ROS 기초

05 ROS 파일 및 빌드시스템

목차 ROS 기초

1. ROS2 파일시스템

파일시스템

- ROS2 패키지 및 소스 코드 검색
- 메시지 파일, 실행 파일, 파라미터 설정 및 환경 설정 파일
- 사용자에게 일관된 경험을 제공하기 위해 동일 구조를 가지고 있음

패키지와 메타패키지

- 소프트웨어 구성을 위한 기본 단위가 패키지
- 응용프로그램은 패키지 단위로 개발되고 관리됨(노드를 하나 이상 포함)
- 450여개가 기본 패키지, ROS Index 기준 624개의 패키지
- sudo apt install 명령어로 설치 가능한 패키지는 900 여개
- 메타패키지는 공통된 목적을 지닌 패키지들을 모아둔 패키지의 집합
- 각 패키지는 package.xml 파일을 포함하고 이름, 저작자, 라이선스, 의존성 등을 기술
- ament는 기본적으로 Cmake를 이용하고 있어서 CMakeLists.txt 파일에 빌드 설정을 기술하고 기타 파일 들로 구성

바이너리 설치과 소스 코드 설치

- 패키지의 사용 목적에 따라 달리 사용
- 바이너리 설치

```
$ sudo apt install ros-foxy-teleop-twist-joy
```

- 소스코드 설치

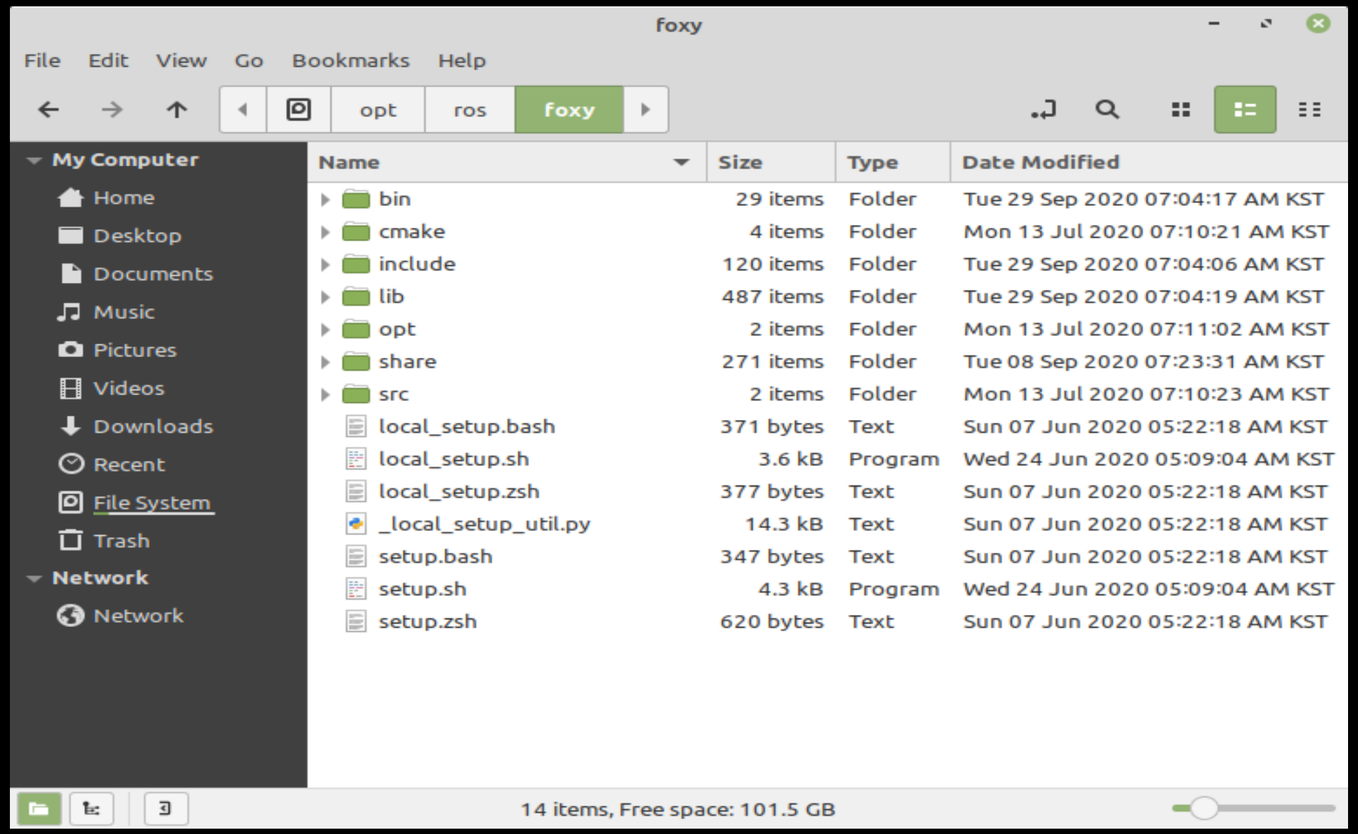
```
$ cd ~/robot_ws/src
$ git clone https://github.com/ros2/teleop_twist_joy.git
$ cd ~/robot_ws/
$ colcon build --symlink-install --packages-select teleop_twist_joy
```

기본 설치 폴더와 사용자 작업 폴더

- /opt 폴더에 ros 이름의 폴더가 생성되고 그 안에 버전 별로 폴더 구성
- /opt/ros/foxy에 핵심 유틸리티와 rqt, Rviz, 로봇 관련 라이브러리, 시뮬레이션, 네비게이션 패키지 등이 설치
- 사용자 작업 폴더는 원하는 곳으로 폴더로 생성 '~/robot_ws/(~/은 리눅스에서 'home/사용자명/' 에 해당하는 폴더를 의미)

기본 설치 폴더와 사용자 작업 폴더

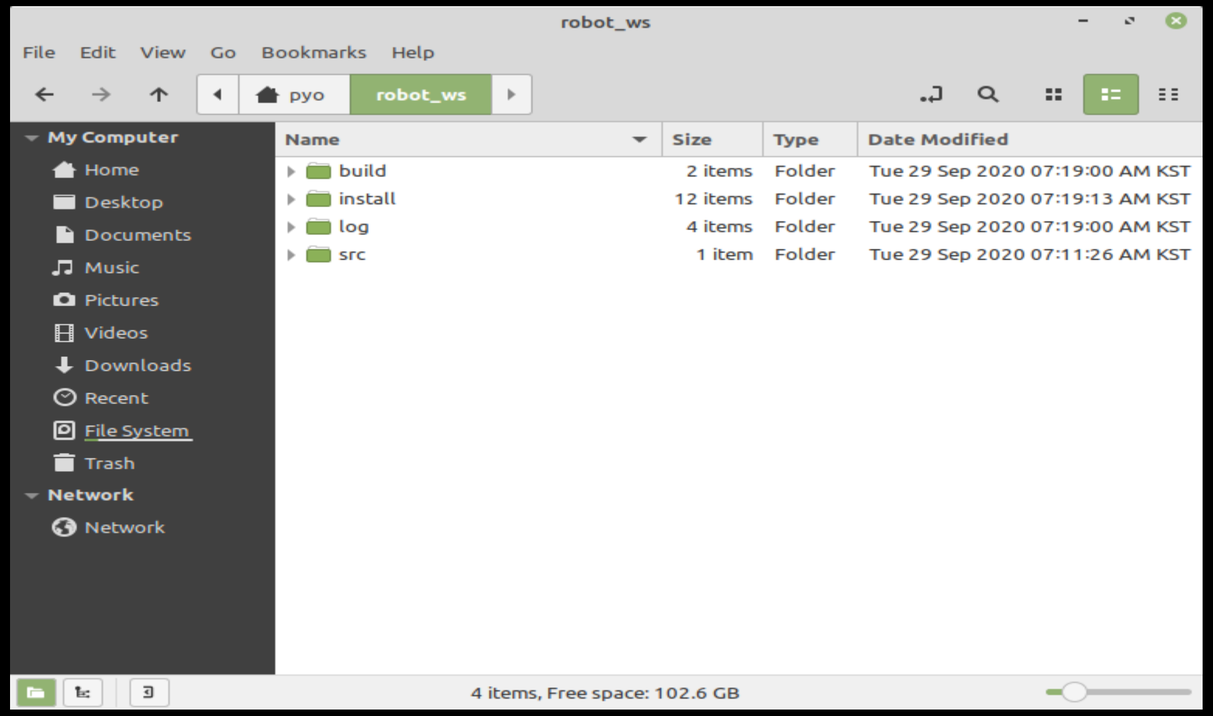
- /opt/ros/[버전이름] 폴더에 설치 ROS 2 Foxy Fitzroy 버전을 설치했다면 설치된 경로는 '/opt/ros/foxy'



- `/bin` 실행 가능한 바이너리 파일
- `/cmake` 빌드 설정 파일
- `/include` 헤더 파일
- `/lib` 라이브러리 파일
- `/opt` 기타 의존 패키지
- `/share` 패키지의 빌드, 환경 설정 파일
- `local_setup.*` 환경 설정 파일
- `setup.*` 환경 설정 파일

기본 설치 폴더와 사용자 작업 폴더

- ~/robot_ws/를 사용자 작업 폴더로 사용하고 사용자 이름이 elice, workspace 폴더 이름을 robot_ws라고 했다면 사용자 작업 폴더 경로는 /home/elice/robot_ws



- /build 빌드 설정 파일용 폴더
- /install msg, srv 헤더 파일과 사용자 패키지 라이브러리, 실행 파일용 폴더
- /log 빌드 로깅 파일용 폴더
- /src 사용자 패키지용 폴더

기본 설치 폴더와 사용자 작업 폴더

- ~/robot_ws/src 폴더는 사용자 패키지 공간

  - /src C/C++ 코드용 폴더
  - /include C/C++ 헤더 파일용 폴더 (폴더 안에는 각 패키지 이름별 폴더로 패키지별 헤더를 구분함)
  - /param 파라미터 파일용 폴더
  - /launch roslaunch에 사용되는 launch 파일용 폴더
  - /패키지_이름_폴더 Python 코드용 폴더
  - /test 테스트 코드 및 테스트 데이터용 폴더
  - /msg 메시지 파일용 폴더
  - /srv 서비스 파일용 폴더
  - /action 액션 파일용 폴더
  - /doc 문서용 폴더
  - package.xml: 패키지 설정 파일 (REP-0140, REP-0149 참고)
  - CMakeLists.txt: C/C++ 빌드 설정 파일
  - setup.py: 파이썬 코드 환경 설정 파일
  - README: 사용자 문서, github 리포지토리의 메인에 표시된다.
  - CONTRIBUTING: 해당 패키지 개발에 공헌하는 방법을 기술하는 파일
  - LICENSE: 이 패키지의 라이선스를 기술하는 파일
  - CHANGELOG.rst: 이 패키지의 버전별 변경 사항 모음 파일 (REP-0132 참고)

2. ROS2 빌드시스템과 빌드 툴

ROS2 빌드시스템과 빌드 툴

- 빌드 시스템과 빌드 툴의 큰 차이는 단일 패키지를 대상으로 하느냐 전체 패키지를 대상하느냐임. 빌드시스템 – 단일 패키지, 빌드 툴 – 시스템 전체
- ament를 사용하며 CmakeList.txt 기술된 설정을 기반으로 빌드 수행
- Devel과 같은 공간을 사용하지 않고 AMENT_PREFIX_PATH와 같은 고유한 환경 설정 사용

빌드 툴

- catkin_make : 기본 툴로 ROS Fuerte 버전 이후 rosbuild의 대체 툴로서 오랜 기간 사용되어 온 ROS 1의 대표 빌드 툴
- **catkin_make_isolated** : 하나의 CMake으로 복수의 패키지를 빌드 할 수 있었으며 격리 빌드를 지원함으로써 모든 패키지를 별도로. 이를 통해 설치용 폴더를 분리하거나 병합할 수 있게 됨
- **catkin_tools** : catkin_make, catkin_make_isolated의 독립 사용에 불편함을 해결하고 Python으로 구성된 패키지도 관리할 수 있게 해주는 툴로 catkin_make의 부족한 부분을 제공
- ament_tools : catkin_make, catkin_make_isolated의 독립 사용에 불편함을 해결하고 Python으로 구성된 패키지도 관리할 수 있게 해주는 툴로 catkin_make의 부족한 부분을 제공
- colcon : ROS 1과 ROS 2 모두를 지원하기 위하여 통합된 빌드 툴로서 소개되었으며 ROS 2 Bouncy 이후 ROS 2의 기본 빌드 툴로 사용

패키지 생성

- 패키지 생성(사용자 작업 폴더에서 실행)

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

- python을 사용하도 rqt plugin을 사용해야하기에 ament_cmake 기입

```
$ ros2 pkg create test_pkg_rclcpp --build-type ament_cmake
$ ros2 pkg create test_pkg_rclpy --build-type ament_python
```

- 새로운 터미널을 열고(Ctrl + Alt + t) 다음 명령어로 작업 폴더 이동

```
$ cd ~/robot_ws/src
```

- 패키지 이름은 모두 소문자를 사용하고 공백이 있으면 안되며 '_ ' 로 단어를 이어 붙임

```
$ ros2 pkg create my_first_ros_rclpy_pkg --build-type ament_python --dependencies rclpy std_msgs
```

패키지 생성

- 기본 구성(클라이언트 라이브러리에 따라 rclcpp, rclpy를 옵션으로 달아줌)

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

3 directories, 2 files
```

```
(my_first_ros_rclpy_pkg)
.
├── 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
```

빌드

- workspace로 이동 후 colcon build 명령어로 전체 빌드(특정 패키지만 선택하여 빌드시에는 –packages-select, symlink를 이용하려면 –symlink-install를 붙임)
- 첫 번째는 전체 패키지 빌드, 두번 째는 해당 패키지만 빌드

```
$ cd ~/robot_ws && colcon build --symlink-install
```

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-select [패키지 이름]
```

빌드 명령어의 옵션

- Multiple workspace : `catkin_ws`와 같이 특정 워크스페이스를 확보하고 하나의 워크스페이스에서 모든 작업, ROS 2에서는 복수의 독립된 워크스페이스를 사용할 수 있어 작업 목적 및 패키지 종류별로 관리
- No non-isolated build : ROS 2에서는 이전 빌드 시스템인 catkin에서 일부 기능으로 사용되었던 `catkin_make_isolated` 형태와 같은 격리 빌드만을 지원. 모든 패키지를 별도로 빌드이 기능 변화를 통해 설치용 폴더를 분리하거나 병합
- No devel space : catkin은 패키지를 빌드 한 후 devel 이라는 폴더에 코드를 저장. 이 폴더는 패키지를 설치할 필요 없이 패키지를 사용할 수 있는 환경을 제공

빌드 시스템 부가 기능

- vcstools(버전 컨트롤 시스템 툴)

```
$ mkdir -p ~/ros2_foxy/src
$ cd ~/ros2_foxy
$ wget https://raw.githubusercontent.com/ros2/ros2/foxy/ros2.repos
$ vcs import src < ros2.repos
```

- ros2. repos 파일은 vcs 타입은 무엇이고, 리포지토리 주소는 어떻게 되며, 설치해야 하는 브랜치는 어떤 것인지가 명시된 파일

빌드 시스템 부가 기능

- rosdep(의존성 관리 툴)
  - 위에 언급된 툴들은 의존성을 고려하지만 의존성 자체를 해결해 주지는 않음.

Package.xml에 기술된 정보를 가지고 패키지들을 설치해주는 역할

```
$ sudo rosdep init
$ rosdep update
$ rosdep install --from-paths src --ignore-src --rosdistro foxy -y --skip-keys "console_bridge fastc
```

빌드 시스템 부가 기능

- bloom(바이너리 패키지 관리툴)
  - 바이너리 패키지 배포 및 관리를 위한 툴
  - dpkg-buildpackage와 같은 플랫폼 종속 도구가 바이너리 패키지 빌드시 사용
  - 개발이 끝나고 배포하고 유지관리할 때 사용하게 되며 시간과 노력이 많이 들어감

3. 환경 설정, 빌드 설정

패키지 설정 파일(packge.xml)

- 패키지의 정보를 기술하는 파일
- rclcpp 계열(좌 - my_first_ros_rclcpp_pkg)
- rclpy 계열(우 my_first_ros_rclpy_pkg)

```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_rclcpp_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pyo@robotis.com">pyo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>

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

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

```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>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pyo@robotis.com">pyo</maintainer>
  <license>TODO: License declaration</license>

  <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>
```

패키지 설정 파일(packge.xml)

- <?xml> 문서 문법을 정의하는 문구로 아래의 내용은 xml 버전 1.0을 따르고 있다는 것을 알린다.
- <package> 이 구문부터 맨 끝의 </package>까지가 ROS 패키지 설정 부분이다. 세부 사항으로 format="3" 이라고 패키지 설정 파일의 버전을 기재한다. ROS 2는 3를 사용하면 된다.
- <name> 패키지의 이름이다. 패키지를 생성할 때 입력한 패키지 이름이 사용된다. 다른 옵션도 마찬가지지만 이는 사용자가 원할 때 언제든지 변경할 수 있다.
- <version> 패키지의 버전이다. 자유롭게 지정할 수 있는데 나중에 패키지를 바이너리 패키지로 공개한다면 버전 관리에 사용되므로 신중할 필요가 있다.
- <description> 패키지의 간단한 설명이다. 보통 2~3 문장으로 기술한다.
- <maintainer> 패키지 관리자의 이름과 이메일 주소를 기재한다.
- <license> 라이선스를 기재한다. Apache 2.0, BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3, Proprietary 등을 기재하면 된다.
- <url> 패키지를 설명하는 웹 페이지 또는 버그 관리, 소스 코드 저장소 등의 주소를 기재한다. 이 종류에 따라 type에 website, bugtracker, repository를 대입하면 된다.
- <author> 패키지 개발에 참여한 개발자의 이름과 이메일 주소를 적는다. 복수의 개발자가 참여한 경우에는 바로 다음 줄에 <author> 태그를 이용하여 추가로 넣어주면 된다.
- <buildtool_depend> 빌드 툴의 의존성을 기술한다.
- <build_depend> 패키지를 빌드할 때 필요한 의존 패키지 이름을 적는다.
- <exec_depend> 패키지를 실행할 때 필요한 의존 패키지 이름을 적는다.
- <test_depend> 패키지를 테스트할 때 필요한 의존 패키지 이름을 적는다.
- <export> 위에서 명시하지 않은 확장 태그명을 사용할 때 쓰인다. 빌드 타입을 적는 <build_type>, RViz 플러그인에 사용되는 <rviz>, RQt 플러그인에 사용되는 <rqt_gui>, deprecated되는 패키지일 경우 유저에게 알릴 수 있는 <deprecated> 태그 등이 있다.

빌드 설정 파일(CMakeLists.txt)

- 빌드 환경을 기술(멀티 플랫폼에서 빌드)
- C++을 사용하는 my_first_ros_rclcpp_pkg만이 CmakeLists.txt 파일 존재

```
cmake_minimum_required(VERSION 3.5)
project(my_first_ros_rclcpp_pkg)

# 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 MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # uncomment the line when a copyright and license is not present in all source files
  #set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # uncomment the line when this package is not in a git repo
  #set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

파이썬 패키지 설정 파일(setup.py)

- `name`: 패키지의 이름
- `version`: 패키지의 버전
- `packages`: 의존하는 패키지, 하나씩 나열해도 되지만 `find_packages()`를 기입해주면 자동으로 의존하는 패키지를 찾아준다.
- `data_files`: 이 패키지에서 사용되는 파일들을 기입하여 함께 배포한다.
  - `ROS`에서는 주로 `resource` 폴더 내에 있는 `ament_index`를 위한 패키지의 이름의 빈 파일이나 `package.xml`, `*.launch.py`, `*.yaml` 등을 기입한다.
- `install_requires`: 의존하는 패키지, 이 패키지를 `pip`을 통해 설치할 때 이곳에 기술된 패키지들을 함께 설치하게 된다.
  - `ROS`에서는 `pip`로 설치하지 않기에 `setuptools`, `launch`만을 기입해준다.
- `tests_require`: 테스트에 필요한 패키지, `ROS`에서는 `pytest`를 사용한다.
- `zip_safe`: 설치시 zip 파일로 아카이브할지 여부를 설정한다.
- `author`, `author_email`, `maintainer`, `maintainer_email`: 저작자, 관리자의 이름과 이메일을 기입한다.
- `keywords`: 이 패키지의 키워드, Python Package Index (PyPI) [8] 배포시 검색하여 이 패키지를 찾을 수 있도록 한다.
- `classifiers`: PyPI에 등록될 메타 데이터 설정으로 `PyPI` 페이지의 좌측 Meta란에서 확인 가능하다.
- `description`: 패키지 설명을 기입한다.
- `license`: 라이선스 종류를 기입한다.
- `entry_points`: 플랫폼 별로 콘솔 스크립트를 설치하도록 콘솔 스크립트 이름과 호출 함수를 기입한다.

파이썬 패키지 설정 파일(setup.py)


```python
from setuptools import setup

package_name = 'my_first_ros_rclpy_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pyo',
    maintainer_email='pyo@robotis.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

파이썬 패키지 환경 설정 파일(setup.cfg)

- develop, install 옵션으로 스크립트 저장 위치 설정
- 해당 패키지의 이름만 변경하여 사용

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

RQt 플러그인 설정 파일(plugin.xml)

- XML 태그로 각 속성을 기술하는 파일

```
<library path="src">
  <class name="Examples" type="examples_rqt.examples.Examples" base_class_type="rqt_gui_py::Plugin">
    <description>
      A plugin visualizing messages and services values
    </description>
    <qtgui>
      <group>
        <label>Visualization</label>
        <icon type="theme">folder</icon>
        <statustip>Plugins related to visualization</statustip>
      </group>
      <label>Viewer</label>
      <icon type="theme">utilities-system-monitor</icon>
      <statustip>A plugin visualizing messages and services values</statustip>
    </qtgui>
  </class>
</library>
```

패키지 변경로그 파일(CHANGELOG.rst)

- 업데이트 내역을 기술하는 파일

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Changelog for package example_rqt_package
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

0.0.2 (2020-10-22)
------------------
* Added new indicators to view message
* Contributors: Pyo

0.0.1 (2020-10-21)
------------------
* Added example_rqt_package as rqt plugin for visualizing messages and services
* Contributors: Pyo
```

라이선스 파일(LICENSE)

- 코드에 사용된 라이선스를 기술하는 파일, 단일 라이선스의 경우에는 파일명은 LICENSE로 확장자를 붙이지 않음
- 복수 라이선스 표기 방법

```
├── LICENSE/
│   ├── LICENSE (Apache 2.0)
│   └── LICENSE (BSD)
```

패키지 설명 파일(README.md)

- 패키지의 필수 포함 파일은 아님
- 사용자를 배려하는 문서로 Markdown 문법을 따름

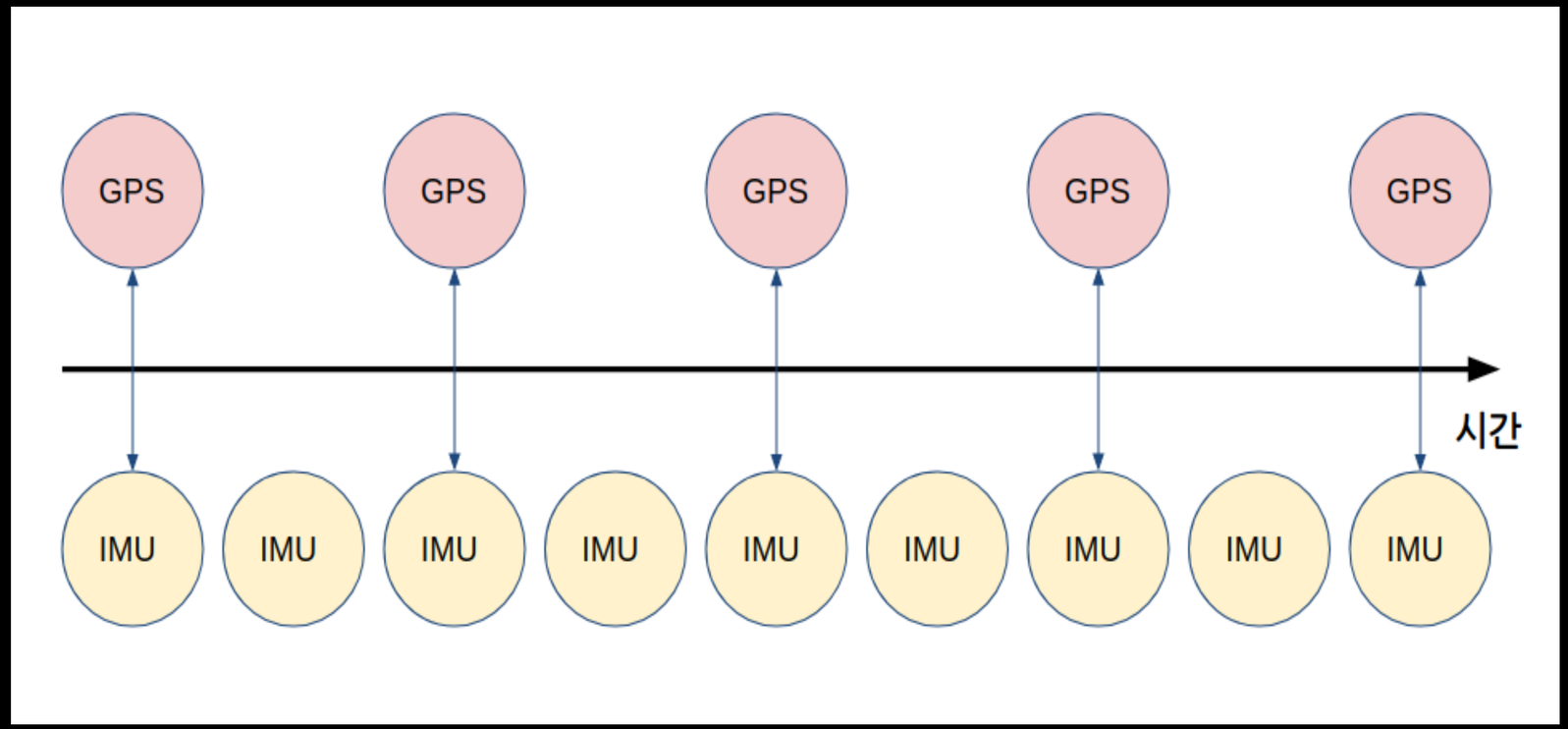
4. Time, Duration, Clock, Rate

시계(Clock)와 시간(Time)

- 다수의 센서를 사용하는 로봇은 각 센서의 변화량과 센서들 간의 시간 동기화가 매우 중요
- 노드들이 통신하며 다양한 정보를 주고 받기 때문에 발간된 정확한 시간이 필수적
- GPS 토픽의 경우 복수개의 인공위성으로 부터 받은 정보를 위도, 경도, 오차 데이터를 발간하는 주기가 약 2초라고 가정, IMU의 로봇 자세 데이터를 발간하는 토픽의 주기는 약 1초라고 가정
- 두 토픽을 구독하여 로봇의 위치를 추정하면 보통 필터를 거치는데 이 때 필터가 동작하는 주기에서 가장 가까운 시간에 발간된 데이터를 골라 인풋으로 넣어줘야만 보다 정확한 결과를 얻을 수 있음

시계(Clock)와 시간(Time)

- 로봇이 4초 후의 GPS 데이터와 3초 후의 IMU 데이터를 필터의 인풋으로 넣는다면 그 아웃풋이 4초 후의 IMU 데이터를 사용한 것보다 좋지 못할 것



시계(Clock)와 시간(Time)

- 토픽에 주요 데이터 뿐만 아니라 토픽 발간 시간을 포함
- stamp, frame id를 포함하고 있는 std_msgs/msg/header는 표준 메시지 타입

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.

# Two-integer timestamp that is expressed as seconds and nanoseconds.
builtin_interfaces/Time stamp

# Transform frame with which this data is associated.
string frame_id
```

시계(Clock)와 시간(Time)

- 시계를 노드가 생성될 때 정해지도록 함. 기본 시계는 System Clock(UTC, 국제 표준)
- rclcpp는 std::chrono 라이브러리를 rclpy는 time 모듈을 캡슐화하여 사용

```
$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=False
[INFO]: sec 1611792893.524822 nsec 1611792893524821381
[INFO]: sec 1611792894.525022 nsec 1611792894525021996
[INFO]: sec 1611792895.524994 nsec 1611792895524993945
[INFO]: sec 1611792896.524888 nsec 1611792896524888265
[INFO]: sec 1611792897.524969 nsec 1611792897524969178
[INFO]: sec 1611792898.524978 nsec 1611792898524977661
[INFO]: Over 5 seconds!
[INFO]: sec 1611792899.525013 nsec 1611792899525012802
[INFO]: sec 1611792900.525024 nsec 1611792900525024424
[INFO]: sec 1611792901.524993 nsec 1611792901524993305
[INFO]: sec 1611792902.525016 nsec 1611792902525015881
[INFO]: sec 1611792903.524888 nsec 1611792903524887863
[INFO]: sec 1611792904.524991 nsec 1611792904524990989
[INFO]: Over 5 seconds!
```

시간 추상화(Time Abstractions)

- 기본 시계 외에도 타임머신처럼 동작하는 시계도 가능
- 과거 데이터를 다룰 때나 시뮬레이션(gazebo, ignition)에서 사용할 수 있음
- rcl/time.h

```
enum rcl_time_source_type_t
 {
   RCL_TIME_SOURCE_UNINITIALIZED = 0,
   RCL_ROS_TIME,
   RCL_SYSTEM_TIME,
   RCL_STEADY_TIME
 };
```

System Time

- 타임서버와의 동기화를 통해 시간이 거꾸로 가는 경우도 있음
- server pc와 remote pc간의 시간동기화는 특정 서버의 시간으로 동기화 가능

```
$ sudo ntpdate ntp.ubuntu.com
28 Jan 09:51:50 ntpdate[3424]: adjust time server 91.189.91.157 offset 0.043664 sec
```

ROS Time

- 보통 시뮬레이션 환경에서 시간을 조절하기 위해 많이 사용
- use_sim_time을 통해 설정하며 True로 설정된 노드는 /clock 토픽을 구독할 때 까지 시간을 0으로 초기화

```
$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=True
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
```

Steady Time, Time API

- Steady time은 Hardware timeouts를 사용한 시간으로 무조건 단조증가 (monotonic) 한다는 특성을 가짐
- Time API는 크게 time, duration, rate가 있음
- Time 클래스는 시간 오퍼레이터 제공 seconds or nanoseconds로 반환

```
if ((now - past).nanoseconds() * 1e-9 > 5) {
    RCLCPP_INFO(node->get_logger(), "Over 5 seconds!");
    past = node->now();
}
```

- ROS2 노드에서는 now 함수를 통해 노드 시간 확인

```
rclcpp::Time now = node->now();
```

Time API

- 좌 : time_rclcpp_example/src/main
- 우 : time_rclpy_example/time_rclpy_example/time_example/main.py

```cpp
#include <memory>
#include <utility>

#include "rclcpp/rclcpp.hpp"
#include "rclcpp/time_source.hpp"

#include "std_msgs/msg/header.hpp"


int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("time_example_node");
  auto time_publisher = node->create_publisher<std_msgs::msg::Header>("time", 10);
  std_msgs::msg::Header msg;

  rclcpp::WallRate loop_rate(1.0);
  rclcpp::Duration duration(1, 0);

  while (rclcpp::ok()) {
    static rclcpp::Time past = node->now();

    rclcpp::Time now = node->now();
    RCLCPP_INFO(node->get_logger(), "sec %lf nsec %ld", now.seconds(), now.nanoseconds());

    if ((now - past).nanoseconds() * 1e-9 > 5) {
      RCLCPP_INFO(node->get_logger(), "Over 5 seconds!");
      past = node->now();
    }

    msg.stamp = now + duration;
    time_publisher->publish(msg);

    rclcpp::spin_some(node);
    loop_rate.sleep();
  }

  rclcpp::shutdown();

  return 0;
}
```

```python
import rclpy
from rclpy.duration import Duration
from std_msgs.msg import Header


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

    node = rclpy.create_node('time_example_node')
    time_publisher = node.create_publisher(Header, 'time', 10)
    msg = Header()

    rate = node.create_rate(1.0)
    duration = Duration(seconds=1, nanoseconds=0)
    past = node.get_clock().now()

    try:
        while rclpy.ok():
            now = node.get_clock().now()
            seconds, nanoseconds = now.seconds_nanoseconds()
            node.get_logger().info('sec {0} nsec {1}'.format(seconds, nanoseconds))

            if ((now - past).nanoseconds * 1e-9) > 5:
                node.get_logger().info('Over 5 seconds!')
                past = node.get_clock().now()

            msg.stamp = (now + duration).to_msg()
            time_publisher.publish(msg)

            rclpy.spin_once(node)
            rate.sleep()
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

Duration

- 순간의 시간(timestamp)이 아닌 기간(3시간 후, 1시간 전 등)을 다룰 수 있는 오퍼레이터를 제공하며 seconds or nanoseconds 단위로 반환
- 이전 시간은 음수로 표기
- 아래 예제는 실제 시간보다 1초 느린 값을 간단히 계산

```
rclcpp::Duration duration(1, 0);
msg.stamp = now + duration;
time_publisher->publish(msg);
```

Rate

- Rate 클래스는 반복문에서 특정 주기를 유지시켜주는 API 제공
- 아래 예시는 WallRate 클래스의 생성자에 헤르츠 단위로 주기 설정 후 sleep() 함수로 주기를 맞춤(Rate는 System clock 사용, WallRate는 Steady clock 사용)
- ROS2는 콜백 함수를 사용하는 Timer API를 제공하기에 이를 사용 추천

```
rclcpp::WallRate loop_rate(1);

while (rclcpp::ok()) {
// 코드 중략
  loop_rate.sleep();
}
```

시계(Clock)와 시간(Time)

- use_sim_time 파라미터 : False, UTC 형식으로 1초 마다 로그, 5초 마다 로그 확인

```
$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=False
[INFO]: sec 1611808317.044796 nsec 1611808317044796531
[INFO]: sec 1611808318.044951 nsec 1611808318044951274
[INFO]: sec 1611808319.044939 nsec 1611808319044938670
[INFO]: sec 1611808320.044937 nsec 1611808320044936942
[INFO]: sec 1611808321.044966 nsec 1611808321044965935
[INFO]: sec 1611808322.044962 nsec 1611808322044962073
[INFO]: Over 5 seconds!
[INFO]: sec 1611808323.044957 nsec 1611808323044957461
[INFO]: sec 1611808324.044966 nsec 1611808324044966062
[INFO]: sec 1611808325.044970 nsec 1611808325044969462
[INFO]: sec 1611808326.044998 nsec 1611808326044997735
[INFO]: sec 1611808327.045034 nsec 1611808327045034118
[INFO]: sec 1611808328.044970 nsec 1611808328044969859
[INFO]: Over 5 seconds!
[INFO]: sec 1611808330.585464 nsec 1611808330585463561
[INFO]: sec 1611808331.585448 nsec 1611808331585448206
[INFO]: sec 1611808332.585462 nsec 1611808332585461648
[INFO]: sec 1611808333.585369 nsec 1611808333585369322
[INFO]: Over 5 seconds!
[INFO]: sec 1611808334.585364 nsec 1611808334585364035
[INFO]: sec 1611808335.585377 nsec 1611808335585377352
[INFO]: sec 1611808336.585368 nsec 1611808336585367969
[INFO]: sec 1611808337.585372 nsec 1611808337585372004
[INFO]: sec 1611808338.585397 nsec 1611808338585397207
[INFO]: sec 1611808339.585366 nsec 1611808339585365766
[INFO]: Over 5 seconds!
```

시계(Clock)와 시간(Time)

- use_sim_time : True, /clock 토픽이 없기에 시간은 항상 0
- /time 토픽을 확인하면 duration을 이용한 시간 수정으로 stamp가 1초씩 느림

```
$ ros2 run time_rclcpp_example time_example --ros-args -p use_sim_time:=True
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
[INFO]: sec 0.000000 nsec 0
$ ros2 topic echo /time
stamp:
  sec: 1
  nanosec: 0
frame_id: ''
---
stamp:
  sec: 1
  nanosec: 0
frame_id: ''
---
stamp:
  sec: 1
  nanosec: 0
frame_id: ''
---
```

시계(Clock)와 시간(Time)

- rclpy로 개발된 노드는 아래와 같고 use_sim_time : True
- 시간이 ROS Time 으로 변경, 결과는 rclcpp와 동일

```
$ ros2 run time_rclpy_example time_example --ros-args -p use_sim_time:=False
[INFO]: sec 1611845885 nsec 1185392
[INFO]: sec 1611845886 nsec 1596232
[INFO]: sec 1611845887 nsec 1656253
[INFO]: sec 1611845888 nsec 1605086
[INFO]: sec 1611845889 nsec 1518249
[INFO]: sec 1611845890 nsec 1772286
[INFO]: Over 5 seconds!
[INFO]: sec 1611845891 nsec 1571463
[INFO]: sec 1611845892 nsec 1440453
[INFO]: sec 1611845893 nsec 1595560
[INFO]: sec 1611845894 nsec 1582543
[INFO]: sec 1611845895 nsec 1552400
[INFO]: sec 1611845896 nsec 1604775
[INFO]: Over 5 seconds!
[INFO]: sec 1611845897 nsec 1605453
```