You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Iron.

Creating a workspace

Goal: Create a workspace and learn how to set up an overlay for development and testing.

Tutorial level: Beginner

Time: 20 minutes

Contents

Background

A workspace is a directory containing ROS 2 packages. Before using ROS 2, it’s necessary to source your ROS 2 installation workspace in the terminal you plan to work in. This makes ROS 2’s packages available for you to use in that terminal.

You also have the option of sourcing an “overlay” - a secondary workspace where you can add new packages without interfering with the existing ROS 2 workspace that you’re extending, or “underlay”. Your underlay must contain the dependencies of all the packages in your overlay. Packages in your overlay will override packages in the underlay. It’s also possible to have several layers of underlays and overlays, with each successive overlay using the packages of its parent underlays.

Prerequisites

- ROS 2 installation
- colcon installation
- git installation
- turtlesim installation
- Have rosdep installed
- Understanding of basic terminal commands (here's a guide for Linux)
- Text editor of your choice

Tasks

1 Source ROS 2 environment

Your main ROS 2 installation will be your underlay for this tutorial. (Keep in mind that an underlay does not necessarily have to be the main ROS 2 installation.)

Depending on how you installed ROS 2 (from source or binaries), and which platform you're on, your exact source command will vary:

Linux macOS Windows

```
source /opt/ros/humble/setup.bash
```

Consult the installation guide you followed if these commands don't work for you.

2 Create a new directory

Best practice is to create a new directory for every new workspace. The name doesn't matter, but it is helpful to have it indicate the purpose of the workspace. Let's choose the directory name `ros2_ws`, for "development workspace":

Linux macOS Windows

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

Another best practice is to put any packages in your workspace into the `src` directory. The above code creates a `src` directory inside `ros2_ws` and then navigates into it.

3 Clone a sample repo

Ensure you're still in the `ros2_ws/src` directory before you clone.

In the rest of the beginner developer tutorials, you will create your own packages, but for now you will practice putting a workspace together using existing packages.

If you went through the Beginner: CLI Tools tutorials, you'll be familiar with `turtlesim`, one of the packages in ros_tutorials.

A repo can have multiple branches. You need to check out the one that targets your installed ROS 2 distro. When you clone this repo, add the `-b` argument followed by that branch.

In the `ros2_ws/src` directory, run the following command:

```
git clone https://github.com/ros/ros_tutorials.git -b humble
```

Now `ros_tutorials` is cloned in your workspace. The `ros_tutorials` repository contains the `turtlesim` package, which we'll use in the rest of this tutorial. The other packages in this repository are not built because they contain a `COLCON_IGNORE` file.

So far you have populated your workspace with a sample package, but it isn't a fully-functional workspace yet. You need to resolve the dependencies first and then build the workspace.

4 Resolve dependencies

Before building the workspace, you need to resolve the package dependencies. You may have all the dependencies already, but best practice is to check for dependencies every time you clone. You wouldn't want a build to fail after a long wait only to realize that you have missing dependencies.

From the root of your workspace (`ros2_ws`), run the following command:

Linux macOS Windows

```
# cd if you're still in the ``src`` directory with the ``ros_tutorials`` clone
cd ..
rosdep install -i --from-path src --rosdistro humble -y
```

If you installed ROS 2 on Linux from source or the “fat” archive, you will need to use the rosdep command from their installation instructions. Here are the from-source rosdep section and the “fat” archive rosdep section.

If you already have all your dependencies, the console will return:

```
#All required rosdeps installed successfully
```

Packages declare their dependencies in the package.xml file (you will learn more about packages in the next tutorial). This command walks through those declarations and installs the ones that are missing. You can learn more about `rosdep` in another tutorial (coming soon).

5 Build the workspace with colcon

From the root of your workspace (`ros2_ws`), you can now build your packages using the command:

Linux macOS Windows

```
colcon build
```

The console will return the following message:

```
Starting >>> turtlesim
Finished <<< turtlesim [5.49s]

Summary: 1 package finished [5.58s]
```

Note

Other useful arguments for `colcon build`:

- `--packages-up-to` builds the package you want, plus all its dependencies, but not the whole workspace (saves time)
- `--symlink-install` saves you from having to rebuild every time you tweak python scripts
- `--event-handlers console_direct+` shows console output while building (can otherwise be found in the `log` directory)
- `--executor sequential` processes the packages one by one instead of using parallelism

Once the build is finished, enter the command in the workspace root (`~/ros2_ws`):

Linux macOS Windows

```
ls
```

And you will see that colcon has created new directories:

```
build  install  log  src
```

The `install` directory is where your workspace's setup files are, which you can use to source your overlay.

6 Source the overlay

Before sourcing the overlay, it is very important that you open a new terminal, separate from the one where you built the workspace. Sourcing an overlay in the same terminal where you built, or likewise building where an overlay is sourced, may create complex issues.

In the new terminal, source your main ROS 2 environment as the "underlay", so you can build the overlay "on top of" it:

Linux macOS Windows

```
source /opt/ros/humble/setup.bash
```

Go into the root of your workspace:

Linux macOS Windows

```
cd ~/ros2_ws
```

In the root, source your overlay:

Linux macOS Windows

```
source install/local_setup.bash
```

Note

Sourcing the `local_setup` of the overlay will only add the packages available in the overlay to your environment. `setup` sources the overlay as well as the underlay it was created in, allowing you to utilize both workspaces.

So, sourcing your main ROS 2 installation's `setup` and then the `ros2_ws` overlay's `local_setup`, like you just did, is the same as just sourcing `ros2_ws`'s `setup`, because that includes the environment of its underlay.

Now you can run the `turtlesim` package from the overlay:

```
ros2 run turtlesim turtlesim_node
```

But how can you tell that this is the overlay turtlesim running, and not your main installation's turtlesim?

Let's modify turtlesim in the overlay so you can see the effects:

- You can modify and rebuild packages in the overlay separately from the underlay.
- The overlay takes precedence over the underlay.

7 Modify the overlay

You can modify `turtlesim` in your overlay by editing the title bar on the turtlesim window. To do this, locate the `turtle_frame.cpp` file in `~/ros2_ws/src/ros_tutorials/turtlesim/src`. Open `turtle_frame.cpp` with your preferred text editor.

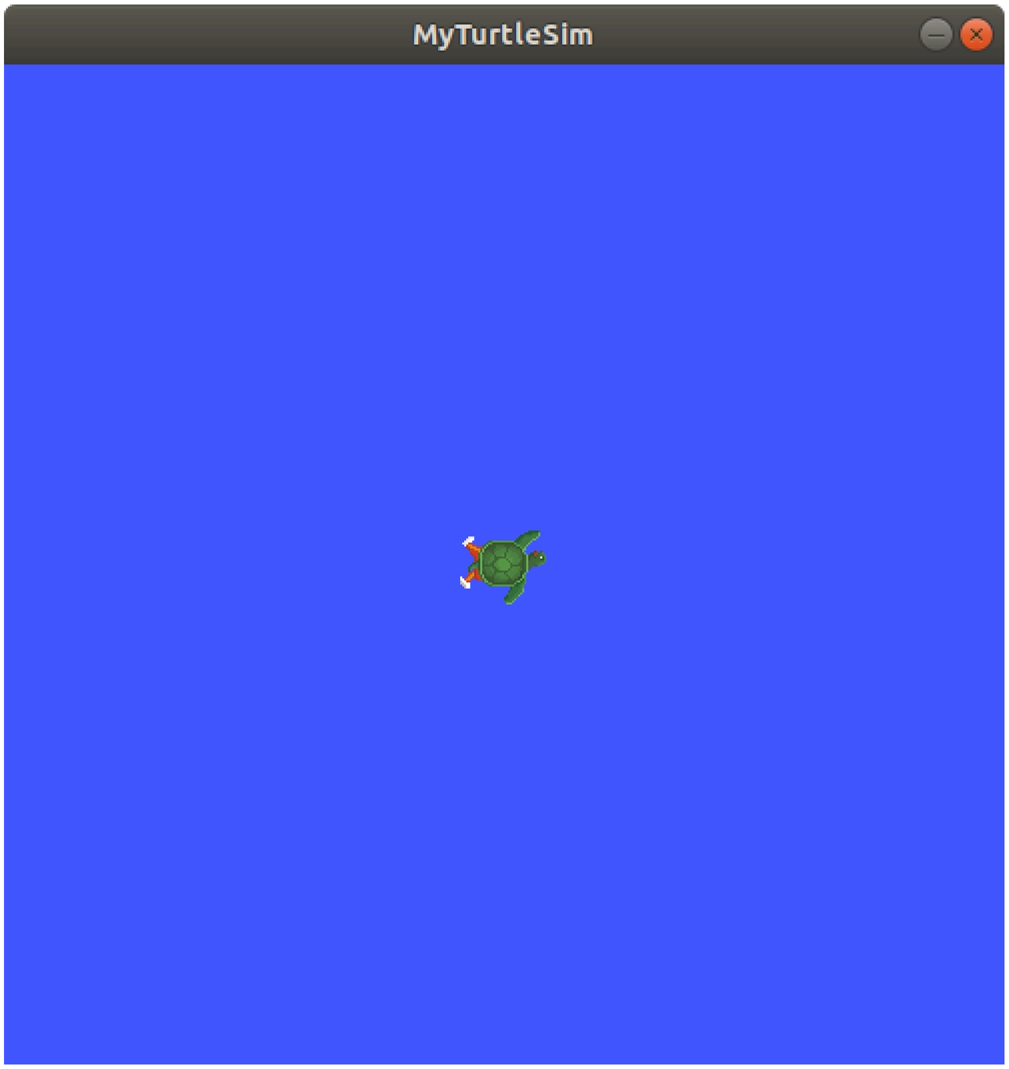
On line 52 you will see the function `setWindowTitle("TurtleSim");`. Change the value `"TurtleSim"` to `"MyTurtleSim"`, and save the file.

Return to the first terminal where you ran `colcon build` earlier and run it again.

Return to the second terminal (where the overlay is sourced) and run turtlesim again:

```
ros2 run turtlesim turtlesim_node
```

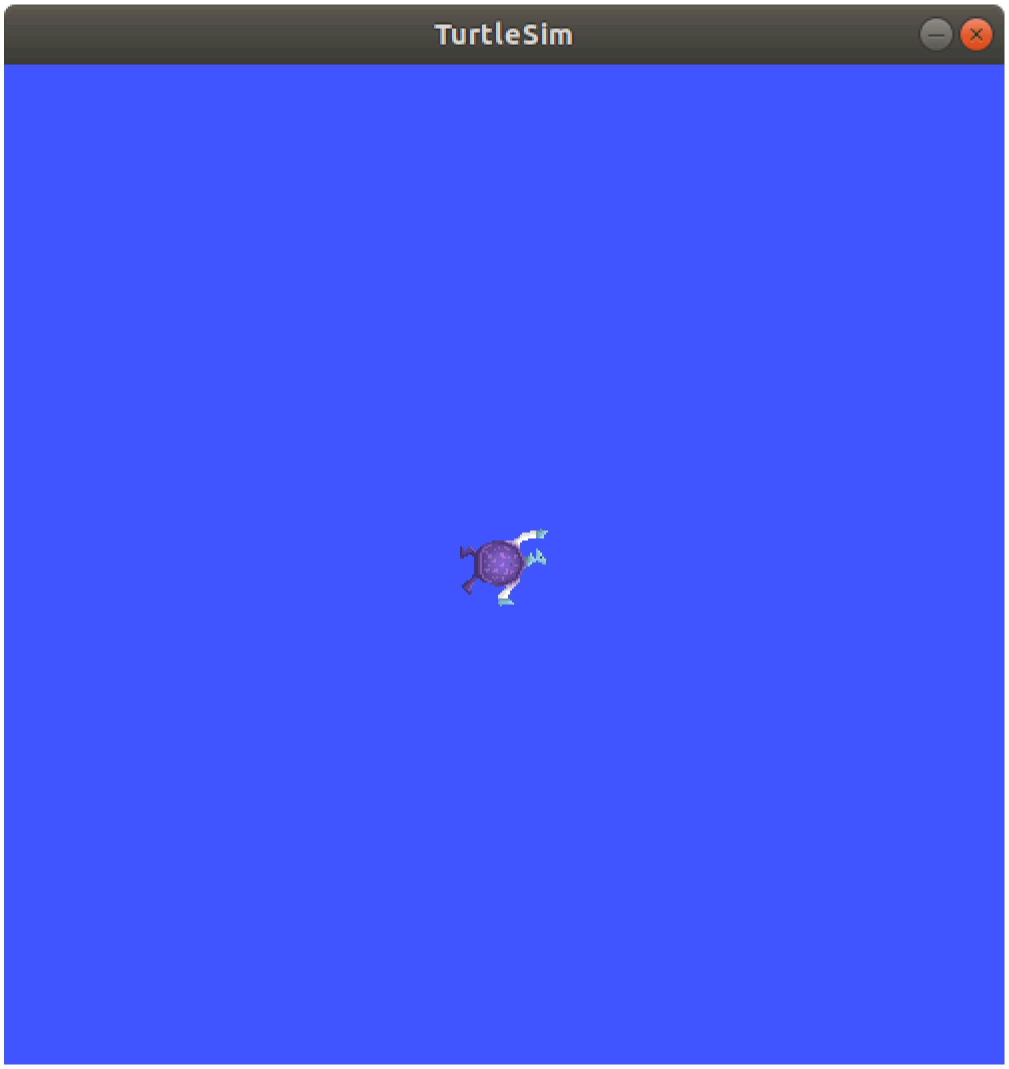
You will see the title bar on the turtlesim window now says "MyTurtleSim".



Even though your main ROS 2 environment was sourced in this terminal earlier, the overlay of your `ros2_ws` environment takes precedence over the contents of the underlay.

To see that your underlay is still intact, open a brand new terminal and source only your ROS 2 installation. Run turtlesim again:

```
ros2 run turtlesim turtlesim_node
```



You can see that modifications in the overlay did not actually affect anything in the underlay.

Summary

In this tutorial, you sourced your main ROS 2 distro install as your underlay, and created an overlay by cloning and building packages in a new workspace. The overlay gets prepended to the path, and takes precedence over the underlay, as you saw with your modified turtlesim.

Using overlays is recommended for working on a small number of packages, so you don't have to put everything in the same workspace and rebuild a huge workspace on every iteration.

Next steps

Now that you understand the details behind creating, building and sourcing your own workspace, you can learn how to create your own packages.