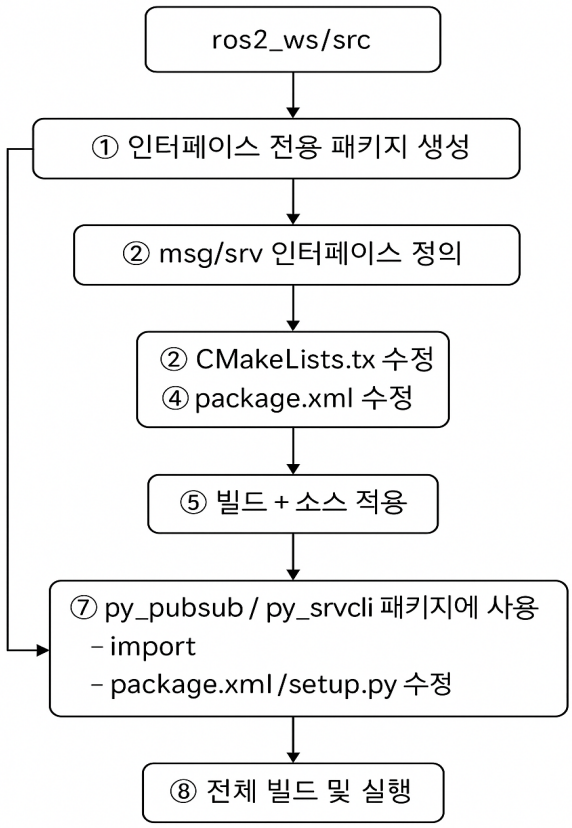
스터디 15 주차 홍송은

ROS 2 Custom Interface 생성 및 활용

사전 준비 사항

- ROS 2 Humble 버전이 설치
- ros2_ws 워크스페이스 구성
- ros2_ws/src 하위에 py_pubsub, py_srvcli 등의 패키지 구성

전체 흐름도



1. 인터페이스 전용 패키지 생성

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake --license Apache-2.0 tutorial_interfaces
```

2. 인터페이스 파일 작성

```
cd ~/ros2_ws/src/tutorial_interfaces
mkdir msg srv
```

msg/Num.msg:

```
int64 num
```

msg/Sphere.msg:

```
geometry_msgs/Point center
float64 radius
```

srv/AddThreeInts.srv:

```
int64 a
int64 b
int64 c
---
int64 sum
```

3. CMakeLists.txt 수정

```
find_package(ament_cmake REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "msg/Sphere.msg"
  "srv/AddThreeInts.srv"
  DEPENDENCIES geometry_msgs
)
ament_package()
```

4. package.xml 수정

<buildtool_depend>ament_cmake</buildtool_depend>
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<depend>geometry_msgs</depend>
<member_of_group>rosidl_interface_packages</member_of_group>

5. 빌드 및 환경 설정

cd ~/ros2_ws
colcon build --packages-select tutorial_interfaces
source install/setup.bash

6. 생성된 인터페이스 확인

ros2 interface show tutorial_interfaces/msg/Num
ros2 interface show tutorial_interfaces/msg/Sphere
ros2 interface show tutorial_interfaces/srv/AddThreeInts

7. Python 패키지에서 활용

- py_pubsub

ros2_ws/src/py_pubsub/py_pubsub/talker.py

ros2_ws/src/py_pubsub/py_pubsub/listener.py:

from tutorial_interfaces.msg import Num

ros2_ws/src/py_pubsub/package.xml:

<exec_depend>tutorial_interfaces</exec_depend>

ros2_ws/src/py_pubsub/setup.py:

```
entry_points={
  'console_scripts': [
    'talker = py_pubsub.publisher_member_function:main',
```

```
        'listener = py_pubsub.subscriber_member_function:main',
    ],
}
```

- py_srvcli

ros2_ws/src/py_srvcli/py_srvcli/service.py

ros2_ws/src/py_srvcli/py_srvcli/client.py:

```
from tutorial_interfaces.srv import AddThreeInts
```

ros2_ws/src/py_srvcli/package.xml:

```
<exec_depend>tutorial_interfaces</exec_depend>
```

ros2_ws/src/py_srvcli/setup.py:

```
entry_points={
    'console_scripts': [
        'client = py_srvcli.client:main',
        'service = py_srvcli.service:main',
    ],
}
```

8. 전체 빌드 및 실행

cd ~/ros2_ws
colcon build --packages-select py_pubsub py_srvcli
source install/setup.bash

pub/sub 테스트

ros2 run py_pubsub talker
ros2 run py_pubsub listener

```
rokey@rokey-550XBE-350XBE:~/ros2_ws$ ros2 run py_pubsub talker
[INFO] [1750749966.604070357] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [1750749967.059504530] [minimal_publisher]: Publishing: "Hello World: 1"
[INFO] [1750749967.559518848] [minimal_publisher]: Publishing: "Hello World: 2"
[INFO] [1750749968.060667243] [minimal_publisher]: Publishing: "Hello World: 3"
[INFO] [1750749968.560416542] [minimal_publisher]: Publishing: "Hello World: 4"
[INFO] [1750749969.059002395] [minimal_publisher]: Publishing: "Hello World: 5"
[INFO] [1750749969.559863094] [minimal_publisher]: Publishing: "Hello World: 6"
[INFO] [1750749970.059005356] [minimal_publisher]: Publishing: "Hello World: 7"
```

```
rokey@rokey-550XBE-350XBE:~/ros2_ws$ ros2 run py_pubsub listener
[INFO] [1750749986.091647361] [minimal_subscriber]: I heard: "Hello World: 39"
[INFO] [1750749986.561525142] [minimal_subscriber]: I heard: "Hello World: 40"
[INFO] [1750749987.059659902] [minimal_subscriber]: I heard: "Hello World: 41"
[INFO] [1750749987.560364797] [minimal_subscriber]: I heard: "Hello World: 42"
[INFO] [1750749988.060326759] [minimal_subscriber]: I heard: "Hello World: 43"
[INFO] [1750749988.561033198] [minimal_subscriber]: I heard: "Hello World: 44"
[INFO] [1750749989.060006646] [minimal_subscriber]: I heard: "Hello World: 45"
```

service/client 테스트

ros2 run py_srvcli service
ros2 run py_srvcli client 2 3 4

```
rokey@rokey-550XBE-350XBE:~/ros2_ws$ ros2 run py_srvcli service
[INFO] [1750750424.208492033] [minimal_service]: Incoming request
a: 2 b: 3 c: 4
```

```
rokey@rokey-550XBE-350XBE:~/ros2_ws$ ros2 run py_srvcli client 2 3 4
[INFO] [1750750424.223353710] [minimal_client_async]: Result of add_three_ints: for 2 + 3 + 4 = 9
rokey@rokey-550XBE-350XBE:~/ros2_ws$
```

Q1. .msg vs .srv 차이

- .msg: 단방향 메시지. pub/sub 에서 사용. 구조체 형태
- .srv: 양방향 요청-응답 구조. ---로 구분
- 내부 처리: .idl → .py, .hpp 자동 생성, rosidl_generate_interfaces()로 통합

Q2. Python 패키지에서 setup.py 구성

```
entry_points={
  'console_scripts': [
    'talker = py_pubsub.publisher_member_function:main',
    'listener = py_pubsub.subscriber_member_function:main',
    'client = py_srvcli.client:main',
    'service = py_srvcli.service:main',
  ],
}
```

- tutorial_interfaces 는 package.xml 의 <exec_depend>로 명시
- install_requires=['setuptools'] 포함 필수

Q3. 스터디 중 질문 피드백

MinimalClientAsync 에서 "Async"라고 부르는 이유는?

ros2_ws/src/py_srvcli/py_srvcli/client_member_function.py

```
class MinimalClientAsync(Node):


  def __init__(self):
    super().__init__('minimal_client_async')
    self.cli = self.create_client(AddThreeInts, 'add_three_ints')
    while not self.cli.wait_for_service(timeout_sec=1.0):
```

```
            self.get_logger().info('service not available, waiting again...')
        self.req = AddThreeInts.Request()

    def send_request(self):
        self.req.a = int(sys.argv[1])
        self.req.b = int(sys.argv[2])
        self.req.c = int(sys.argv[3])
        self.future = self.cli.call_async(self.req)


def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    minimal_client.send_request()

    while rclpy.ok():
        rclpy.spin_once(minimal_client)
        if minimal_client.future.done():
            try:
                response = minimal_client.future.result()
            except Exception as e:
                minimal_client.get_logger().info(
                    'Service call failed %r' % (e,))
```

```
        else:
            minimal_client.get_logger().info(
                'Result of add_three_ints: for %d + %d + %d = %d' %
                (minimal_client.req.a, minimal_client.req.b, minimal_client.req.c,
response.sum))
        break

    minimal_client.destroy_node()
    rclpy.shutdown()

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

- 서비스는 본래 동기 아닌가요?
  - Service Server 는 항상 동기적으로 동작
  - 요청이 오면 바로 처리하고 응답을 반환하는 구조
- 그런데 왜 Client 쪽은 Async 라고 하나요?
  - 클라이언트는 요청을 보내는 방식을 선택 가능
    ➔ call() → 동기 호출: 응답 올 때까지 코드가 멈춤
    ➔ call_async() → 비동기 호출: 요청만 보내고 계속 다음 코드 실행
- 이 코드에서 Async 라고 부른 이유는?
  - call_async()를 사용했기 때문이며, 비동기로 서비스 요청을 보내고, 결과는 future.result()로 나중에 받음

```
self.future = self.cli.call_async(self.req)
```

- 응답을 기다리는 방법?
  - rclpy.spin_once() 사용

```
while rclpy.ok():
```

```
rclpy.spin_once(minimal_client)
if minimal_client.future.done():
  ...
```

- ROS 2 이벤트 루프를 한 번만 실행하고 빠져나오는 방식
- 이 구조 덕분에 다른 작업과 병행하거나 빠르게 반복 체크 가능

핵심 요약

- 인터페이스는 독립 패키지(ament_cmake)로 정의
- msg/srv → CMakeLists.txt + package.xml 설정 필요
- Python 사용 시 import, setup.py, package.xml 조정 필요
- 빌드 후 source install/setup.bash 실행 필수