ROS Programming

ROS 프로그래밍

05 ROS2 기본 프로그래밍

목차 ROS 프로그래밍

1. ROS2 프로그래밍 규칙

코드스타일 가이드

- 협업 프로그래밍 작업시에는 일관된 규칙을 만들고 이를 준수하여 자동화 툴로 자가 검토
- 초기에는 귀찮고 버거울 수 있으나 빈번히 생기는 개발자의 부가적인 선택을 줄여줌
- 다른 협업 개발자 및 이용자의 코드 이해도를 높이며 상호 간의 코드 리뷰가 용이
- 특정 기능으로 인해 생길 수 있는 오류와 다양한 이슈를 피할 수 있음

기본 이름 규칙

- `ROS` 인터페이스 류의 파일은 /msg 및 /srv 또는 /action 에 폴더에 위치시키며 인터페이스 파일명은 `CamelCased` 규칙을 따름
- *.msg 및 *.srv 또는 *.action는 *.h(pp) 변환 후 인터페이스 타입으로 구조체 및 타입으로 사용되기 때문
  - CamelCased
  - snake_case
  - ALL_CAPITALS

코드스타일 가이드

- 그 이외에 특정 목적에 의해 만들어지는 하기 파일 이름은 예외적으로 대소문자 규칙을 따르지 않고 하기와 같이 고유의 이름을 사용
  - package.xml
  - CMakeLists.txt
  - README.md
  - LICENSE
  - CHANGELOG.rst
  - .gitignore
  - .travis.yml
  - *.repos

C++ Style

- Google C++ Style Guide를 사용하고 있으며 ROS의 특성에 따라 일부를 수정하여 사용
- 기본 규칙 : C++14 Standard를 준수
- 라인 길이 : 최대 100 문자
- 이름 규칙 : `CamelCased`, `snake_case`, `ALL_CAPITALS` 만을 사용
  - CamelCased : 타입, 클래스, 구조체, 열거형
  - snake_case : 파일, 패키지, 인터페이스, 네임스페이스, 변수, 함수, 메소드
  - ALL_CAPITALS : 상수, 매크로

C++ Style

- 소스 파일은 `.cpp` 확장자를 사용
- 헤더 파일은 `.hpp` 확장자를 사용
- 전역변수(global variable)는 사용이 피치 못한 경우에는 `g_` 접두어를 붙임
- 클래스 멤버 변수(class member variable)는 마지막에 밑줄(`_`)을 붙임

• 공백 문자 대 탭(Spaces vs. Tabs)

- 기본 들여쓰기(indent)는 공백 문자(space) `2개`를 사용한다. (탭(tab)문자 사용 금지)
- `Class`의 `public:`, `protected:`, `private:`은 들여쓰기를 사용 X

C++ Style

- 괄호(Brace)
  - 모든 if, else, do, while, for 구문에 괄호를 사용
  - 괄호 및 공백 사용은 아래 예제를 참고

C++ Style

- 좌 : 올바른 사용법, 우 : 잘못된 사용법

```cpp
int main(int argc, char **argv)
{
  if (condition) {
    return 0;
  } else {
    return 1;
  }
}

if (this && that || both) {
  ...
}

// Long condition; open brace
if (
  this && that || both && this && that || both && this && that || both && this && that)
{
  ...
}

// Short function call
call_func(foo, bar);

// Long function call; wrap at the open parenthesis
call_func(
  foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar,
  foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar, foo, bar);

// Very long function argument; separate it for readability
call_func(
  bang,
  fooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo,
  bar, bat);

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
  Type par_name1,  // 2 space indent
  Type par_name2,
  Type par_name3)
{
  DoSomething();  // 2 space indent
  ...
}

MyClass::MyClass(int var)
: some_var_(var),
  some_other_var_(var + 1)
{
  ...
  DoSomething();
```

```cpp
```
int main(int argc, char **argv) {
  return 0;
}

if (this &&
    that ||
    both) {
  ...
}

ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
    Type par_name1, // 4 space indent
    Type par_name2,
    Type par_name3) {
  DoSomething(); // 2 space indent
  ...
}

MyClass::MyClass(int var)
    : some_var_(var),   // 4 space indent
      some_other_var_(var + 1) { // lined up
  ...
  DoSomething();
  ...
}
```
```

C++ Style

- 주석(Comments)
  - 문서 주석에는 `/** */`을 사용, 구현 주석에는 `//`을 사용
- 린터(Linters)
  - C++ 코드 스타일의 자동 오류 검출을 위하여 ament_cpplint, ament_uncrustify를 사용, 정적 코드 분석이 필요한 경우 ament_cppcheck 사용

코드스타일 가이드

- 기타
  - Boost 라이브러리의 사용은 가능한 피하고 어쩔 수 없을 경우에만 사용
  - 포인트 구문은 `char * c;`처럼 사용(`char* c;` 이나 `char *c;` 처럼 사용하지 않는다.)
  - 중첩 템플릿은 `set<list<string>>`처럼 사용(`set<list<string> >` 또는 `set< list<string> >`처럼 사용하지 않는다.)

Python Style

- ROS 2 Developer Guide 및 ROS 2 Code style 에서 다루고 있는 Python 코드 스타일은 Python Enhancement Proposals (PEPs) PEP 8를 준수
- 기본 규칙 : Python3(Python 3.50이상) 사용
- 라인 길이 : 최대 100 문자
- 이름 규칙 : `CamelCased`, `snake_case`, `ALL_CAPITALS` 만을 사용
  - CamelCased : 타입, 클래스
  - snake_case : 파일, 패키지, 인터페이스, 네임스페이스, 변수, 함수, 메소드
  - ALL_CAPITALS : 상수

Python Style

- 공백 문자 대 탭(Spaces vs Tabs)
  - 기본 들여쓰기(indent)는 공백 문자(space) `4개`를 사용한다. (탭(tab)문자 사용 금지)
  - `Hanging indent`(문장 중간에 들여쓰기를 사용하는 형식)의 사용 방법은 아래 예제를 참고
  - 괄호 및 공백 사용은 아래 예제를 참고

Python Style

- 올바른 사용

```
foo = function_name(var_one, var_two, var_three, var_four)

def long_long_long_long_function_name(
        var_one,
        var_two,
        var_three,
        var_four):
    print(var_one)
```

- 잘못된 사용

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

foo = long_function_name(var_one, var_two,
    var_three, var_four)

def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Python Style

- 괄호 (Brace)

  - 괄호는 계산식 및 배열 인덱스로 사용하며, 자료형에 따라 적절한 괄호(대괄호 `[]`, 중괄호 `{}`, 소괄호 `()`)를 사용

```
list = [1, 2, 3, 4, 5]
dictionary = {'age': 30, 'name': '홍길동'}
tupple = (1, 2, 3, 4, 5)
```

Python Style

- 주석(Comments)

  - 문서 주석에는 `"""`을 사용하며 Docstring Conventions을 기술한 PEP 257 을 준수. 구현 주석에는 `#`을 사용

- 린터(Linters)

  - Python 코드 스타일의 자동 오류 검출을 위하여 ament_flake8를 사용

- 기타

  - 모든 문자는 큰 따옴표(`"`, double quotes)가 아닌 작은 따옴표(`'`, single quotes)를 사용하여 표현

2. ROS2 프로그래밍 기초

패키지 생성

- ros2 pkg create 명령어 사용(옵션 사용). 실행 위치는 사용자 작업 폴더

```
$ ros2 pkg create [패키지이름] --build-type [빌드 타입] --dependencies [의존하는패키지1] [의존하는패키지n]
```

```
$ cd ~/robot_ws/src/
$ ros2 pkg create my_first_ros_rclpy_pkg --build-type ament_python --dependencies rclpy std_msgs
```

- 의존 패키지로 rclpy와 std_msgs를 옵션으로 사용
- package.xml에서 직접 입력해도 무방

패키지 생성

- '~/robot_ws/src'에 'my_first_ros_rclpy_pkg' 패키지 폴더와 ROS 패키지가 갖추어야 할 기본 내부 폴더 그리고 package.xml 파일들이 생성

```
.
├── my_first_ros_rclpy_pkg
│   └── __init__.py
├── resource
│   └── my_first_ros_rclpy_pkg
├── test
│   ├── test_copyright.py
│   ├── test_flake8.py
│   └── test_pep257.py
├── package.xml
├── setup.cfg
└── setup.py

3 directories, 8 files
```

패키지 설정 파일(package.xml)

- build_type은 C++은 ament_cmake, Python은 ament_python

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.or
<package format="3">
  <name>my_first_ros_rclpy_pkg</name>
  <version>0.0.2</version>
  <description>ROS 2 rclpy basic package for the ROS 2 seminar</description>
  <maintainer email="pyo@robotis.com">Pyo</maintainer>
  <license>Apache License 2.0</license>
  <author email="mikael@osrfoundation.org">Mikael Arguedas</author>
  <author email="pyo@robotis.com">Pyo</author>

  <depend>rclpy</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

파이썬 패키지 설정 파일(setup.py)

- entry_points 옵션의 console_scripts 키를 사용한 실행 파일 설정
- `helloworld_publisher`과 `helloworld_subscriber` 콘솔 스크립트는 각각 my_first_ros_rclpy_pkg.helloworld_publisher 모듈과 my_first_ros_rclpy_pkg.helloworld_subscriber 모듈의 main 함수가 호출
- 이를 통해 `ros2 run` 또는 `ros2 launch`를 이용하여 해당 스크립트를 실행

Python Style

```python
from setuptools import find_packages
from setuptools import setup

package_name = 'my_first_ros_rclpy_pkg'

setup(
    name=package_name,
    version='0.0.2',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    author='Mikael Arguedas, Pyo',
    author_email='mikael@osrfoundation.org, pyo@robotis.com',
    maintainer='Pyo',
    maintainer_email='pyo@robotis.com',
    keywords=['ROS'],
    classifiers=[
        'Intended Audience :: Developers',
        'License :: OSI Approved :: Apache Software License',
        'Programming Language :: Python',
        'Topic :: Software Development',
    ],
    description='ROS 2 rclpy basic package for the ROS 2 seminar',
    license='Apache License, Version 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'helloworld_publisher = my_first_ros_rclpy_pkg.helloworld_publisher:main',
            'helloworld_subscriber = my_first_ros_rclpy_pkg.helloworld_subscriber:main',
        ],
    },
)
```

파이썬 패키지 환경 설정 파일(setup.cfg)

- 패키지 환경 설정 파일 (setup.cfg)에서의 주의점은 `my_first_ros_rclpy_pkg`와 같이 패키지 이름을 기재해야한다는 것
- 나중에 colcon를 이용하여 빌드하게 되면 `/home/[유저이름]/robot_ws/install/my_first_ros_rclpy_pkg/lib/my_first_ros_rclpy_pkg`와 같은 지정 폴더에 실행 파일이 생성된다는 점

```
[develop]
script-dir=$base/lib/my_first_ros_rclpy_pkg
[install]
install-scripts=$base/lib/my_first_ros_rclpy_pkg
```

퍼블리셔 노드 작성

- 퍼블리셔 노드의 Python 코드는 `~/robot_ws/src/my_first_ros_rclpy_pkg/my_first_ros_rclpy_pkg/` 폴더에 `helloworld_publisher.py`라는 이름으로 소스 코드 파일을 직접 생성
- rclpy의 Node 클래스를 사용, 퍼블리셔의 QoS 설정을 위하여 QoSProfile 클래스를 사용
- 메시지의 타입은 std_msgs.msg 모듈의 String 메시지 인터페이스를 사용

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String
```

퍼블리셔 노드 작성

- 이 노드의 메인 클래스는 HelloworldPublisher으로 Node클래스를 상속하여 사용

```
class HelloworldPublisher(Node):
```

- 부모 클래스를 호출하고 노드 이름을 'helloworld_publisher ' 로 지정
- QoSProfile을 호출하고 depth를 10으로 설정(버퍼에 10개까지 저장하라는 설정)
- 매개변수로는 토픽 메시지 타입(String)과 토픽 이름(helloworld), QoS(qos_profile) 설정
- create_timer로 콜백함수 수행(timer_period_sec를 1로 설정)
- count는 콜백함수에 사용되는 카운터 값

퍼블리셔 노드 작성


```python
def __init__(self):
    super().__init__('helloworld_publisher')
    qos_profile = QoSProfile(depth=10)
    self.helloworld_publisher = self.create_publisher(String, 'helloworld', qos_profile)
    self.timer = self.create_timer(1, self.publish_helloworld_msg)
    self.count = 0
```

퍼블리셔 노드 작성

- 보낼 메시지는 msg.data에 저장. 매번 콜백함수가 실행될때마다 1씩 증가하는 count 값을 문자열에 포함시켜 publish 함수를 통해 퍼블리시
- get_logger 함수는 콘솔창에 출력하는 함수로 종류에 따라 debug, info, warning, error, fatal과 같이 5가지 종류가 있음
- 일반적인 정보 전달에는 info를 사용하기에 퍼블리시 메시지를 출력(print와 유사)

```
def publish_helloworld_msg(self):
    msg = String()
    msg.data = 'Hello World: {0}'.format(self.count)
    self.helloworld_publisher.publish(msg)
    self.get_logger().info('Published message: {0}'.format(msg.data))
    self.count += 1
```

퍼블리셔 노드 작성

- rclpy.init을 이용하여 초기화하고 rclpy.spin 함수로 생성한 노드를 spin시켜 지정된 콜백함수가 실행되도록 함
- Ctrl + c와 같은 인터럽트 시그널 예외 상황에서는 node 소멸, rclpy.shutdown 함수로 노드 종료

```
def main(args=None):
    rclpy.init(args=args)
    node = HelloworldPublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

서브스크라이버 노드 작성

- ~/robot_ws/src/my_first_ros_rclpy_pkg/my_first_ros_rclpy_pkg/` 폴더에 `helloworld_subscriber.py`라는 이름으로 소스 코드 파일을 직접 생성
- import 구문은 퍼블리셔와 동일

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String
```

- 이 노드의 메인 클래스는 HelloworldSubscriber으로 Node클래스를 상속하여 사용

```
class HelloworldSubscriber(Node):
```

서브스크라이버 노드 작성

- 토픽 메시지 타입으로 `String`, 토픽 이름으로 `helloworld`, 콜백함수는 subscribe_topic_message, QoS 설정으로 좀전에 설정한 `qos_profile`으로 설정

```
def __init__(self):
    super().__init__('Helloworld_subscriber')
    qos_profile = QoSProfile(depth=10)
    self.helloworld_subscriber = self.create_subscription(
        String,
        'helloworld',
        self.subscribe_topic_message,
        qos_profile)
```

- subscribe_topic_message 함수. Msg.data를 info 함수를 통해 출력

```
def subscribe_topic_message(self, msg):
    self.get_logger().info('Received message: {0}'.format(msg.data))
```

서브스크라이버 노드 작성

- HelloworldSubscriber을 node로 선언하여 사용한다는 것 이외에는 위에서 설명한 퍼블리셔 노드의 main 함수와 동일

```python
def main(args=None):
    rclpy.init(args=args)
    node = HelloworldSubscriber()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

빌드

- workspace로 이동하고 colcon build 명령어로 전체 빌드

```
(워크스페이스내의 모든 패키지 빌드하는 방법)
$ cd ~/robot_ws && colcon build --symlink-install

(특정 패키지만 빌드하는 방법)
$ cd ~/robot_ws && colcon build --symlink-install --packages-select [패키지 이름1] [패키지 이름2] [패키지

(특정 패키지 및 의존성 패키지를 함께 빌드하는 방법)
$ cd ~/robot_ws && colcon build --symlink-install --packages-up-to [패키지 이름]
```

빌드

- my_first_ros_rclpy_pkg 패키지만 빌드하려면 하기와 같은 명령어를 통해 가능

```
$ cd ~/robot_ws
$ colcon build --symlink-install --packages-select my_first_ros_rclpy_pkg
Starting >>> my_first_ros_rclpy_pkg
Finished <<< my_first_ros_rclpy_pkg [0.66s]

Summary: 1 package finished [0.87s]
```

- 특정 패키지의 첫 빌드 때에는 빌드 후에 하기 명령어와 같이 환경 설정 파일을 불러와서 실행 가능한 패키지의 노드 설정들을 해줘야 빌드된 노드를 실행할 수 있으니 아래와 같이 실행

```
. ~/robot_ws/install/local_setup.bash
```

실행

- 각 노드의 실행은 ros2 run 명령어를 통해 아래와 같이 실행

```
$ ros2 run my_first_ros_rclpy_pkg helloworld_subscriber
[INFO]: Received message: Hello World: 0
[INFO]: Received message: Hello World: 1
[INFO]: Received message: Hello World: 2
[INFO]: Received message: Hello World: 3
[INFO]: Received message: Hello World: 4
[INFO]: Received message: Hello World: 5
```

```
$ ros2 run my_first_ros_rclpy_pkg helloworld_publisher
[INFO]: Published message: Hello World: 0
[INFO]: Published message: Hello World: 1
[INFO]: Published message: Hello World: 2
[INFO]: Published message: Hello World: 3
[INFO]: Published message: Hello World: 4
[INFO]: Published message: Hello World: 5
```

3. 토픽, 서비스, 액션 인터페이스

ROS2 인터페이스 신규 작성

- ROS 2 프로그래밍은 노드간의 메시지 통신을 위해 정수(integer), 부동 소수점(floating point), 불(boolean)같은 기본 인터페이스를 모아둔 std_msgs이나 병진속도 x, y, z, 회전속도 x, y, z를 표현할 수 있는 geometry_msgs의 Twist.msg 인터페이스, 레이저 스캐닝 값을 담을 수 있는 sensor_msgs의 LaserScan.msg과 같은 이미 만들어 놓은 인터페이스를 사용하는 것이 일반적
- 하지만 이러한 인터페이스들이 유저가 원하는 모든 정보를 담을 수는 없고 토픽 인터페이스 이외에 서비스나 액션 인터페이스는 매우 기본적인 인터페이스만 제공하기에 사용자가 원하는 정보의 형태가 아니라면 새로 만들어 써야함

ROS2 인터페이스 신규 작성

- 인터페이스으로만 구성된 패키지를 별도로 만들어 사용하는 것이 의존성면에서 관리 용이
- 예를 들어 A라는 패키지에서 a라는 인터페이스를 사용한다고 했을 때 a라는 인터페이스를 사용하는 B, C라는 패키지가 있다면 B와 C패키지는 a 인터페이스가 담긴 A라는 패키지를 통째로 의존
- 이러한 이유로 인터페이스는 별도의 독립된 패키지로 구성

  (예: geometry_msgs, turtlebot3_msgs)
- msg_srv_action_interface_example 이라는 이름의 패키지를 만들 것

  – ArithmeticArgument.msg, ArithmeticOperator.srv, ArithmeticChecker.action

인터페이스 패키지 생성

- msg_srv_action_interface_example 이라는 패키지를 생성 후 아래와 같이 인터페이스 파일을 담을 msg, srv, action 폴더 생성

```
$ cd ~/robot_ws/src
$ ros2 pkg create --build-type ament_cmake msg_srv_action_interface_example
$ cd msg_srv_action_interface_example
$ mkdir msg srv action
```

- 각 인터페이스 파일을 각 폴더안에 넣기(인터페이스 류의 파일명은 `CamelCased` 규칙을 따름, 변환 후 구조체 및 XKDLQ으로 사용되기 때문)

```
├── action
│   └── ArithmeticChecker.action
├── msg
│   └── ArithmeticArgument.msg
└── srv
    └── ArithmeticOperator.srv
```

인터페이스 패키지 생성

- ArithmeticArgument.msg 생성
  - 저장 위치 : msg_srv_action_interface_example/msg

```
# Messages
builtin_interfaces/Time stamp
float32 argument_a
float32 argument_b
```

인터페이스 패키지 생성

- ArithmeticOperator.srv 생성
  - 저장 위치: msg_srv_action_interface_example/srv

```
# Constants
int8 PLUS = 1
int8 MINUS = 2
int8 MULTIPLY = 3
int8 DIVISION = 4

# Request
int8 arithmetic_operator
---
# Response
float32 arithmetic_result
```

인터페이스 패키지 생성

- ArithmeticChecker.action 생성
  - 저장 위치: msg_srv_action_interface_example/action

```
# Goal
float32 goal_sum
---
# Result
string[] all_formula
float32 total_sum
---
# Feedback
string[] formula
```

msg, srv, action 인터페이스 비교

	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
사용 예	[ArithmeticArgument.msg]	[ArithmeticOperator.srv]	[ArithmeticChecker.action]
	builtin_interfaces/Time stamp float32 argument_a float32 argument_b	# Constants int8 PLUS = 1 int8 MINUS = 2 int8 MULTIPLY = 3 int8 DIVISION = 4 # Request int8 arithmetic_operator --- # Response float32 arithmetic_result	# Goal float32 goal_sum --- # Result string[] all_formula float32 total_sum --- # Feedback string[] formula

패키지 설정 파일(packge.xml)

- msg_srv_action_interface_example의 package.xml은 아래와 같이 작성

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org
<package format="3">
  <name>msg_srv_action_interface_example</name>
  <version>0.2.0</version>
  <description>
    ROS 2 example for message, service and action interface
  </description>
  <maintainer email="passionvirus@gmail.com">Pyo</maintainer>
  <license>Apache 2.0</license>
  <author email="passionvirus@gmail.com">Pyo</author>
  <author email="routiful@gmail.com">Darby Lim</author>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>builtin_interfaces</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

패키지 설정 파일(packge.xml)

- 일반적인 패키지와 다른 점은 아래와 같이 빌드 시에 DDS에서 사용되는 IDL(Interface Definition Language)생성과 관련한 `rosidl_default_generators`이 사용된다는 점과
- 실행 시에 `builtin_interfaces`와 `rosidl_default_runtime`이 사용된다는 점
- 그 이외에는 일반적인 패키지의 설정과 동일. 인터페이스 전용 패키지를 만들 때에는 필수적인 의존성 패키지이니 기억

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>builtin_interfaces</exec_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
```

빌드 설정 파일(CmakeLists.txt)

- msg_srv_action_interface_example의 CMakeLists.txt은 아래와 같이 작성

```
##############################################################################
# Set minimum required version of cmake, project name and compile options
##############################################################################
cmake_minimum_required(VERSION 3.5)
project(msg_srv_action_interface_example)

if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Wpedantic")
endif()

##############################################################################
# Find and load build settings from external packages
##############################################################################
find_package(ament_cmake REQUIRED)
find_package(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

##############################################################################
# Declare ROS messages, services and actions
##############################################################################
set(msg_files
  "msg/ArithmeticArgument.msg"
)

set(srv_files
  "srv/ArithmeticOperator.srv"
)

set(action_files
  "action/ArithmeticChecker.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)

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

빌드 설정 파일(CmakeLists.txt)

- 일반적인 패키지와 다른 점은 아래와 같이 `set` 명령어로 msg, srv, action 파일을 지정해주고 `rosidl_generate_interfaces`에 해당 셋들을 기입해주면 끝

```
set(msg_files
  "msg/ArithmeticArgument.msg"
)

set(srv_files
  "srv/ArithmeticOperator.srv"
)

set(action_files
  "action/ArithmeticChecker.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)
```

빌드

```
$ cw
$ cbp msg_srv_action_interface_example
```

- `cw`는 `cd ~/robot_ws`의 alias
- * `cbp`는 `colcon build --symlink-install --packages-select`의 alias
- 빌드한 후 빌드에 문제가 없다면 `~/robot_ws/install/msg_srv_action_interface_example` 폴더 안에 우리가 작성한 ROS 인터페이스를 사용하기 위한 파일들이 저장

빌드

- msg_srv_action_interface_example의 변환된 인터페이스 파일 (*.h, *hpp, *.py, *idl)
- - *.h, *hpp: include/msg_srv_action_interface_example
- - *.py: lib/python3.6/site-packages/msg_srv_action_interface_example
- - *.idl: share/msg_srv_action_interface_example

```
~/robot_ws/install/msg_srv_action_interface_example
├── include
│   └── msg_srv_action_interface_example
│       ├── action
│       ├── msg
│       └── srv
├── lib
│   └── python3.6
│       └── site-packages
│           └── msg_srv_action_interface_example
│               ├── action
│               ├── msg
│               └── srv
└── share
    └── msg_srv_action_interface_example
        ├── action
        ├── msg
        └── srv
```