For an example, check out the CTD driver.

1. Set up a Git repo that hosts the msg (C++ node) and driver (Python node) together

Let's say we are creating a driver called *awesome_driver*.

1. Create a Git repo called *awesome_driver*
2. Clone this repo into your *colcon_ws/src*
3. I like adding *.gitignore* file to my repo, so that Git stops bothering me about build, install and log folders are untracked.

2. Create the C++ package for custom msg

Create package structure

Inside *colcon_ws/src/awesome_driver*, create a ROS2 C++ package:

```
cd colcon_ws/src/awesome_driver
# Create package
ros2 pkg create awesome_interfaces
# Remove unnecessary folders
cd awesome_interfaces
rm -rf include/ src/
# Make folder for msg
mkdir msg
```

Update the message

Copy paste the custom message definition from LoLo driver into the newly created `msg` folder. Note that ROS2's messages needs to be alphanumeric, so no hyphens or underscores are allowed. Name the message with CamelCase, i.e. *AwesomeName.msg*.

If your message has a field with type *time*, you need to change the type to *builtin_interfaces/Time*, since things moved in ROS2... So:

```
###### original definition
time time
float32 data1
float32 data2
############################

###### new definition
builtin_interfaces/Time time
float32 data1
float32 data2
############################
```

You should now have a folder structure that looks like this:

```
colcon_ws/src/awesome_driver
├── awesome_interfaces
|    ├── CMakeLists.txt
|    ├── msg
|        ├── AwesomeName.msg
|    └── package.xml
```

You now need to update the *CMakeLists.txt* and *package.xml* files.

Update package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>awesome_interfaces</name>
  <version>0.0.0</version>
  # Update the description and maintainer
  <description>TODO: Package description</description>
  <maintainer email="user@todo.todo">user</maintainer>
  <license>TODO: License declaration</license>
  <buildtool_depend>ament_cmake</buildtool_depend>

  # Add the following three lines to package.xml
  #########################
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>
  #########################

  # Add the following one line if the message contains type *time*
  #########################
  <depend>builtin_interfaces</depend>
  #########################


  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Update CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(awesome_interfaces)

# 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_package(ament_cmake REQUIRED)

# Add the following 1 line if your message depends on type *time*
########################################
find_package(builtin_interfaces REQUIRED)
########################################

# Add the following lines to CMakeLists.txt
########################################
find_package(rosidl_default_generators REQUIRED)
rosidl_generate_interfaces(${PROJECT_NAME}
    "msg/AwesomeName.msg"

    # Add the following if your message depends on type *time*
    DEPENDENCIES builtin_interfaces
)
```

```
########################################

ament_export_dependencies(rosidl_default_runtime)

ament_package()
```

With all above changes, your message package should hopefully build. Run the following from `colcon_ws/src/awesome_driver` to make sure it builds:

```
colcon build
```

3. Create Python package for the driver

Create package structure

Inside *colcon_ws/src/awesome_driver*, create a ROS2 Python package: `bash ros2 pkg create awesome_driver --build-type ament_python`
This creates a simple Python package in *awesome_driver* folder. So now, your *colcon_ws/src/awesome_driver* folder should look like this:

```
├── awesome_driver # Python package for driver
│   ├── package.xml
│   ├── awesome_driver
│   │   └── __init__.py
│   ├── resource
│   │   └── awesome_driver
│   ├── setup.cfg
│   ├── setup.py
│   └── test
│       ├── test_copyright.py
│       ├── test_flake8.py
│       └── test_pep257.py
├── awesome_interfaces # C++ package for msg
│   ├── CMakeLists.txt
│   ├── msg
│   │   └── AwesomeName.msg
│   └── package.xml
└── README.md
```

I didn't actually need the test files, so I deleted the entire test folder. But this is up to you...

Make the driver node

Inside *colcon_ws/src/awesome_driver/awesome_driver/awesome_driver* (Yes, three times 😂, in the same folder as the *__init__.py* file), write your driver ROS2 node. Check the CTD node for reference.

Let's assume that your driver node is called `parse_awesome.py`. For ROS2 to find your driver node, you need to update the `setup.py` file at *colcon_ws/src/awesome_driver/awesome_driver/setup.py*:

```
    entry_points={
        'console_scripts': [
        # add entry point to your driver
        # name_of_the_executable = package_name.python_filename:main
            'awesome_driver = awesome_driver.parse_awesome:main',
        ],
    },
```

Now, you can run `colcon build` to make sure that this node builds as well :)

How to test the driver's data parsing?

I initially thought that the sensors stream the data as String, like in the old CTD parser. In that case, checking the data correctness is pretty simple: you need to make sure that your node can listen to *input_topic* and publish to *output_topic*. There is example data in the old parser, e.g. example CTD string data.

To test this:

1. In one terminal, run your driver node: `ros2 run awesome_driver awesome_driver`
2. In a second terminal, listen to the output topic: `ros2 topic echo /output/topic/name`
3. In a third terminal, try to publish the example string into *input_topic*: `ros2 topic pub --once /input/topic/name std_msgs/msg "{data: 'example-data-string'}"
4. Make sure that the second terminal prints out correct output_topic info

I then realized that the String messages are actually published by the corresponding serial readers, e.g. the serial reader for CTD. This is trickier to test, but I think as long as we have tested the in and output messages as above, we can leave the serial reader test to when we can connect to the driver.

For the serial port changes, see examples in this commit for CTD. Alternatively, you can choose to incorporate the serial port readings into the parser itself, i.e. instead of listening to *input_topic*, you parse directly from the serial port data. I'm not sure what is to be preferred. For easy of testing, I opted for separating the two.

Resources

- Write a Minimal ROS2 Python Node - The Robotics Back-End
- ROS2 Create Custom Message (Msg/Srv) - The Robotics Back-End