Integrating launch files into ROS 2 packages

Goal: Add a launch file to a ROS 2 package

Tutorial level: Intermediate

Time: 10 minutes

Contents

Prerequisites

You should have gone through the tutorial on how to create a ROS 2 package.

As always, don't forget to source ROS 2 in every new terminal you open.

Background

In the previous tutorial, we saw how to write a standalone launch file. This tutorial will show how to add a launch file to an existing package, and the conventions typically used.

Tasks

1 Create a package

Create a workspace for the package to live in:

Linux | macOS | Windows

```
mkdir -p launch_ws/src
cd launch_ws/src
```

Python package | C++ package

```
ros2 pkg create --build-type ament_python --license Apache-2.0 py_launch_example
```

2 Creating the structure to hold launch files

By convention, all launch files for a package are stored in the `launch` directory inside of the package. Make sure to create a `launch` directory at the top-level of the package you created above.

Python package | C++ package

For Python packages, the directory containing your package should look like this:

```
src/
  py_launch_example/
    launch/
    package.xml
    py_launch_example/
    resource/
    setup.cfg
    setup.py
    test/
```

To enable colcon to locate and utilize our launch files, we need to inform Python's setup tools of their presence. To achieve this, open the `setup.py` file, add the necessary `import` statements at the top, and include the launch files into the `data_files` parameter of `setup`:

```
import os
from glob import glob
# Other imports ...

package_name = 'py_launch_example'

setup(
    # Other parameters ...
    data_files=[
        # ... Other data files
        # Include all launch files.
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.
[pxy][yma]*')))
    ]
)
```

3 Writing the launch file

Python launch file | XML launch file | YAML launch file

Inside your `launch` directory, create a new launch file called `my_script_launch.py`. `_launch.py` is recommended, but not required, as the file suffix for Python launch files. However, the launch file name needs to end with `launch.py` to be recognized and autocompleted by `ros2 launch`.

Your launch file should define the `generate_launch_description()` function which returns a `launch.LaunchDescription()` to be used by the `ros2 launch` verb.

```
import launch
import launch_ros.actions

def generate_launch_description():
    return launch.LaunchDescription([
        launch_ros.actions.Node(
            package='demo_nodes_cpp',
            executable='talker',
            name='talker'),
  ])
```

4 Building and running the launch file

Go to the top-level of the workspace, and build it:

```
colcon build
```

After the `colcon build` has been successful and you've sourced the workspace, you should be able to run the launch file as follows:

Python package | C++ package

Python launch file | XML launch file | YAML launch file

```
ros2 launch py_launch_example my_script_launch.py
```

Documentation

The launch documentation provides more details on concepts that are also used in `launch_ros`.

Additional documentation/examples of launch capabilities are forthcoming. See the source code (https://github.com/ros2/launch and https://github.com/ros2/launch_ros) in the meantime.