Using `colcon` to build packages

Table of Contents

Goal: Build a ROS 2 workspace with `colcon`.

Tutorial level: Beginner

Time: 20 minutes

This is a brief tutorial on how to create and build a ROS 2 workspace with `colcon`. It is a practical tutorial and not designed to replace the core documentation.

Background

`colcon` is an iteration on the ROS build tools `catkin_make`, `catkin_make_isolated`, `catkin_tools` and `ament_tools`. For more information on the design of colcon see this document.

The source code can be found in the colcon GitHub organization.

Prerequisites

Install colcon

Linux | macOS | Windows

```
sudo apt install python3-colcon-common-extensions
```

Install ROS 2

To build the samples, you will need to install ROS 2.

Follow the installation instructions.

> **Attention**
>
> If installing from Debian packages, this tutorial requires the desktop installation.

Basics

A ROS workspace is a directory with a particular structure. Commonly there is a `src` subdirectory. Inside that subdirectory is where the source code of ROS packages will be located. Typically the directory starts otherwise empty.

colcon does out of source builds. By default it will create the following directories as peers of the `src` directory:

- The `build` directory will be where intermediate files are stored. For each package a subfolder will be created in which e.g. CMake is being invoked.
- The `install` directory is where each package will be installed to. By default each package will be installed into a separate subdirectory.
- The `log` directory contains various logging information about each colcon invocation.

> **Note**
>
> Compared to catkin there is no `devel` directory.

Create a workspace

First, create a directory (`ros2_ws`) to contain our workspace:

Linux | macOS | Windows

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
```

At this point the workspace contains a single empty directory `src`:

```
.
└── src

1 directory, 0 files
```

Add some sources

Let's clone the examples repository into the `src` directory of the workspace:

```
git clone https://github.com/ros2/examples src/examples -b humble
```

Now the workspace should have the source code to the ROS 2 examples:

```
.
└── src
    └── examples
        ├── CONTRIBUTING.md
        ├── LICENSE
        ├── rclcpp
        ├── rclpy
        └── README.md

4 directories, 3 files
```

Source an underlay

It is important that we have sourced the environment for an existing ROS 2 installation that will provide our workspace with the necessary build dependencies for the example packages. This is achieved by sourcing the setup script provided by a binary installation or a source installation, ie. another colcon workspace (see Installation). We call this environment an **underlay**.

Our workspace, `ros2_ws`, will be an **overlay** on top of the existing ROS 2 installation. In general, it is recommended to use an overlay when you plan to iterate on a small number of packages, rather than putting all of your packages into the same workspace.

Build the workspace

> **Attention**
>
> To build packages on Windows you need to be in a Visual Studio environment, see Building the ROS 2 Code for more details.

In the root of the workspace, run `colcon build`. Since build types such as `ament_cmake` do not support the concept of the `devel` space and require the package to be installed, colcon supports the option `--symlink-install`. This allows the installed files to be changed by changing the files in the `source` space (e.g. Python files or other non-compiled resources) for faster iteration.

Linux | macOS | Windows

```
colcon build --symlink-install
```

After the build is finished, we should see the `build`, `install`, and `log` directories:

```
.
├── build
├── install
├── log
└── src

4 directories, 0 files
```

Run tests

To run tests for the packages we just built, run the following:

Linux | macOS | Windows

```
colcon test
```

Source the environment

When colcon has completed building successfully, the output will be in the `install` directory. Before you can use any of the installed executables or libraries, you will need to add them to your path and library paths. colcon will have generated bash/bat files in the `install` directory to help set up the environment. These files will add all of the required elements to your path and library paths as well as provide any bash or shell commands exported by packages.

Linux | macOS | Windows

```
source install/setup.bash
```

Try a demo

With the environment sourced, we can run executables built by colcon. Let's run a subscriber node from the examples:

```
ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
```

In another terminal, let's run a publisher node (don't forget to source the setup script):

```
ros2 run examples_rclcpp_minimal_publisher publisher_member_function
```

You should see messages from the publisher and subscriber with numbers incrementing.

Create your own package

colcon uses the `package.xml` specification defined in REP 149 (format 2 is also supported).

colcon supports multiple build types. The recommended build types are `ament_cmake` and `ament_python`. Also supported are pure `cmake` packages.

An example of an `ament_python` build is the ament_index_python package , where the setup.py is the primary entry point for building.

A package such as demo_nodes_cpp uses the `ament_cmake` build type, and uses CMake as the build tool.

For convenience, you can use the tool `ros2 pkg create` to create a new package based on a template.

> **Note**
>
> For `catkin` users, this is the equivalent of `catkin_create_package`.

Setup `colcon_cd`

The command `colcon_cd` allows you to quickly change the current working directory of your shell to the directory of a package. As an example `colcon_cd some_ros_package` would quickly bring you to the directory `~/ros2_ws/src/some_ros_package`.

Linux | macOS | Windows

```
echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc
echo "export _colcon_cd_root=/opt/ros/humble/" >> ~/.bashrc
```

Depending on the way you installed `colcon_cd` and where your workspace is, the instructions above may vary, please refer to the documentation for more details. To undo this in Linux and macOS, locate your system's shell startup script and remove the appended source and export commands.

Setup `colcon` tab completion

The command `colcon` supports command completion for bash and bash-like shells if the `colcon-argcomplete` package is installed.

Linux | macOS | Windows

```
echo "source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash" >> ~/.bashrc
```

Depending on the way you installed `colcon` and where your workspace is, the instructions above may vary, please refer to the documentation for more details. To undo this in Linux and macOS, locate your system's shell startup script and remove the appended source command.

Tips

- If you do not want to build a specific package place an empty file named `COLCON_IGNORE` in the directory and it will not be indexed.
- If you want to avoid configuring and building tests in CMake packages you can pass: `--cmake-args -DBUILD_TESTING=0`.
- If you want to run a single particular test from a package:

```
colcon test --packages-select YOUR_PKG_NAME --ctest-args -R YOUR_TEST_IN_PKG
```