Create a ROS2 package for Both Python and Cpp Nodes

In ROS2, when you create a package you have to select a build type: either ament_cmake or ament_python. This will determine whether your package is a Cpp package, or a Python package.

But… How to do if you want to create a ROS2 package containing both Python nodes and Cpp nodes?

Good news for you: it's possible, and in this tutorial I'll show you how to create and configure your Python and Cpp nodes in the same package.

First things first, make sure you already know how to setup a standard ROS2 Python package, and a ROS2 Cpp package.

>> Watch this video as an additional resource to this article:

You want to learn ROS2 efficiently?

*Check out **ROS2 For Beginners** and learn ROS2 step by step, in 1 week.*

After watching the video, subscribe to the Robotics Back-End Youtube channel so you don't miss the next tutorials!

Table of Contents

1. How we'll manage to setup Python and Cpp nodes in the same ROS2 package
2. Setup your ROS2 Cpp and Python package
    2.1. Create a standard Cpp package
    2.2. Add a Cpp node + header

2.3. Add a Python node + module to import
2.4. ROS2 Package architecture with both Python and Cpp nodes – final
3. Configure your ROS2 package for both Cpp and Python
3.1. package.xml
3.2. CMakeLists.txt
3.3. Compile and run your ROS2 Cpp and Python nodes
4. Going further

How we'll manage to setup Python and Cpp nodes in the same ROS2 package

First we'll create a ROS2 Cpp package, which contains a package.xml and CMakeLists.txt. For Cpp related stuff, nothing will be too different from what you're already used to do.

For Python, we'll add some folders and files so you an add your Py nodes inside the package. And we'll configure the Python stuff… in the CMakeLists.txt. This is one of the main difference you'll have from a standard Python package. No more setup.py and setup.cfg, everything will be done in the CMakeLists.txt.

Note: the method I propose here is not necessarily the only one. You have many different ways to achieve that. First follow this tutorial to get something working, and then modify the architecture/configuration as you need.

Setup your ROS2 Cpp and Python package

The name of the package for this tutorial will be "my_cpp_py_pkg".

Create a standard Cpp package

First create the package with the ament_cmake build type .

```
$ cd ~/ros2_ws/src/
$ ros2 pkg create my_cpp_py_pkg --build-type ament_cmake
```

For now the package contains those files:

```
my_cpp_py_pkg/
├── CMakeLists.txt
├── include
│   └── my_cpp_py_pkg
├── package.xml
└── src
```

This is the base for a Cpp package. Later on, from that base we'll add necessary files and configuration for Python.

Add a Cpp node + header

Let's add a .cpp file in the src/ directory, and an .hpp header file in the include/my_cpp_py_pkg/ directory. No need to create new folders here.

```
$ cd my_cpp_py_pkg/
$ touch src/cpp_node.cpp
$ touch include/my_cpp_py_pkg/cpp_header.hpp
```

Note: if your cpp_node.cpp file needs to include the cpp_header.hpp file you'll have to write `#include "my_cpp_py_pkg/cpp_header.hpp"`.

Note 2: you have to have a main() in your cpp_node otherwise the compilation step won't succeed. You can just add a minimal C++ node if you're following this tutorial by the letter.

Add a Python node + module to import

Let's setup the package so it can also have Python nodes.

```
$ mkdir my_cpp_py_pkg
$ touch my_cpp_py_pkg/__init__.py
$ mkdir scripts
```

The first folder we create has the same name as the package. Inside this folder we create an empty __init__.py file. This is something that is already present when you create a Python package.

This folder will host any library and module we want to use or export.

Then we create a scripts/ folder. In there we'll place our executables (and nodes).

Now that we have the structure, let's add one Python module and one node.

```
$ touch my_cpp_py_pkg/module_to_import.py
$ touch scripts/py_node.py
```

Important: you have to add a shebang line first thing in the py_node.py file:

```
1. #!/usr/bin/env python3
2. ...
```

This is something you don't have to do with a standard ROS2 Python package (it's managed for you), but with this setup if you don't add this line, you will get an error when you try to start the node with `ros2 run` or from a launch file.

Also (and we'll come back to it later), if you want to be able to modify your code and re-run it without recompiling every time, or if you simply want to start your node by launching your script directly, then make the script executable: `chmod +x scripts/py_node.py`.

Note: if you want to import the module_to_import.py file from your py_node.py file (or from any other file from other packages), you'll have to write `from my_cpp_py_pkg.module_to_import import ...`.

ROS2 Package architecture with both Python and Cpp nodes – final

By now your package should look like this:

```
my_cpp_py_pkg/
# --> package info, configuration, and compilation
├── CMakeLists.txt
├── package.xml
# --> Python stuff
├── my_cpp_py_pkg
│   ├── __init__.py
│   └── module_to_import.py
├── scripts
│   └── py_node.py
# --> Cpp stuff
├── include
│   └── my_cpp_py_pkg
│       └── cpp_header.hpp
└── src
    └── cpp_node.cpp
```

The CMakeLists.txt and package.xml will be used for both Python and Cpp nodes. For the rest, you can see that Cpp stuff is clearly separated from Python stuff.

Now, don't forget to write something to run in the Python and Cpp files, and after that let's configure the package!

Configure your ROS2 package for both Cpp and Python

package.xml

```
1. <?xml version="1.0"?>
2. <?xml-model href="http://download.ros.org/schema/package_format3.xsd"
   schematypens="http://www.w3.org/2001/XMLSchema"?>
3. <package format="3">
```

```
  <name>my_cpp_py_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="your@email.com">Name</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>ament_cmake_python</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>rclpy</depend>

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

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

First of all, if you ever need to share your package with other people, or publish it on the Internet, don't forget to modify the version, description, maintainer (+ author if needed), and license tags.

Here is what we added:

```
  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>ament_cmake_python</buildtool_depend>
```

By default you should already have a buildtool_depend tag for ament_cmake, since that's what we asked when creating the package from command line.

Here we add another buildtool_depend tag: ament_cmake_python.

Note: in a standard Python package, you'd have ament_python, not ament_cmake_python. Make sure not to mix those 2. Using ament_cmake_python means that we'll be able to setup our Python stuff with cmake, so, from the CMakeLists.txt file.

```
  <depend>rclcpp</depend>
  <depend>rclpy</depend>
```

We add a dependency for the ROS2 Cpp library (rclcpp) as well as the ROS2 Python library (rclpy).

That's it for package.xml.

CMakeLists.txt

Here's the complete CMakeLists.txt to install both Cpp and Python nodes. I have removed some stuff we don't use here ("Default to C99", and the test section).

```
cmake_minimum_required(VERSION 3.5)
project(my_cpp_py_pkg)

# 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(ament_cmake_python REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclpy REQUIRED)

# Include Cpp "include" directory
include_directories(include)

# Create Cpp executable
add_executable(cpp_executable src/cpp_node.cpp)
ament_target_dependencies(cpp_executable rclcpp)

# Install Cpp executables
install(TARGETS
  cpp_executable
  DESTINATION lib/${PROJECT_NAME}
)

# Install Python modules
ament_python_install_package(${PROJECT_NAME})

# Install Python executables
install(PROGRAMS
  scripts/py_node.py
  DESTINATION lib/${PROJECT_NAME}
)

ament_package()
```

Basically, we can split this config into 3 parts: dependencies, Cpp part, and Python part.

```
# Find dependencies
find_package(ament_cmake REQUIRED)
find_package(ament_cmake_python REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclpy REQUIRED)
```

We import external dependencies for both Cpp and Python at the same time.

```
# Include Cpp "include" directory
include_directories(include)
```

```
3.
4.  # Create Cpp executable
5.  add_executable(cpp_executable src/cpp_node.cpp)
6.  ament_target_dependencies(cpp_executable rclcpp)
7.
8.  # Install Cpp executables
9.  install(TARGETS
10.   cpp_executable
11.   DESTINATION lib/${PROJECT_NAME}
12. )
```

This is the Cpp part.

First we include the “include” directory so the cpp_header.hpp file can be found.

Then it’s business as usual: you create an executable, link with dependencies, and install the executable in the lib/ folder of your package (inside the install/ folder of your ROS2 workspace).

```
1.  # Install Python modules
2.  ament_python_install_package(${PROJECT_NAME})
3.
4.  # Install Python executables
5.  install(PROGRAMS
6.    scripts/py_node.py
7.    DESTINATION lib/${PROJECT_NAME}
8.  )
```

And this is the Python part, which should be new to you if you’ve always installed ROS2 Python nodes from setup.py (which does not exist here).

First, use `ament_python_install_package(${PROJECT_NAME})` to install any Python module (in this example: files under my_cpp_py_pkg/ folder), so you can find them from this – or another – package.

Then, we install the scripts/py_node.py file. We put this file in the install lib/ folder, which is the same folder as for ROS2 Cpp nodes. Thus all Cpp/Python executables will be at the same place.

For any new Python script you need to install, just add a new line here. After being copied, the “installed” file will be automatically made executable.

Compile and run your ROS2 Cpp and Python nodes

Let’s compile our ROS2 Python/Cpp package.

```
$ cd ~/ros2_ws/
$ colcon build --packages-select my_cpp_py_pkg
```

Open 2 new terminals, source the ROS2 environment, and you can start both Cpp and Python nodes.

```
$ ros2 run my_cpp_py_pkg cpp_executable
```

```
$ ros2 run my_cpp_py_pkg py_node.py
```

Important: if you wish to compile with the `--symlink-install` option (so you can modify and re-run a script without having to re-compile), you'll have to make your scripts executable with `chmod +x`. Otherwise when you try to run your node you'll get this error :"No executable found".

Going further

In this tutorial you've seen how to create and setup a ROS2 package for both Python and Cpp nodes.

As you can see it's not that difficult, and as long as you clearly separate your Python/Cpp files, as well as the Python/Cpp configuration, nothing should go wrong.

If you wish to install other things in this package, such as launch files, YAML config files, etc., then you just need to install them as you would in a standard Cpp package.

ROS2 Tutorials

>> Learn ROS2 in 1 Week <<

Want to learn ROS2?

ROS2 For Beginners
Learn ROS2 in 1 Week