Catkin Workspaces

Before we actually create a package, the fundamental unit for organizing code in ROS, we should talk about catkin workspaces. A catkin workspace is a directory where you can modify, build, and install multiple independent catkin packages.

A Catkin workspace can contain four different spaces:

Source Space

The source space contains the source code for one or more catkin packages. This space should not be affected by configuration, build, or install steps. This space is located in the `src/` directory of the catkin workspace.

Build Space

The build space is where the catkin packages are built by CMake. Catkin and CMake place the results of the build process here like their cache information and other intermediate files. To build your catkin workspace run `catkin_make` in the root directory of your catkin workspace. You typically will not have to directly interact with anything in the build space. This space is located in the `build/` directory of the catkin workspace.

Development (devel) Space

After a sucessful build, the executable files will be placed in the development (devel) space. This is why we always tell you in lab to `source devel/setup.bash` after building with `catkin_make`, as it allows you to use the file you just built. The devel space is meant to act as a staging area where you can test your ROS nodes after building them, but in this class we tend to solely rely on the devel space rather than the install space mentioned in the next section. This space is located in the `devel/` directory of the catkin workspace.

Install Space

Finally after development is finished, if you choose to install your packages (which we rarely if ever do in this class), the built targets can be installed into the install space. This space is located in the

`install/` directory of the catkin workspace.

File Structure of a Catkin Workspace

The file structure of a catkin workspace is shown below:

```
catkin_workspace/
|
├── src/            # Source Space
|   ├── CMakeLists.txt
|   ├── package_1/
|   ├── package_2/
|   └── ...
|
├── build/          # Build Space
|   ├── CATKIN_IGNORE # Prevents catkin from walking this directory
|   └── ...
|
├── devel/          # Development Space
|   ├── setup.bash  # Setup script for devel space
|   └── ...
|
├── install/        # Install Space
|   └── ...
|
└── ...
```

Creating a Catkin Workspace

To create a catkin workspace, we first need to create a directory for the workspace, with a `/src` subdirectory. We will call this directory `catkin_ws`. We can then run `catkin_make` in the root of the caktin workspace to initialize the workspace which will create a CMakeLists.txt file in the `/src` folder. The commands to do this are shown below:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws
catkin_make
```

Now we should have our build, devel and install spaces. Now to “overlay” our workspace on top of the ROS enviroment/workspace we can `source devel/setup.bash`. This will add our workspace to the `ROS_PACKAGE_PATH` enviroment variable.

```
$ echo $ROS_PACKAGE_PATH
/home/cc/<your-semester>/class/ee106a-xxx/catkin_ws/src:/opt/ros/noetic/share
```

Now in our current enviroment, ROS will look in our workspace for a package before looking in the default ROS workspace.

Packages

In ROS, packages are the primary units for organizing software. A package may contain ROS nodes, libraries, ROS message definitions, ROS service definitions, launch files, and more.

Core Components of a Package

The following are essential pieces of a ROS package.

package.xml (Package Manifest)

The package.xml file is an XML file that describes a package. It provides information such as the package name, version, dependencies, and more.

CMakeLists.txt

We use a tool called catkin to build ROS packages. Catkin is ROS’s version of CMake, a tool used to build software. To build a workspace or package, Catkin needs a CMakeLists.txt file. This file is used to specify the dependencies of the package, the package name, the build flags, and other build options.

src/ (Source Files)

The src directory contains the source files for the package. This includes the source files for nodes, libraries, and other executables.

Filestructure of a Package

An example package structure is shown below (the optional files will be explained in later chapters):

```
erics_package/
│
├── CMakeLists.txt
│
├── package.xml
│
├── src/            # Source files (e.g. files for nodes)
│   ├── publisher_node.py
│   └── subscriber_node.py
│
├── scripts/        # Python scripts/nodes or other executable scripts
│   ├── python_node.py
│   └── utility_script.sh
│
├── msg/            # ROS message definitions (optional)
│   └── CustomMessage.msg
│
├── srv/            # ROS service definitions (optional)
│   └── CustomService.srv
│
├── launch/         # ROS launch files (optional)
│   └── start_nodes.launch
│
└── ...
```

Metapackages

Package-ception. A metapackage is a package that contains other packages. Very straightforward. The same rules for creating a package apply to metapackages with the addition condition that they have extra content in their package.xml file and CMakeLists.txt file as detailed here. An example metapackage structure is shown below:

```
erics_mega_metapackage/
│
├── CMakeLists.txt
│
```

```
├── package.xml
│
├── package_1/      # Package 1
│   ├── CMakeLists.txt
│   ├── package.xml
│   ├── src/
│   └── ...
│
├── package_2/      # Package 2
│   ├── CMakeLists.txt
│   ├── package.xml
│   ├── src/
│   └── ...
│
└── ...
```

Rules for Creating a Package

The following are the basic rules for creating a package:

- Packages must have a package.xml file
- Packages must have a CMakeLists.txt file which uses catkin

Ideally packages should also be placed in a catkin workspace, but packages can be built standalone as well.

Creating a Catkin Package

To follow proper convention, when creating a package when should place it in a catkin workspace. To create a package, we can use the `catkin_create_pkg <package_name> <dependency_1> ... <dependency_n>` command as follows:

```
# You should have created the catkin workspace previously
$ cd ~/catkin_ws/src
$ catkin_create_pkg erics_pkg std_msgs rospy roscpp
```

In this example, a package called `erics_pkg` will created with the dependencies `std_msgs`, `rospy`, and `roscpp`. The `catkin_create_pkg` command will create a `package.xml` file and a `CMakeLists.txt` file. The

`package.xml` file will be populated with the package name and dependencies specified. The `CMakeLists.txt` file will also be populated with the package name and dependencies specified.

Building a Catkin Package and Souring the Workspace

Now we can build the package(s) in the catkin workspace and source the setup file:

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

We can now use our newly created package(s)!

Navigation Commands

rospack

- `rospack find <package_name>`: Prints the path to a package
- `rospack depends <package_name>`: Prints the dependencies of a package (including the dependencies of the dependencies)
- `rospack depends1 <package_name>`: Prints the only first order (direct, rather than the dependencies of the dependecies) dependencies of a package
- `rospack list`: Lists all packages in the ROS_PACKAGE_PATH

roscd

- `roscd <package_name>`: Changes the current directory to the package directory

rosls

- `rosls <package_name>`: Lists the contents of the package directory

Tab Completion

If you have properly sourced your ROS workspace pressing tab[1] in the tmerinal after typing part of a package name will autocomplete the package name. If you can not autocomplete the

package name, then this is a sign that you have not properly sourced your ROS workspace.

1 If there are multiple autocompletions, pressing tab twice will show all possible autocompletions. ↩