ROS – Lecture 2

catkin build system
ROS packages
Building ROS nodes

catkin Build System

- catkin is the ROS build system
  - The set of tools that ROS uses to generate executable programs, libraries and interfaces
- The original ROS build system was rosbuild
  - Still used for older packages
- Implemented as custom CMake macros along with some Python code

catkin Workspace

- A set of directories in which a set of related ROS code lives
- You can have multiple ROS workspaces, but you can only work in one of them at any one time
- Contains the following spaces:

Source space	Contains the source code of catkin packages. Each folder within the source space contains one or more catkin packages.
Build Space	is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
Development (Devel) Space	is where built targets are placed prior to being installed
Install Space	Once targets are built, they can be installed into the install space by invoking the install target.

```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

ROS Development Setup

- Create a new catkin workspace
- Create a new ROS package
- Download and configure Eclipse
- Create Eclipse project file for your package
- Import package into Eclipse
- Write the code
- Update the make file
- Build the package

Creating a catkin Workspace

- Creating a Workspace Tutorial

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

- Initially, the workspace will contain only the top-level CMakeLists.txt
- Note that in the ready-made ROS Indigo VM you already have a catkin_ws workspace with the beginner_tutorials package

Building catkin Workspace

- **catkin_make** command builds the workspace and all the packages within it

```
cd ~/catkin_ws
catkin_make
```

ROS Package

- ROS software is organized into packages, each of which contains some combination of code, data, and documentation
- A ROS package is simply a directory inside a catkin workspace that has a package.xml file in it
- Packages are the most atomic unit of build and the unit of release
- A package contains the source files for one node or more and configuration files

Common Files and Directories

Directory	Explanation
include/	C++ include headers
src/	Source files
msg/	Folder containing Message (msg) types
srv/	Folder containing Service (srv) types
launch/	Folder containing launch files
package.xml	The package manifest
CMakeLists.txt	CMake build file

The Package Manifest

- package.xml defines properties of the package:
  - the package name
  - version numbers
  - authors
  - dependencies on other catkin packages
  - and more

The Package Manifest

- Example for a package manifest:

```
<package>
  <name>foo_core</name>
  <version>1.2.4</version>
  <description>
    This package provides foo capability.
  </description>
  <maintainer email="ivana@willowgarage.com">Ivana Bildbotz</maintainer>
  <license>BSD</license>

  <url>http://ros.org/wiki/foo_core</url>
  <author>Ivana Bildbotz</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>message_generation</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>

  <run_depend>message_runtime</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>

  <test_depend>python-mock</test_depend>
</package>
```

Creating a ROS Package

- Creating a ROS Package Tutorial
- Change to the source directory of the workspace

```
$ cd ~/catkin_ws/src
```

- **catkin_create_pkg** creates a new package with the specified dependencies

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

- For example, create a first_pkg package:

```
$ catkin_create_pkg first_pkg std_msgs rospy roscpp
```

Python First Node Example

```
1 #!/usr/bin/env python
2 # license removed for brevity
3 import rospy
4 from std_msgs.msg import String
5
6 def talker():
7     pub = rospy.Publisher('chatter', String, queue_size=10)
8     rospy.init_node('talker', anonymous=True)
9     rate = rospy.Rate(10) # 10hz
10    while not rospy.is_shutdown():
11        hello_str = "hello world %s" % rospy.get_time()
12        rospy.loginfo(hello_str)
13        pub.publish(hello_str)
14        rate.sleep()
15
16 if __name__ == '__main__':
17    try:
18        talker()
19    except rospy.ROSInterruptException:
20        pass
```

ROS Python Client Library

- **rospy** is a ROS client implementation.

- Library documentation can be found at:
  - http://wiki.ros.org/rospy

ROS Init

- A version of rospy.init_node() must be called before using any of the rest of the ROS system

- Typical call:

```
rospy.init_node("talker")
```

- Node names must be unique in a running system

ROS Rate

- A class to help run loops at a desired frequency.
- Specify the destired rate to run in Hz

```
rate = rospy.Rate(10)
```

- rate.sleep() method
  - Sleeps for any leftover time in a cycle.
  - Calculated from the last time sleep, reset, or the constructor was called

ROS shutdown

- Call **rospy.is_shutdown()** to check if the node should continue running

- rospy.is_shutdown() will return True if:
  - a SIGINT is received (Ctrl-C)
  - we have been kicked off the network by another node with the same name
  - all Node Handles have been destroyed

ROS Logging

- ROS_INFO prints an informative message
  - ROS_INFO("My INFO message.");
- All messages are printed with their level and the current timestamp
  - [INFO] [1356440230.837067170]: My INFO message.
- This function allows parameters as in printf:
  - ROS_INFO("My INFO message with argument: %f", val);
- ROS comes with five classic logging levels: DEBUG, INFO, WARN, ERROR, and FATAL
- Also, C++ STL streams are supported, e.g.:

```
ROS_INFO_STREAM("My message with argument: " << val);
```

ROS Logging

- ROS also automatically logs all messages that use ROS_INFO to log files on the filesystem for you so that you can go back and analyze a test later
  - Your node's log file will be in ~/.ros/log (defined by the ROS_LOG_DIR environment variable)

	Debug	Info	Warn	Error	Fatal
stdout	X	X			
stderr			X	X	X
log file	X	X	X	X	X
/rosout	X	X	X	X	X

Building Your Node

- Before building your node, you should modify the generated CMakeLists.txt in the package
- The following slide shows the changes that you need to make in order to create the executable for the node

CMakeLists.txt

To get ROS to generate the language-specific message code, we need to make sure that we tell the build system about the new message definitions. We can do this by adding these lines to our *package.xml* file:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

Next, we need to make a few changes to the *CMakeLists.txt* file. First, we need to add `message_generation` to the end of the `find_package()` call , so that `catkin` knows to look for the `message_generation` package:

```
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation   # Add message_generation here, after the other packages
)
```

Then we need to tell `catkin` that we're going to use messages at runtime, by adding `message_runtime` to the `catkin_package()` call:

```
catkin_package(
  CATKIN_DEPENDS message_runtime   # This will not be the only thing here
)
```

CMakeLists.txt (contd.)

```
add_message_files(
  FILES
  Complex.msg
)
```

Finally, still in the *CMakeLists.txt* file, we need to make sure the `generate_messages()` call is uncommented and contains all the dependencies that are needed by our messages:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

package.xml

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

Building Your Nodes

- To build the package in the terminal call catkin_make

Running the Node From Terminal

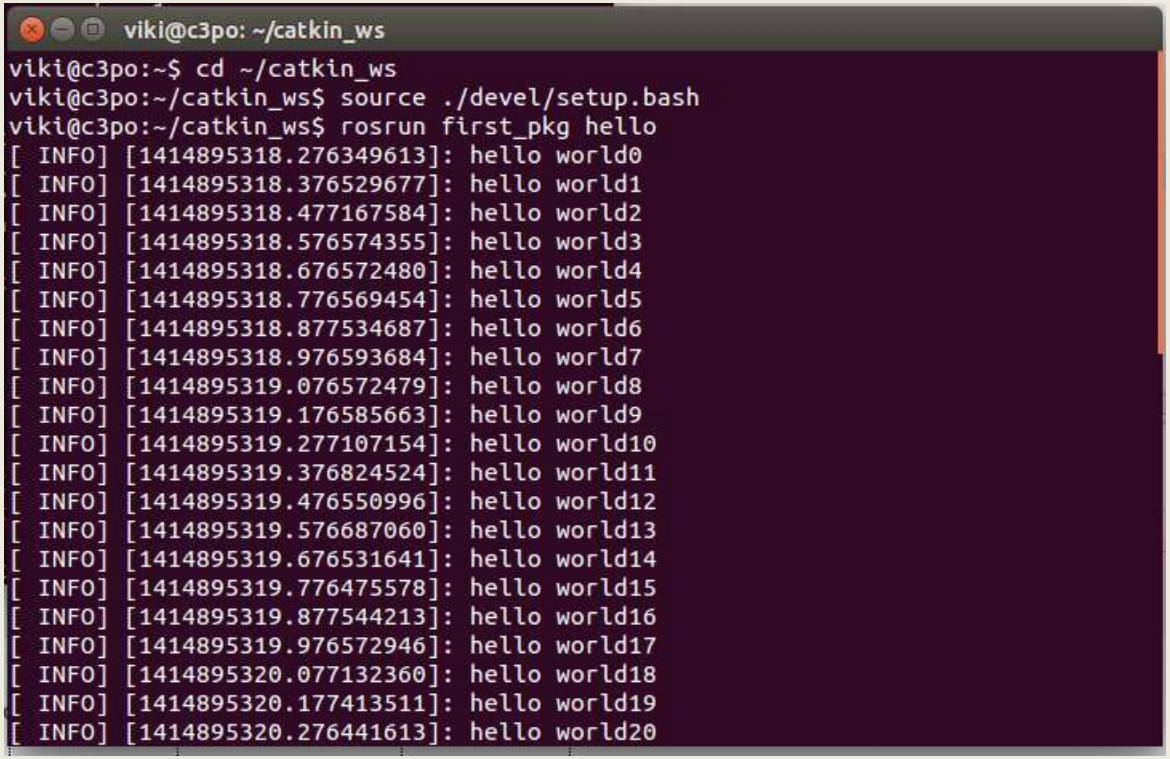
- Make sure you have sourced your workspace's setup.sh file after calling catkin_make:

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

  - Can add this line to your .bashrc startup file
- Now you can use rosrun to run your node:

```
$ rosrun first_pkg filename.py
```

Running the Node From Terminal



Ex. 2

- Create a new ROS package called "timer_package"
- Create a node in this package called "timer_node"
- The node should print to the console the current time every 0.5 second