        # goal request가 제대로 보내졌는지 알기 위해 future가
        # 더불어, feedback_callback을 묶어 feedback 발생 시
        self._send_goal_future = self.action_client.sen
            goal_msg, feedback_callback=self.feedback_c
        )


        # server가 존재한다면, Goal Request의 성공 유무,
        # 최종 Result에 대한 callback도 필요
        self._send_goal_future.add_done_callback(self.g
    ##=================================================
```

▼ `def feedback_callback (self, feedback_msg)`

```
    ##=============== feedback을 받아오는 함수 ==========
    def feedback_callback(self, feedback_msg):#send_goa
        feedback = feedback_msg.feedback
        print(f"Received feedback: {feedback.partial_se
    ##=================================================
```

▼ `def goal_response_callback(self, future)`

```
    ##=============== Goal Request에 대한 응답으로 실행되는
    def goal_response_callback(self, future):
```

```
        goal_handle = future.result()

        # Goal type에 따라 성공 유무를 판단합니다.
        if not goal_handle.accepted:
            self.get_logger().info("Goal rejected")
            return

        self.get_logger().info("Goal accepted")

        # 최종 Result 데이터를 다룰 callback을 연동합니다.
        self._get_result_future = goal_handle.get_resul
        self._get_result_future.add_done_callback(self.
    ##=================================================
```

▼ `def get_result_callback(self, future)`

```
    ##===============  Result callback 함수 ============
    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().warn(f"Action Done !! Result:
        rclpy.shutdown()
    ##=================================================
```

📌 함수에 대한 실행 시점

- `send_goal` : goal send 시점에 **`feedback_callback`**이 묶이며 send_goal 이 완료되는 시점에 **`goal_response_callback`** 으로 이동합니다.

```
self._send_goal_future = self.action_client.send_goal_asyn
    goal_msg, feedback_callback=self.feedback_callback
)

self._send_goal_future.add_done_callback(self.goal_respons
```

- `goal_response_callback` : **`get_result_async`** 이 완료되는 시점에 **`get_result_callback`** 으로 이동합니다.

```
self._get_result_future = goal_handle.get_result_async()
self._get_result_future.add_done_callback(self.get_result_
```

- `feedback_callback` : 지속적으로 feedback을 출력합니다.
- `get_result_callback` : 최종 마지막에 실행되는 함수로 Result를 출력합니다.

- main 부분

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

    fibonacci_action_client = FibonacciActionClient()

    # Client Node 생성 이후 직접 send_goal을 해줍니다. (Service
        # Goal Request에 대한 future를 반환받음
    future = fibonacci_action_client.send_goal(5)

    rclpy.spin(fibonacci_action_client)


if __name__ == '__main__':
    main()
```

Add an entry point

- `ros2 run` 커맨드를 통해 작성한 service node 실행시키기 위해서는 `setup.py` 속의 `entry_points` 구역에 아래의 내용을 추가해야합니다.
- 경로로 이동 및 수정
-

```
$ cd ~/ros2_ws/src/fibonacci_action
$ gedit setup.py
```

```
entry_points={
    'console_scripts': [
        'fibonacci_action_server= fibonacci_action.fibonac
        'fibonacci_action_client= fibonacci_action.fibonac
    ],
},
```

Build and run

- 이제 패키지를 build하고 실행해보도록 하겠습니다
- 실행 과정(`ros2 run` 실행 전에 수행해야 하는 코드)
  1. 먼저 실행을 위한 경로로 이동하여 ROS2 실행 환경을 실행합니다.

     ```
     $ cd ~/ros2_ws
     $ source /opt/ros/foxy/setup.bash
     ```

  2. 그 다음에 빌드를 수행합니다.

     ```
     $ colcon build --symlink-install --packages-select fibo
     Starting >>> fibonacci_action
     Finished <<< fibonacci_action [0.94s]

     Summary: 1 package finished [1.12s]
     ```

  3. 마지막으로 로컬에 위치한 패키지의 환경 변수를 설정하기 위해서 setup file을 source 합니다!

     ```
     $ source install/local_setup.bash
     ```

💡 install 디렉토리에 위치한 `local_setup`과 `setup`은 뭐가 다른 걸까요?

- `local_setup` 은 내가 설치한 패키지의 환경 변수를 source 하기 위한 파일!
- `setup` 은 `/opt/ros/foxy` 와 같이 글로벌하게 사용되는 환경 변수도 source 합니다.
  즉,
  `source /opt/ros/foxy/setup.bash & source install/setup.bash` 과 동일합니다.

실행

```
# 터미널1

$ ros2 run fibonacci_action fibonacci_action_server
[INFO] [1683486665.920066930] [fibonacci_action_server]: =
[INFO] [1683486673.420139016] [fibonacci_action_server]: R
[INFO] [1683486673.421320117] [fibonacci_action_server]: E
Feedback: array('i', [0, 1, 1])
Feedback: array('i', [0, 1, 1, 2])
Feedback: array('i', [0, 1, 1, 2, 3])
Feedback: array('i', [0, 1, 1, 2, 3, 5])
[WARN] [1683486677.428286037] [fibonacci_action_server]: =
```

```
# 터미널 2

$ ros2 run fibonacci_action fibonacci_action_client
[INFO] [1683486673.418898780] [fibonacci_action_client]: =
future?:<rclpy.task.Future object at 0x7ff002c90b20>
[INFO] [1683486673.421179634] [fibonacci_action_client]: G
Received feedback: array('i', [0, 1, 1])
Received feedback: array('i', [0, 1, 1, 2])
Received feedback: array('i', [0, 1, 1, 2, 3])
```

```
Received feedback: array('i', [0, 1, 1, 2, 3, 5])
[WARN] [1683486677.431033816] [fibonacci_action_client]: A
```