ROS Applied Trainings

Robot Operating System (ROS) and Applications

Supported by ROSIN - ROS-Industrial Quality-Assured Robot Software Components.
More information: rosin-project.eu

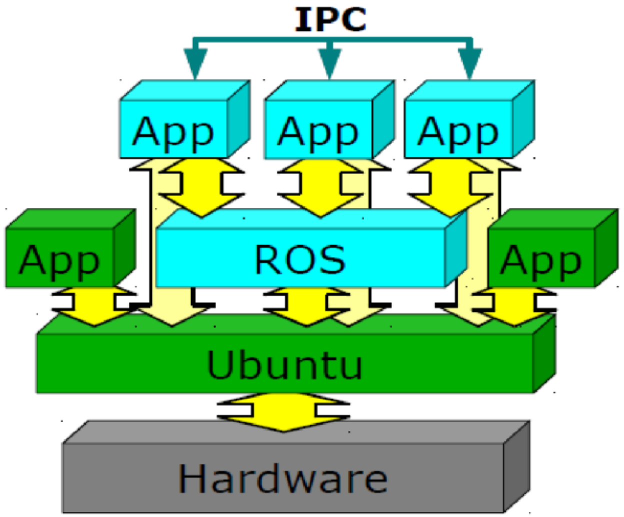
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 732287.

Annex_3_2_ROS_Training_eng.pptx

Content – 09.00-09.45

- ➢What is ROS – What is not?
- ➢Why should we use ROS?
- ➢ROS Sensors and companies using ROS
- ➢Setting up ROS
- ➢Linux basic comands

What is ROS – What is not?



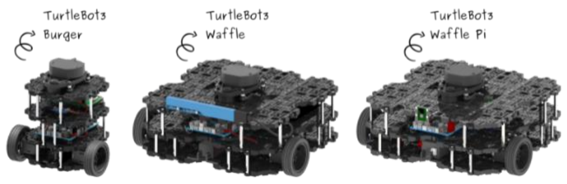
- Programming Language
- Library
- Operating System
- İntegrated development Environment
- It is a meta operating system for open source robots that runs services and various macros.

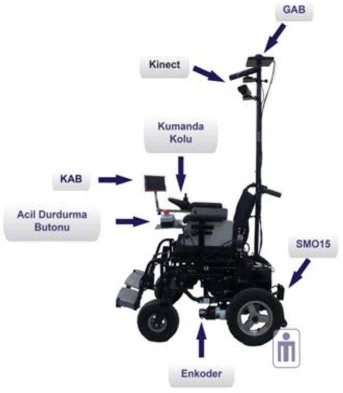
Why should we use ROS?

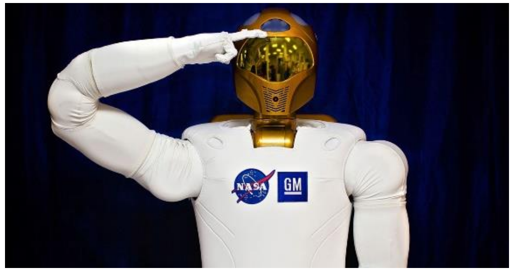
- Language independent structure (C++, Python, Lisp, Java, Lua)
- Modular run, parameters, messages and services allow instant intervention
- Systematic data transfer with Node/topic
- Driver support for many sensor, motor and robot platform
- Open source
- Algorithms, libraries and packages for mapping, localization and detection
- Active community
- Rapid testing
- Harware abstraction
- Visualizors

ROS Sensors

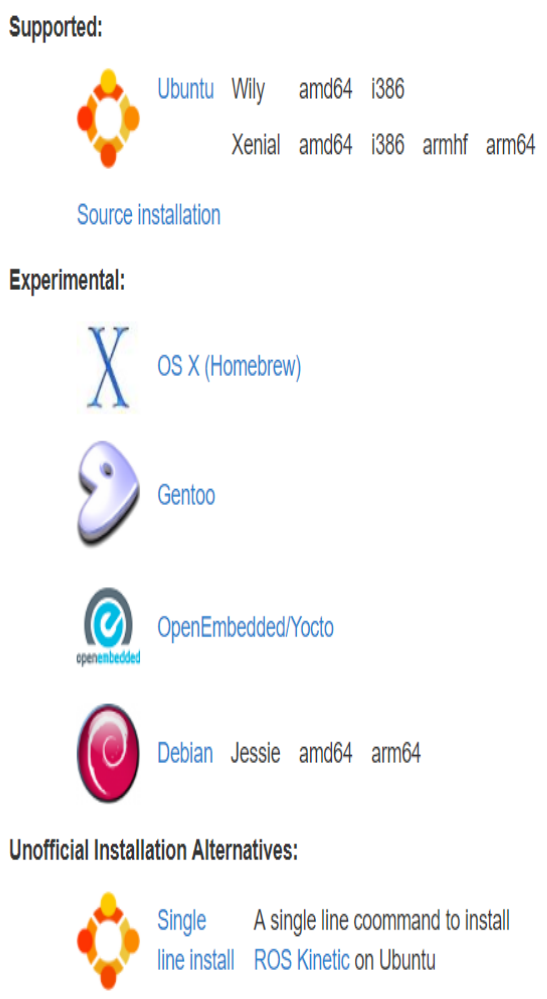
ROS using Companies







Setting up ROS– 1/2



Supported:

Ubuntu	Wily	amd64	i386		
	Xenial	amd64	i386	armhf	arm64

Source installation

Experimental:

OS X (Homebrew)

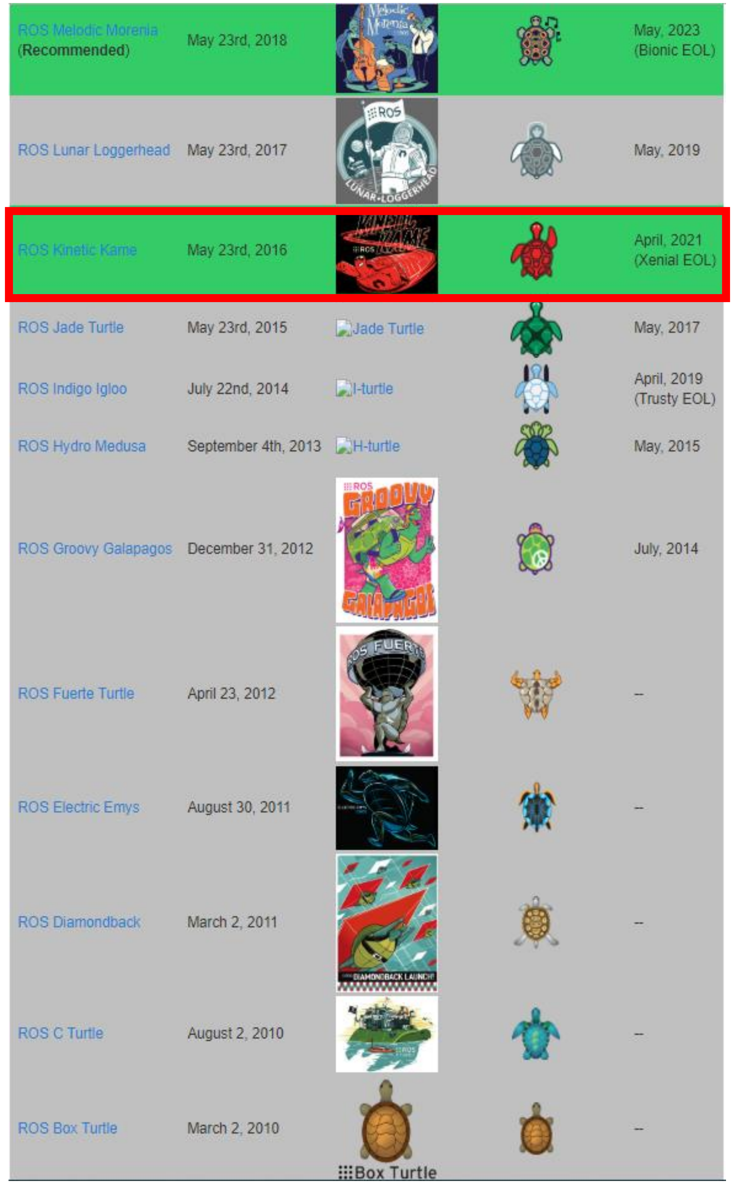
Gentoo

OpenEmbedded/Yocto

Debian Jessie amd64 arm64

Unofficial Installation Alternatives:

Single line install	A single line coommand to install ROS Kinetic on Ubuntu



ROS Melodic Morenia (Recommended)	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015	Jade Turtle		May, 2017
ROS Indigo Igloo	July 22nd, 2014	I-turtle		April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013	H-turtle		May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010	Box Turtle		--

Setting up ROS– 2/2

1. Setting up the computer to accept the software from settings.ros.org

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list‘
```

2. Setting up keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3. Making sure the Debian package is up to date

```
sudo apt-get update
```

4. Installing the full version(ROS,rqt,rviz,robot-generic libraries, 2D/3D simulators ...)

```
sudo apt-get install ros-kinetic-desktop-full
```

5. *Starting and updating rosdep*

```
sudo rosdep init
rosdep update
```

6. Environment Set-up

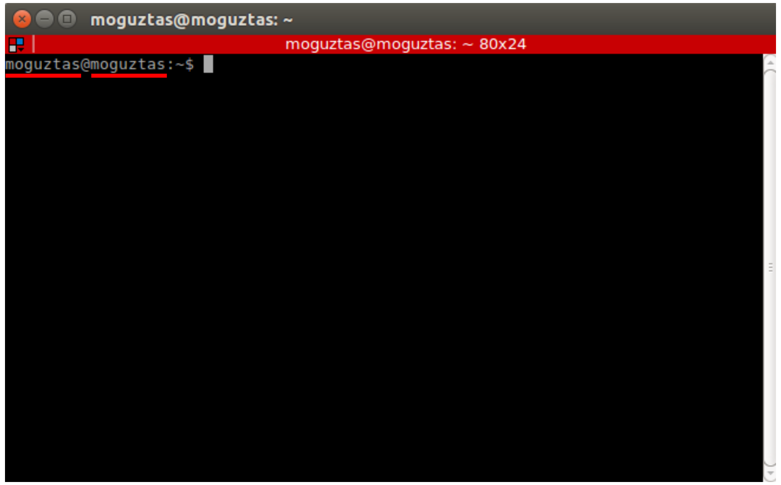
```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

7. Installing dependencies for compiler packages

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Linux Basic Commands– 1/4

- Openning new terminal: Ctrl + Alt + T
- moguztas@moguztas:
  - moguztas: user name
  - moguztas: computer name
- Copy: Ctrl + Shift + C
- Paste: Ctrl + Shift + V
- Terminating process: Ctrl + C



```
moguztas@moguztas:~$
```

Linux Basic Commands– 2/4

- **cd :** means «Change Directory». Enters the folder on the specified path.
- **ls :** means «List». Lists the files and documents in the entered folder.
- **mkdir <directory_name>:** means «Make Directory». Creates directory in given path.
- **mkdir -p <directory_path>/<dierctory_name>:** It realizes the folder creation process all the way. If there are non-existent folders, it creates nested folders.
- **rm <file_to_be_deleted>:** means «Remove». Removes specified directory..
- **rm -rf < file_to_be_deleted >:** means «Recursive Remove». deletes multiple files.
- **mv < file_to_be _moved> < directory_to_be _moved>:** Performs file transfer. Rename can also be done using the mv command.
- **cp < file_to_be _coppied> < folder_to_be _coppied>:** used for copying.
- **wget ‘<download_url>' :** It downloads the file specified on the internet to the folder on the computer.

Linux Basic Commands– 3/4

- **sudo apt-get install <package_name>:** Searches and installs the package on repository.
- **sudo apt-get remove <package_name>:** Searches for and deletes the package from the computer.
- **sudo apt-get update:** Retrieves the information of the repository stored in the sources.list file to the computer.
- **sudo apt-get upgrade:** Updates the packages on the computer.
- **apt-cache search <package_name>:** Searches and fetches the relevant package in repositories.
- **sudo chmod <permision_type> <file/directory_name>:** Gives file permissions to the relevant folder or document. Using 777 as the permission type means Read-Write-Execute.
- **sudo su:** allows doing operations as a super user.
- **sudo service <service_name> start :** Starts service running on Linux.
- **sudo service < service_name> stop :** Terminates service running on Linux.
- **sudo service < service_name> restart :** Restarts the service running on Linux.

Linux Basic Commands – 4/4

- **history:** Shows code history in terminal.
- **clear:** Deletes codes in the terminal.
- **ps :** means «Processes». Lists processes running on Linux system.
- **ps -aux | grep <process>:** Returns the specific process or processes running on Linux.
- **kill -9 <process_ID> :** Allows killing process running on Linux and having ID with command above.
- **udo lsusb :** Lists USB devices registered and running on Linux system.
- **cat :** Print the contents of a file on the terminal screen.
- **pwd :** Suppresses the path of a file to the terminal screen.
- **echo <variable> :** Allows printing of global variables and variables defined later on the Linux terminal to the screen.
- **<editör_name> <file> :** Allows editing a file on Linux.
- **setxkbmap tr :** maps the keyboard to use the layout determined by the options specified on the command line(tr= Turkish).
- **g++ hello_world.cpp -o hello_world :** Compiles C / C ++ files via GNU C Compiler.
- **./hello_world :** Executable file execution

Content – 10.00-11.45

- ROS Architecture Introduction
- File System - File System Level
  - Packages
  - Metapackages
  - Package Manifests
  - Message types
  - Service types
- Transaction Graph - Computation Graph Level
  - Nodes
  - Parameter Service
  - Messages
  - Topics
  - Services
  - Bags
  - Master
- Community Level
- Publisher-Subscriber ve Service Client Structures
- ROS Tools

ROS Architecture

ROS architecture is divided into 3 parts:

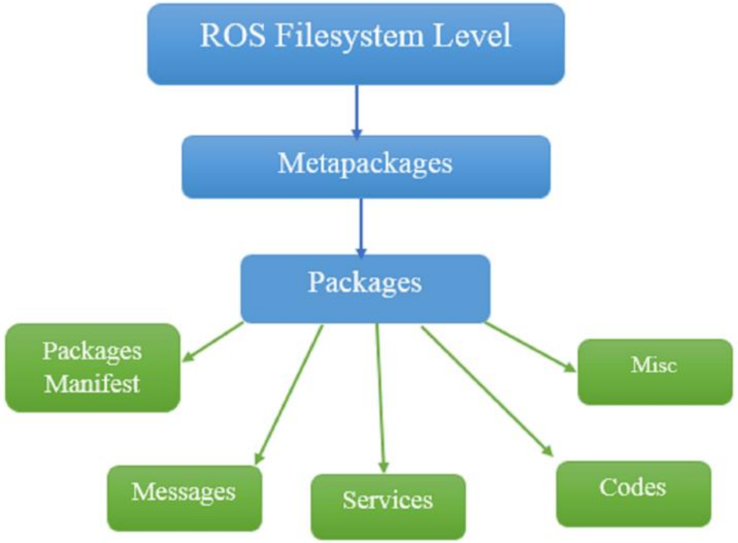
Filesystem Level: Contains a group concept to explain how ROS is internally created, the folder structure and the minimum number of files that should work.

Computation Graph Level: Contains concepts that explain how ROS uses communication between processes and systems.

Community Level: Includes a set of tools and concepts for sharing information, algorithms and code among developers.

Filesystem Level

- Packages
- Metapackages
- Package Manifests
- Message types
- Service types



Packages – 1/3

- A package can contain processes (nodes), ROS-linked libraries, datasets, configuration files, or anything else useful.
- The purpose of creating the packages is to divide the codes into small pieces and make them reusable.
- Packages are the main units that keep the software organized in ROS.
- Packages usually consist of typical files and folders:
  - **include / package_name /:** This directory contains the header files of the libraries we need.
  - **msg /:** If a message type other than the standard message types will be created, it must be created in this folder.
  - **Scripts /:** Codes written in script languages such as Python should be placed in this folder.
  - **src /:** Where the source files of the programs are located.
  - **srv /:** This is where the service files are located.
  - **CMakeLists.txt:** It is a CMake compilation (built) file containing the orders given to the compiler and many more building information.
  - **package.xml:** The package file of the packages

Packages – 2/3

ROS has tools that can help us create, edit, and work with packages:

- **rospack:** To find and learn about packages.
- **roscreate-pkg:** Creates a new package.
- **rosmake:** It is used to compile packages.
- **rosdep:** It is used to load dependencies of packages.
- **catkin_create_pkg:** Creates a new package.

ROS has a package called rosbash that allows us to move between packages and folders and files of packages. Some commands supported in the rosbash tool:

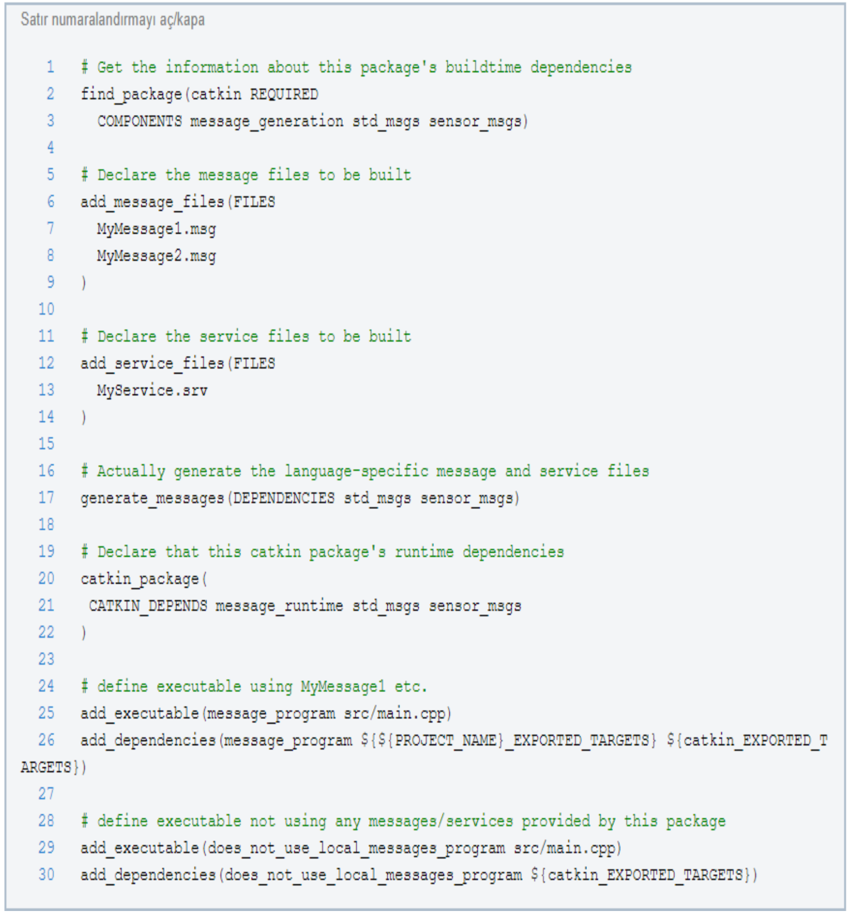
- **roscd:** Enables browsing between ROS directories.
- **rosed:** Enables editing files.
- **roscp:** Enables copying files from another package.
- **rosrun:** Enables executable files to run.
- **rosls:** It is used to list the files in the package.

Packages – 3/3

The package must contain the CMakeLists.txt file. This file tells catkin how and where to load the codes.

The CMakeLists.txt file should follow the format below, otherwise packages cannot be created correctly.

1. **Required CMake Version** (cmake_minimum_required)
2. **Package Name** (project())
3. **Find other CMake/Catkin packages needed for build** (find_package())
4. **Enable Python module support** (catkin_python_setup())
5. **Message/Service/Action Generators** (add_message_files(), add_service_files(), add_action_files())
6. **Invoke message/service/action generation** (generate_messages())
7. **Specify package build info export** (catkin_package())
8. **Libraries/Executables to build** (add_library()/add_executable()/target_link_libraries())
9. **Tests to build** (catkin_add_gtest())
10. **Install rules** (install())

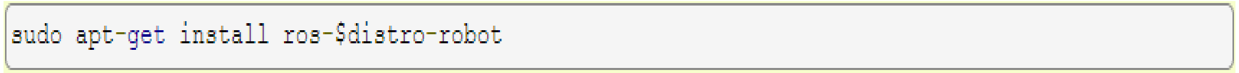


```
# Get the information about this package's buildtime dependencies
find_package(catkin REQUIRED
  COMPONENTS message_generation std_msgs sensor_msgs)

# Declare the message files to be built
add_message_files(FILES
  MyMessage1.msg
  MyMessage2.msg
)

# Declare the service files to be built
add_service_files(FILES
  MyService.srv
)

# Actually generate the language-specific message and service files
generate_messages(DEPENDENCIES std_msgs sensor_msgs)

# Declare that this catkin package's runtime dependencies
catkin_package(
 CATKIN_DEPENDS message_runtime std_msgs sensor_msgs
)

# define executable using MyMessage1 etc.
add_executable(message_program src/main.cpp)
add_dependencies(message_program ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

# define executable not using any messages/services provided by this package
add_executable(does_not_use_local_messages_program src/main.cpp)
add_dependencies(does_not_use_local_messages_program ${catkin_EXPORTED_TARGETS})
```

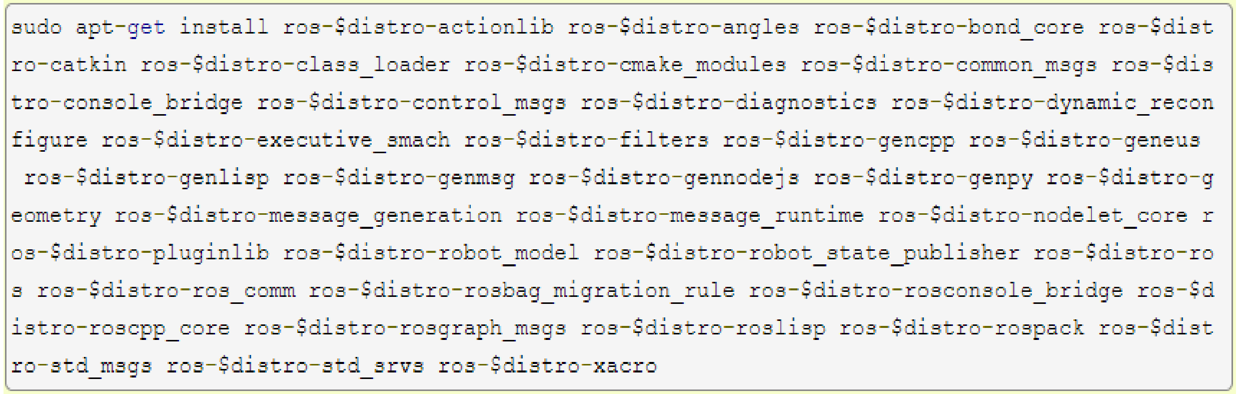
Metapackages

- Meta packages are used to run packages organized (simply group multiple packets).
- Meta packages are special packages in ROS that contain only the package.xml file.

Örnek: *robot* metapackage includes **packages:** [control_msgs, diagnostics, executive_smach, filters, geometry, joint_state_publisher, kdl_parser, kdl_parser_py, robot_state_publisher, urdf, urdf_parser_plugin, xacro]



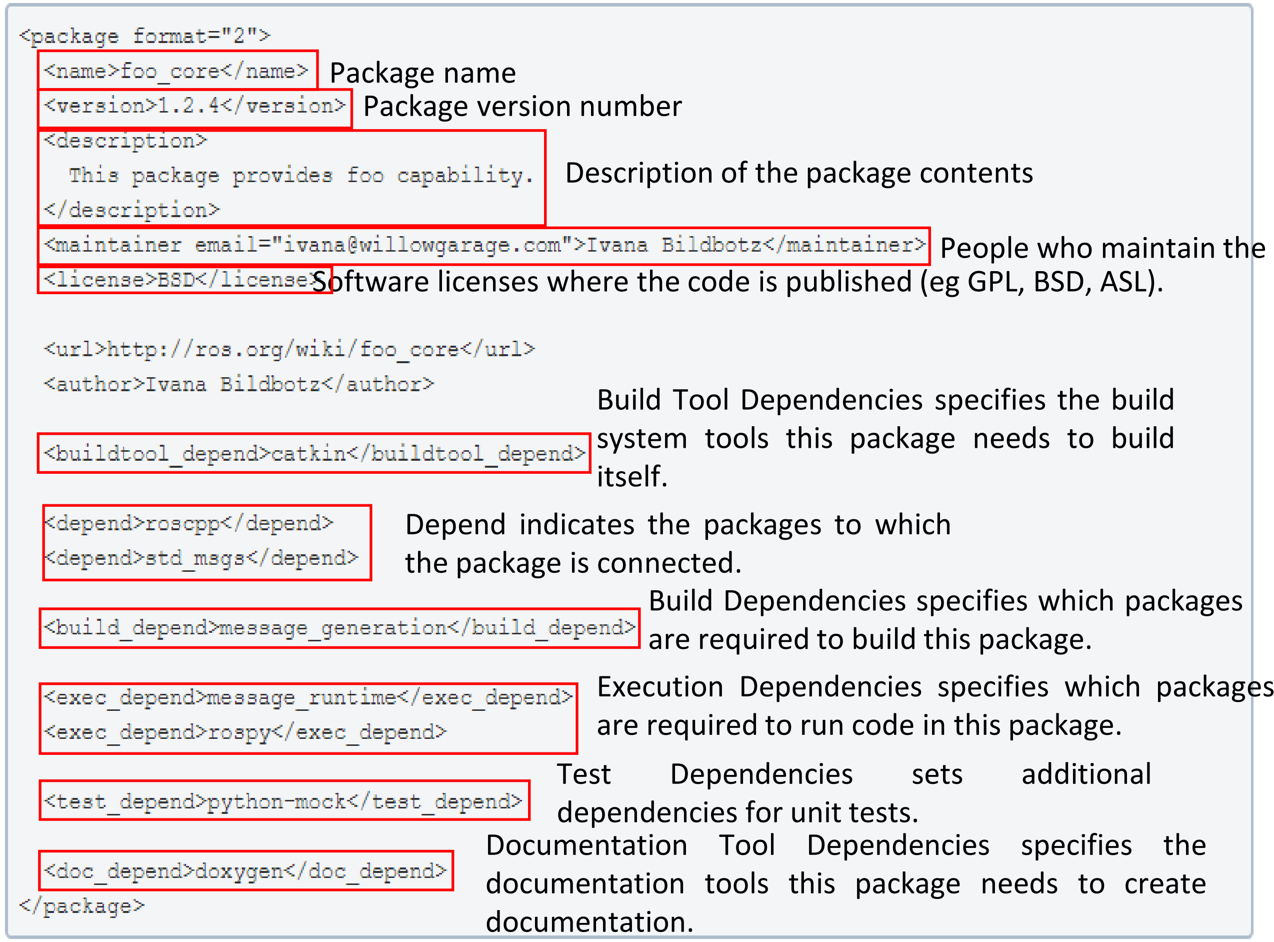
```
sudo apt-get install ros-$distro-robot
```



```
sudo apt-get install ros-$distro-actionlib ros-$distro-angles ros-$distro-bond_core ros-$dist
ro-catkin ros-$distro-class_loader ros-$distro-cmake_modules ros-$distro-common_msgs ros-$dis
tro-console_bridge ros-$distro-control_msgs ros-$distro-diagnostics ros-$distro-dynamic_recon
figure ros-$distro-executive_smach ros-$distro-filters ros-$distro-gencpp ros-$distro-geneus
 ros-$distro-genlisp ros-$distro-genmsg ros-$distro-gennodejs ros-$distro-genpy ros-$distro-g
eometry ros-$distro-message_generation ros-$distro-message_runtime ros-$distro-nodelet_core r
os-$distro-pluginlib ros-$distro-robot_model ros-$distro-robot_state_publisher ros-$distro-ro
s ros-$distro-ros_comm ros-$distro-rosbag_migration_rule ros-$distro-rosconsole_bridge ros-$d
istro-roscpp_core ros-$distro-rosgraph_msgs ros-$distro-roslisp ros-$distro-rospack ros-$dist
ro-std_msgs ros-$distro-std_srvs ros-$distro-xacro
```

Package Manifests

Package notifications **(package.xml)** is a file that contains other information about a package: its name, version, description, license information, dependencies, and exported packages. The reason why this file was created is to facilitate package loading and distribution.



```
<package format="2">
  <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>

  <depend>roscpp</depend>
  <depend>std_msgs</depend>

  <build_depend>message_generation</build_depend>

  <exec_depend>message_runtime</exec_depend>
  <exec_depend>rospy</exec_depend>

  <test_depend>python-mock</test_depend>

  <doc_depend>doxygen</doc_depend>
</package>
```

Package name

Package version number

Description of the package contents

People who maintain the

Software licenses where the code is published (eg GPL, BSD, ASL).

Build Tool Dependencies specifies the build system tools this package needs to build itself.

Depend indicates the packages to which the package is connected.

Build Dependencies specifies which packages are required to build this package.

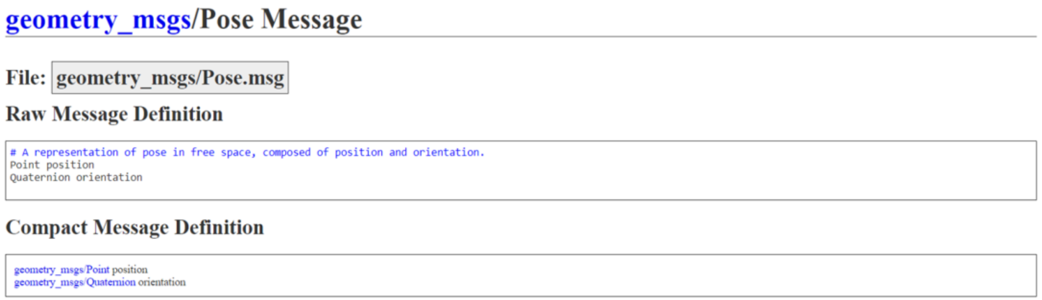
Execution Dependencies specifies which packages are required to run code in this package.

Test Dependencies sets additional dependencies for unit tests.

Documentation Tool Dependencies specifies the documentation tools this package needs to create documentation.

Message Types

The message file must be in the **msg /** folder and have the extension **.msg** (my_package / msg / MyMessageType.msg).



geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

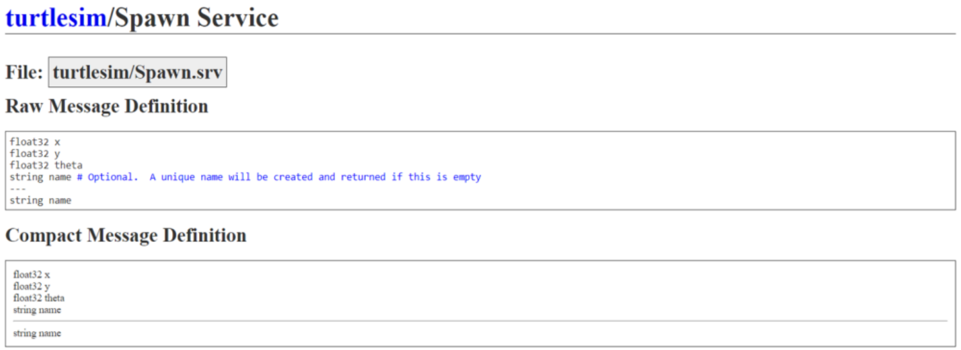
```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

Service Types

The service file must be in the **srv /** folder and have the extension **.srv** (my_package / srv / MyServiceType.srv).



turtlesim/Spawn Service

File: turtlesim/Spawn.srv

Raw Message Definition

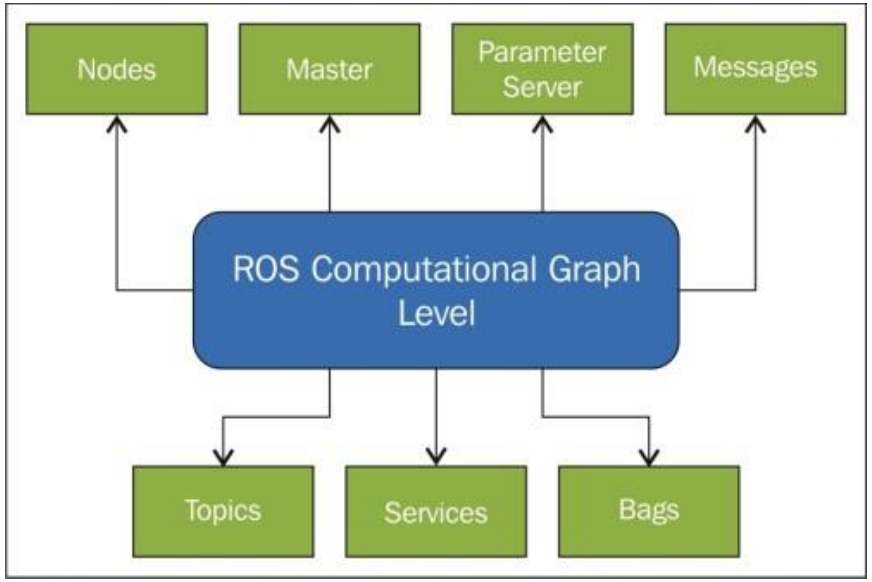
```
float32 x
float32 y
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
```

Compact Message Definition

```
float32 x
float32 y
float32 theta
string name
string name
```

Computation Graph Level

- ➢Nodes
- ➢Parameter Service
- ➢Messages
- ➢Topics
- ➢Services
- ➢Bags
- ➢Master



Nodes – 1/2

- Nodes are computations.
- A node can be written using different libraries such as roscpp for C ++ and rospy for Python.
- Using nodes in ROS gives us fault tolerance and simplifies the system and functions, separating the code and functions.
- A node must have a unique name in the system.
- A strong feature of ROS nodes is the ability to change parameters (node name, subject name, etc.) when starting the node. With the modification process, the node can be configured without recompiling the code, so it can be easily adapted to different scenarios. .
  - Example of changing the topic name in the node:

    rosrun book_tutorials tutorialX topic1:=/level1/topic1
  - Example of changing parameters in node:

    rosrun book_tutorials tutorialX _param: = 9.0
- ROS has another node type called nodelets. These special nodes are designed to run multiple nodes in a single operation. With this, nodes can communicate more efficiently without overloading the network. Nodelets are especially useful for camera systems and 3D sensors where the volume of data transferred is very high.

Not: Instead of having a large node that does everything in the system, it is more efficient to have many nodes that provide only one functionality.

Nodes – 2/2

ROS has the rosnode tool to process nodes and provide information. Some commands supported in the rosnode tool:

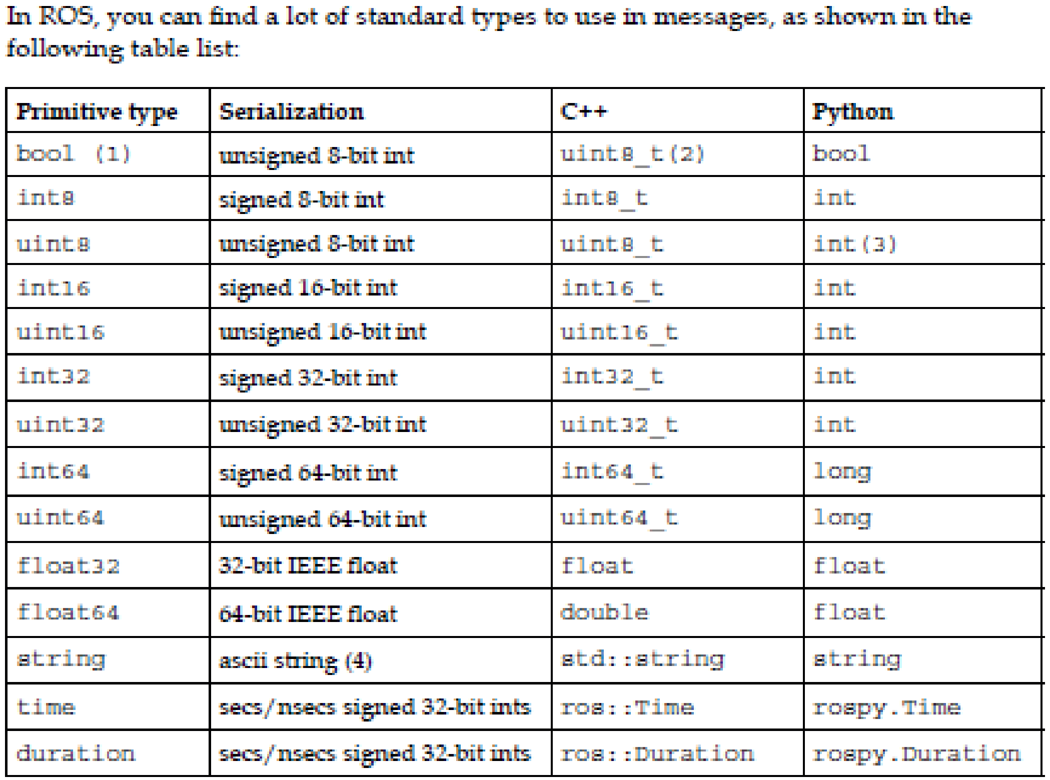
- **rosnode info node_name:** Prints information about the node.
- **rosnode kill node_name:** Terminates a running node.
- **rosnode list:** Lists active nodes.
- **rosnode machine hostname:** Lists the nodes running on a particular machine.
- **rosnode ping node_name:** Tests the connection to the node.

Parameter Service

- With parameters, it is possible to configure running nodes or change the operating parameters of a node.
- ROS has the rosparam tool to work with Parameter Server. Some commands supported in the rosparam tool:
  - **rosparam list:** Lists all parameters on the server.
  - **rosparam get parameter:** Gets the value of a parameter.
  - **rosparam set parameter: S**ets the value of a parameter.
  - **rosparam delete parameter:** Deletes a parameter.
  - **rosparam dump file:** Saves the parameter server in a file.
  - **rosparam load file:** Loads a file (with its parameters) on the parameter server..

Messages – 1/2

- Nodes communicate with each other through messages. A message contains data that provides information to other nodes..
- A message consists of two parts, **type** and **name**.
- We can create our own message type.

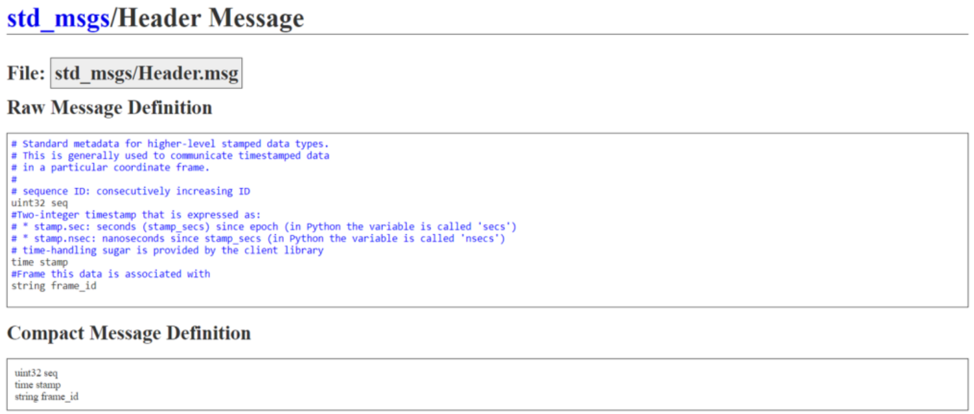


In ROS, you can find a lot of standard types to use in messages, as shown in the following table list:

Primitive type	Serialization	C++	Python
bool (1)	unsigned 8-bit int	uint8_t(2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int(3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	string
time	secs/nsecs signed 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

Messages – 2/2

Headers are a special type in ROS. The timeline has the numbering system and sequence number that let us know who the messages are coming from.



std_msgs/Header Message

File: std_msgs/Header.msg

Raw Message Definition

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

Compact Message Definition

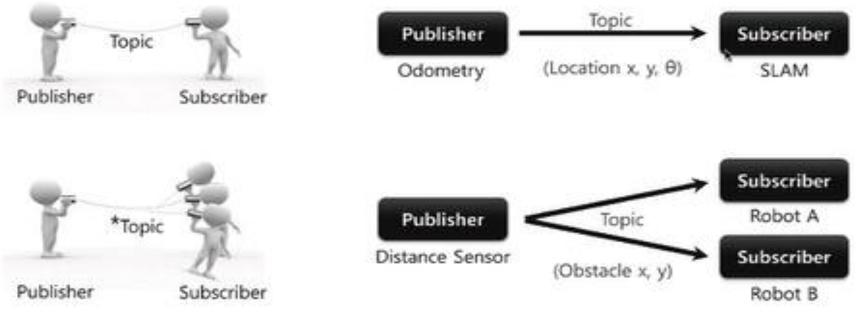
```
uint32 seq
time stamp
string frame_id
```

ROS has a tool called rosmsg that allows us to see the message definition and the source file where the message type is specified. Some commands supported in the rosmsg tool:

- **rosmsg show:** Displays the fields of this message.
- **rosmsg list:** Lists all posts.
- **rosmsg package:** Lists all messages in the particular package.
- **rosmsg packages:** Lists all packages with messages.
- **rosmsg users:** Searches for code files using the message type.

Topics – 1/2

- The messages are routed through a transport system with broadcast / subscribe semantics. A node sends a message by posting a specific subject.
- The subject is a name used to describe the content of the message.
- A node that deals with a particular type of data will subscribe to the appropriate topic.
- It is important that the subject names are unique to avoid confusion.
- You can have multiple concurrent publishers and subscribers for a single topic, and a single node can broadcast and / or subscribe to multiple topics.



Topics – 2/2

ROS has a tool to work on topics called *rostopic*. Some commands supported in the *rostopic* tool:

- **rostopic bw/topic:** Shows the bandwidth used by the topic.
- **rostopic echo/topic:** Print messages on the screen.
- **rostopic find message_type:** Find topics by type.
- **rostopic hz/topic:** Shows the publish rate of the topic.
- **rostopic info/topic:** Prints information about the topic, such as message type, publishers, and subscribers.
- **rostopic list:** Prints information on active topics.
- **rostopic pub/topic type args:** Publishes relevant data. It enables us to create and publish data directly from the command line on the topic we want.
- **rostopic type/topic:** Prints the subject type, that is, the type of message it posts.

Services

- The broadcasting / subscribing model is a very flexible communication paradigm, but many-to-many, one-way transportation is not generally suitable for request / response interactions desired in a distributed system.
- The request / response is done through services defined by a double message structure: one for request and the other for response.
- The provider provides a service under a name and uses a service by sending a customer request message and waiting for the answer.

It has *rossrv* and *rosservice* command line tools to work with ROS services. Some commands supported in these tools:

- **rosservice call/service args:** Calls the service with the given arguments.
- **rosservice find msg-type:** Finds service by service type.
- **rosservice info/service:** Prints information about the service.
- **rosservice list:** Lists active services.
- **rosservice type/servis:** Prints service type.

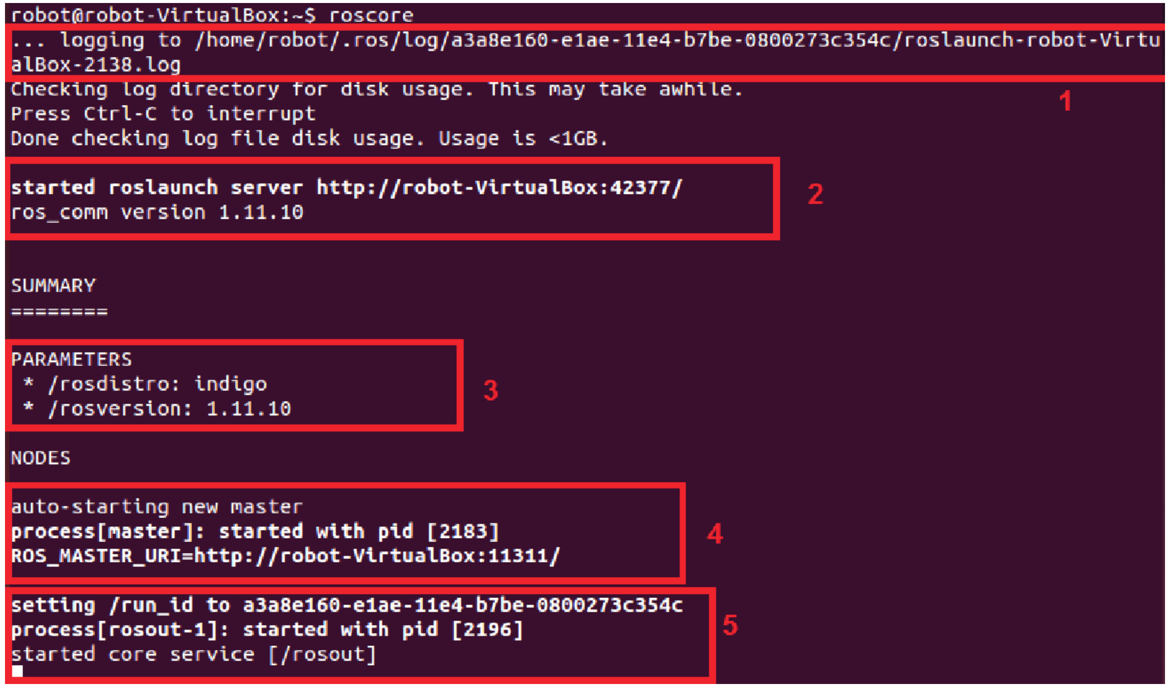
Bags

- The bag is a file created by ROS, created in .bag format to record all information of all messages, subjects, services and other information and then play it back.
- Bags are an important mechanism for storing data, such as sensor data, which can be difficult to collect but is required to develop and test algorithms.
- Tools that can be used in ROS to use bag files:
    - **rosbag:** Used to record, play and perform the requested data.
    - **rqt_bag:** It is used to visualize the data in graphic environment.

Master – 1/2

- The part of the nodes in the ROS that facilitates communication with each other is called the ROS master.
- ROS Master provides search to the rest of the Trading Chart. Without a master, the nodes cannot find each other, exchange messages, or call for service.
- Before operating any ROS node, we must start the ROS Master and ROS parameter server. We can start the ROS Master and ROS parameter server using a single command called *roscore*.

Master – 2/2

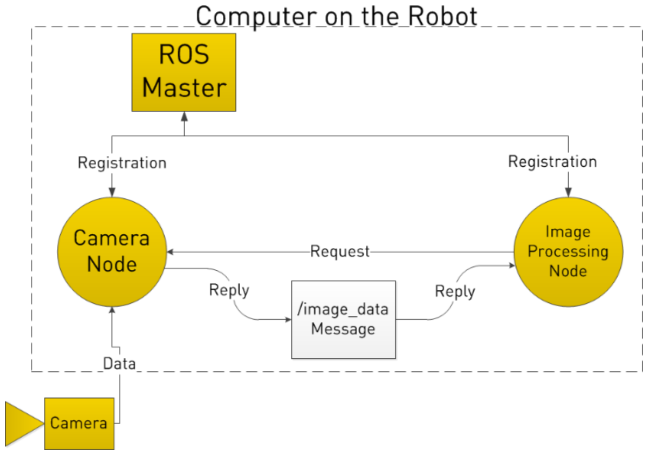


```
robot@robot-VirtualBox:~$ roscore
... logging to /home/robot/.ros/log/a3a8e160-e1ae-11e4-b7be-0800273c354c/roslaunch-robot-Virtu
alBox-2138.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://robot-VirtualBox:42377/
ros_comm version 1.11.10


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.10

NODES

auto-starting new master
process[master]: started with pid [2183]
ROS_MASTER_URI=http://robot-VirtualBox:11311/

setting /run_id to a3a8e160-e1ae-11e4-b7be-0800273c354c
process[rosout-1]: started with pid [2196]
started core service [/rosout]
```

- ○ In the first part, a log file is created inside the ~ / .ros / log folder to collect the logs from the ROS nodes. This file can be used for debugging purposes.
- ○ In the second part, a ROS initialization file called roscore is launched. This section shows the address of the ROS parameter server in the port.
- ○ In the third section, parameters such as rosdistro and rosversion are displayed.
- ○ In the fourth section, it is seen that the rosmaster node was started using ROS_MASTER_URI, which we previously defined as the environment variable.
- ○ In the fifth chapter, it is seen that the rosout node has started.

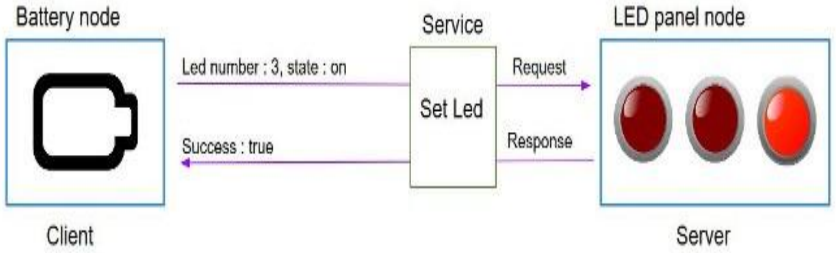
Publish-Subscribe

This communication model requires that the message be broadcast without the *publisher* explicitly specifying the recipients or having the knowledge of the intended recipients. The *subscriber* records the relevant ones from the published messages

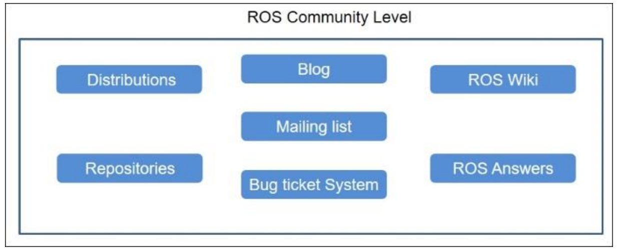


Service-Client

It is a communication model that provides one-time communication and the customer sends the request and the server returns a response. Used when the robot is asked to perform a special task (for example, from point A to point B).



Community Level

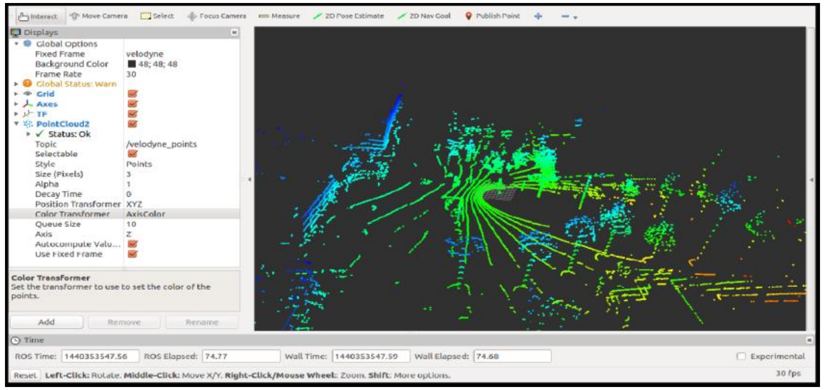


- **Dağıtımlar (Distributions):** ROS Distributions are collections of version stacks that you can load.
- **Depolar (Repositories):** ROS offers a code repository where different organizations can develop and publish their own robot software components.
- **ROS Wiki:** The main forum that documents information about ROS.
- **Mail Listesi:** The Ros-users mailing list is the primary communication channel, a forum that asks questions about the ROS software as well as new updates to ROS.
- **ROS Answers:** It is a question and answer site to answer your questions about ROS.
- **Blog:** http://www.ros.org/news , provides regular updates, including photos and videos.

ROS Tools

ROS has several GUI and command line tools to inspect and debug messages. Some of those:

- **rviz:** One of the 3D visualizers available in ROS to visualize 2D and 3D values from ROS topics and parameters.



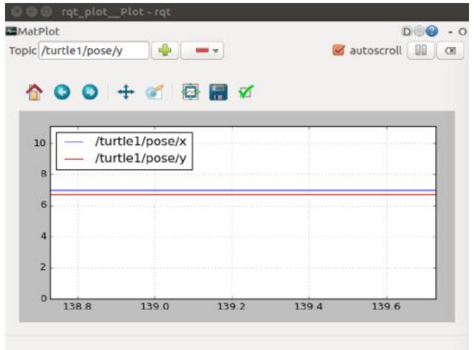
Fixed Frame velodyne
Background Color ■ 48; 48; 48
Frame Rate 30
Topic /velodyne_points
Selectable ☑
Style Points
Size (Pixels) 3
Alpha 1
Decay Time 0
Position Transformer XYZ
Color Transformer AxisColor
Queue Size 10
Axis Z
Autocompute Valu... ☑
Use Fixed Frame ☑

Color Transformer
Set the transformer to use to set the color of the points.

ROS Time: 1440353547.56 ROS Elapsed: 74.77 Wall Time: 1440353547.59 Wall Elapsed: 74.68

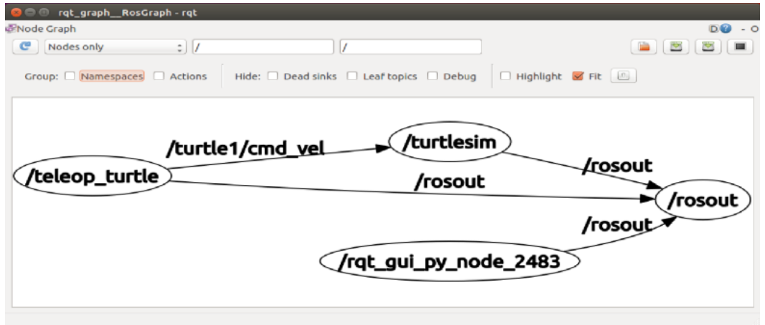
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

- **rqt_plot:** A tool for drawing scalar values in the form of ROS topics.



Topic /turtle1/pose/y
☑ autoscroll

- **rqt_graph:** Visualizes the connection graph between ROS nodes.



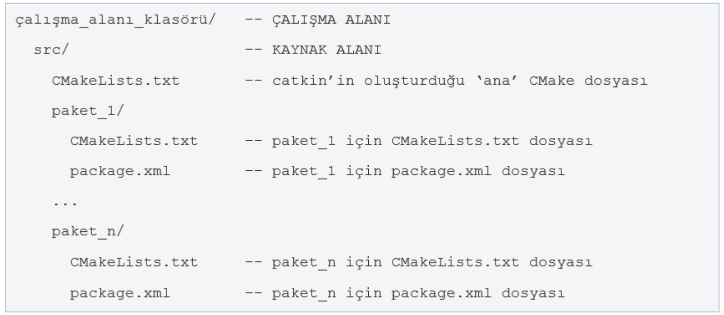
Nodes only
/
/
Group: ☐ Namespaces ☐ Actions
Hide: ☐ Dead sinks ☐ Leaf topics ☐ Debug
☐ Highlight ☑ Fit

Content – 13.00-14.45

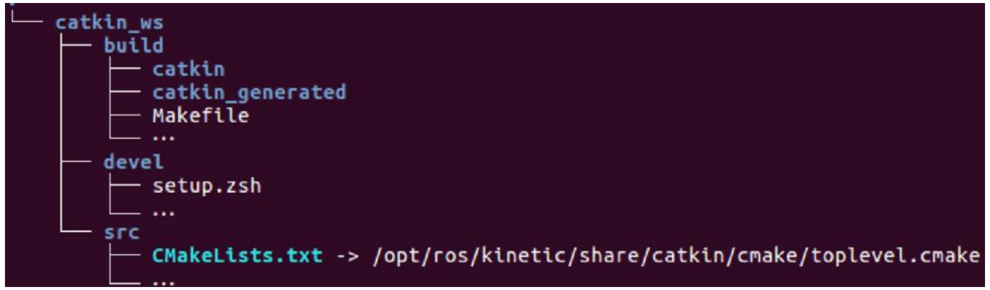
- ➢Application 1: Preparation of ROS Environment
- ➢Application 2: Creating a Catkin Package and Getting to Know the ROS Environment
- ➢Application 3: TurtleSim
- ➢Application 4: Creating Messages and Services
- ➢Application 5: Publisher-Subscirber Application
- ➢Application 6: Service-Client Application
- ➢Application 7: Saving and Playing Data

Application 1: Preparation of ROS Environment– 1/3

The workspace is a folder that contains packages. These packages contain source files. It is useful when various packages are wanted to be compiled (centralized) at the same time.



```
çalışma_alanı_klasörü/   -- ÇALIŞMA ALANI
  src/                   -- KAYNAK ALANI
    CMakeLists.txt       -- catkin’in oluşturduğu ‘ana’ CMake dosyası
    paket_1/
      CMakeLists.txt     -- paket_1 için CMakeLists.txt dosyası
      package.xml        -- paket_1 için package.xml dosyası
    ...
    paket_n/
      CMakeLists.txt     -- paket_n için CMakeLists.txt dosyası
      package.xml        -- paket_n için package.xml dosyası
```



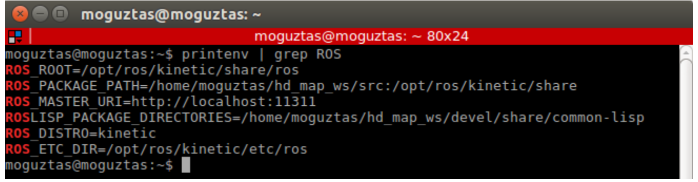
```
└── catkin_ws
    ├── build
    │   ├── catkin
    │   ├── catkin_generated
    │   ├── Makefile
    │   └── ...
    ├── devel
    │   ├── setup.zsh
    │   └── ...
    └── src
        ├── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake
        └── ...
```

- **Kaynak alan (src):** Resource area (src folder), packages, projects, etc. Placed. This area also contains the *CMakeLists.txt* file.
- **Derleme alanı (build):** stores cmake and catkin, cache information, configuration, and other buffer files for packages and projects in the build folder.
- **Geliştirme alanı (devel):** It is used to protect compiled programs and test programs without the installation phase..

Application 1: Preparation of ROS Environment– 2/3

1. Lets Check environment:

printenv | grep ROS



```
moguztas@moguztas:~$ printenv | grep ROS
ROS_ROOT=/opt/ros/kinetic/share/ros
ROS_PACKAGE_PATH=/home/moguztas/hd_map_ws/src:/opt/ros/kinetic/share
ROS_MASTER_URI=http://localhost:11311
ROSLISP_PACKAGE_DIRECTORIES=/home/moguztas/hd_map_ws/devel/share/common-lisp
ROS_DISTRO=kinetic
ROS_ETC_DIR=/opt/ros/kinetic/etc/ros
moguztas@moguztas:~$
```

2. In order not to make our configuration settings every time::

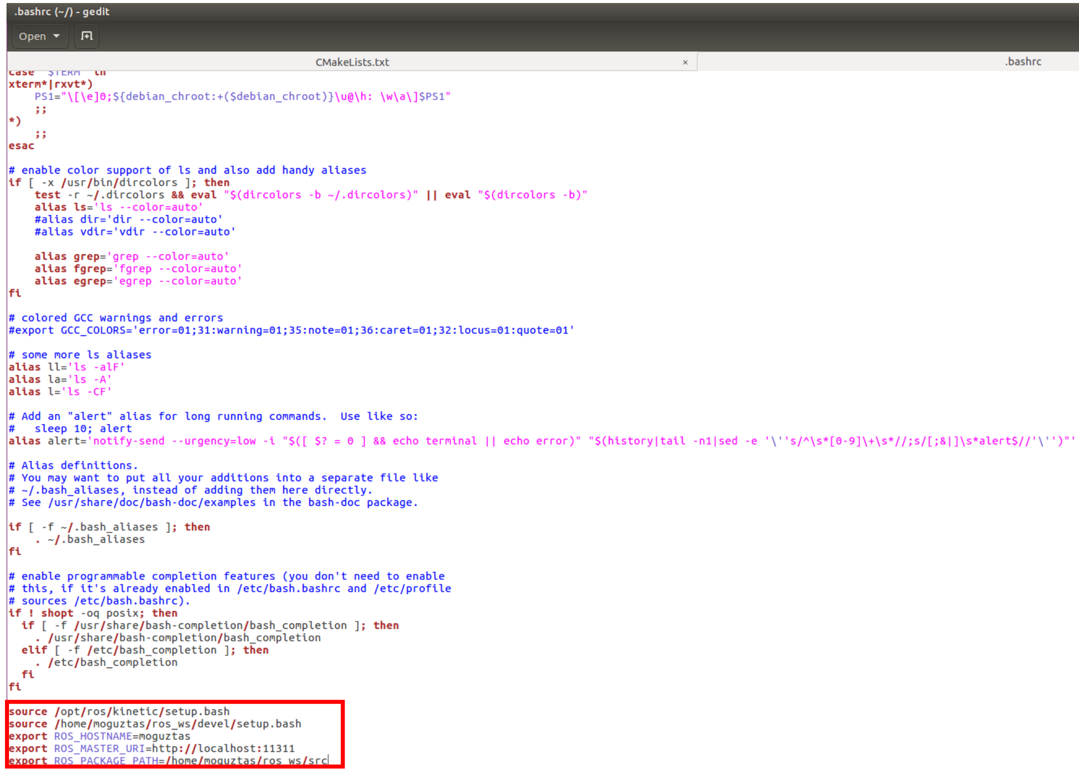
gedit ~/.bashrc

The following codes are added to the screen opened in the Gedit editor.

<source /opt/ros/kinetic/setup.bash>

<source /home/<user_name>/ros_ws/devel/setup.bash>

Not: The terminal must be renewed with the bash command so that changes made in *bashrc* can be detected in the terminal that was opened previously.



```
case  $TERM  in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

source /opt/ros/kinetic/setup.bash
source /home/moguztas/ros_ws/devel/setup.bash
export ROS_HOSTNAME=moguztas
export ROS_MASTER_URI=http://localhost:11311
export ROS_PACKAGE_PATH=/home/moguztas/ros_ws/src
```

Application 1: Preparation of ROS Environment– 3/3

3. To create work environment:

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

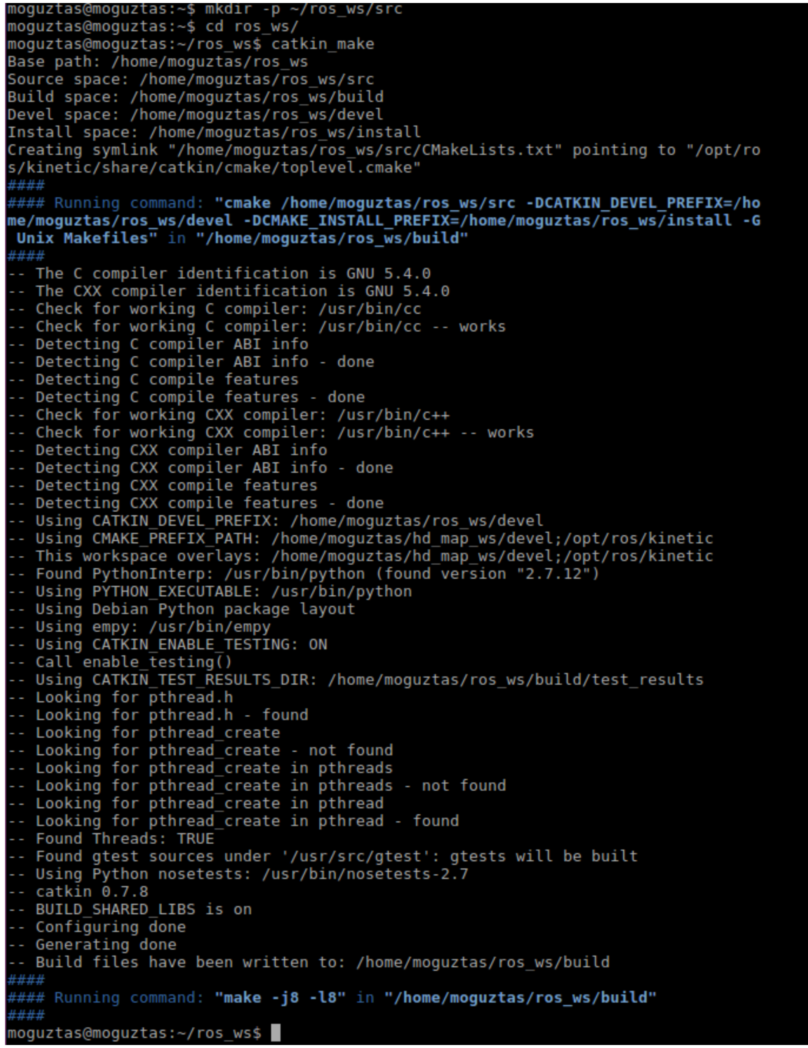
Not: A block of code that can do the same with catkin_make :

```
cd ~/ros_ws
cd src
catkin_init_workspace
cd ..
mkdir build
cd build
cmake ../src -DCMAKE_INSTALL_PREFIX=../install -DCATKIN_DEVEL_PREFIX=../devel
make
```

Not: The following code is written on the terminal screen for the location of the ROS_PACKAGE_PATH configuration variable.

```
echo $ROS_PACKAGE_PATH
```

/home/(USER_NAME)/ros_ws/src:/opt/ros/kinetic/share



```
moguztas@moguztas:~$ mkdir -p ~/ros_ws/src
moguztas@moguztas:~$ cd ros_ws/
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
Creating symlink "/home/moguztas/ros_ws/src/CMakeLists.txt" pointing to "/opt/ro
s/kinetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/moguztas/ros_ws/src -DCATKIN_DEVEL_PREFIX=/ho
me/moguztas/ros_ws/devel -DCMAKE_INSTALL_PREFIX=/home/moguztas/ros_ws/install -G
 Unix Makefiles" in "/home/moguztas/ros_ws/build"
####
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- Found PythonInterp: /usr/bin/python (found version "2.7.12")
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
moguztas@moguztas:~/ros_ws$ 
```

Application 2: Creating a Catkin Package and Getting to Know the ROS Environment– 1/5

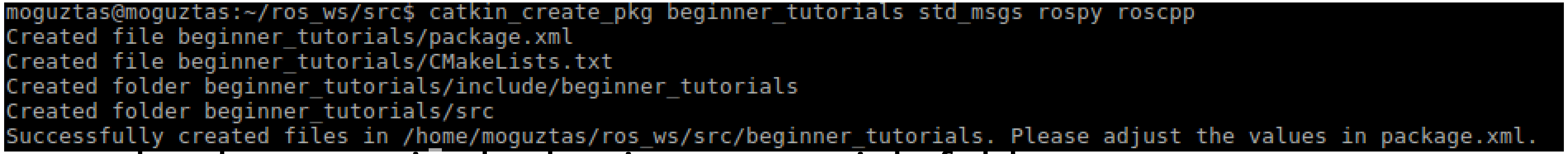
1. First, go to the src directory in the ros workspace..

cd ~/ros_ws/src

2. *With the catkin_create_pkg command, a package named beginner_tutorials is linked to std_msgs, roscpp and rospy*:

catkin_create_pkg beginner_tutorials std_msgs rospy roscpp

Not: This will create a file containing package.xml named beginner_tutorial and a CMakeLists.txt file. The CMakeLists.txt file is partially populated by the catkin_create_pkg command..

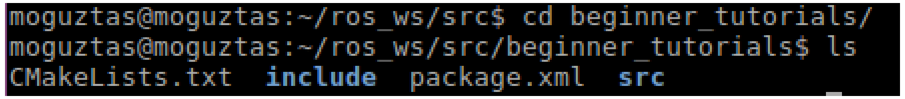


```
moguztas@moguztas:~/ros_ws/src$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
Created file beginner_tutorials/package.xml
Created file beginner_tutorials/CMakeLists.txt
Created folder beginner_tutorials/include/beginner_tutorials
Created folder beginner_tutorials/src
Successfully created files in /home/moguztas/ros_ws/src/beginner_tutorials. Please adjust the values in package.xml.
```

3. To see what happens in the beginner tutorials folder :

cd beginner_tutorials

ls



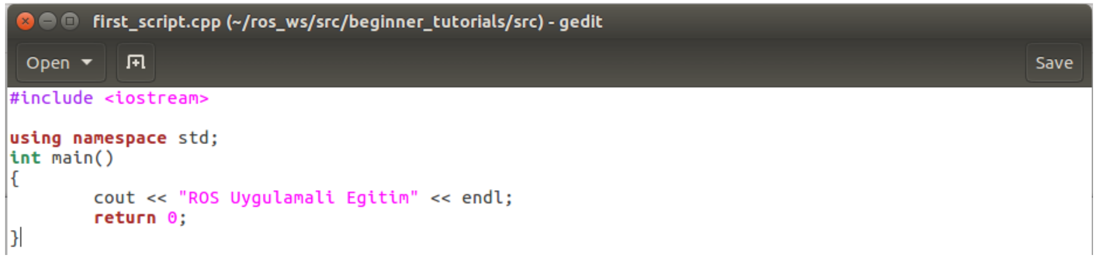
```
moguztas@moguztas:~/ros_ws/src$ cd beginner_tutorials/
moguztas@moguztas:~/ros_ws/src/beginner_tutorials$ ls
CMakeLists.txt  include  package.xml  src
```

Application 2: Creating a Catkin Package and Getting to Know the ROS Environment– 2/5

4. Let's go to beginner_tutorials src file and write a simple code here:

cd src

gedit first_script.cpp

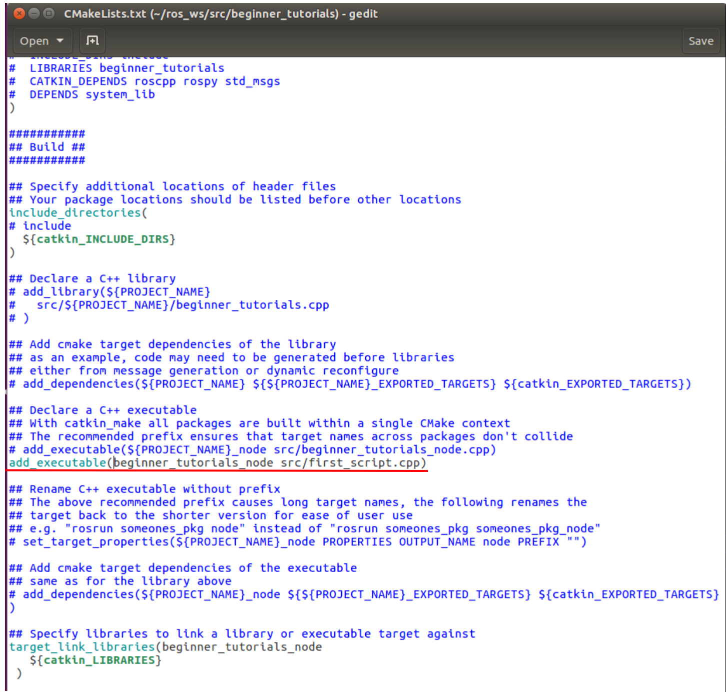

first_script.cpp (~/ros_ws/src/beginner_tutorials/src) - gedit
Open
Save

```cpp
#include <iostream>

using namespace std;
int main()
{
	cout << "ROS Uygulamali Egitim" << endl;
	return 0;
}
```

5. Let's make the code we write in the CMakeLists.txt file executable:

cd ..

gedit CMakeLists.txt

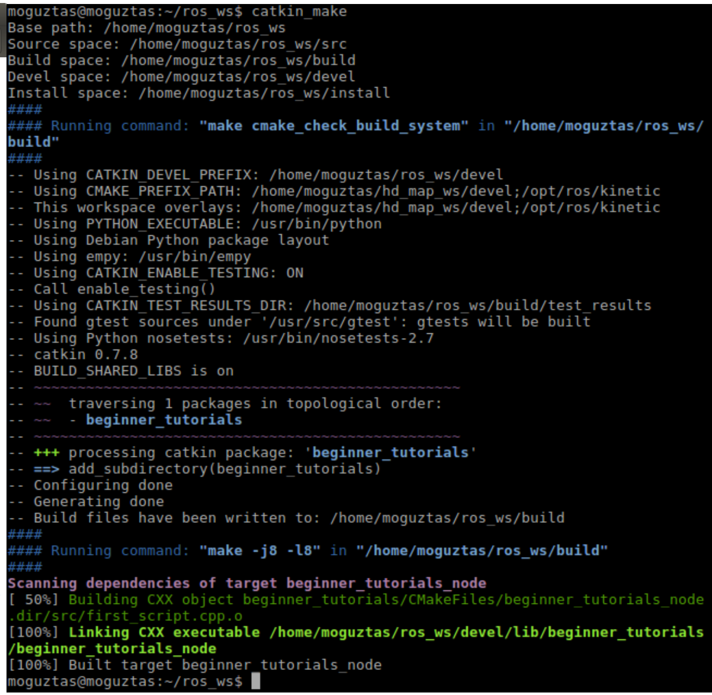


```
#  INCLUDE_DIRS include
#  LIBRARIES beginner_tutorials
#  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/beginner_tutorials.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/beginner_tutorials_node.cpp)
add_executable(beginner_tutorials_node src/first_script.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS}
)

## Specify libraries to link a library or executable target against
target_link_libraries(beginner_tutorials_node
   ${catkin_LIBRARIES}
 )
```

6. Let's compile our workspace.

cd ~/ros_ws/

catkin_make



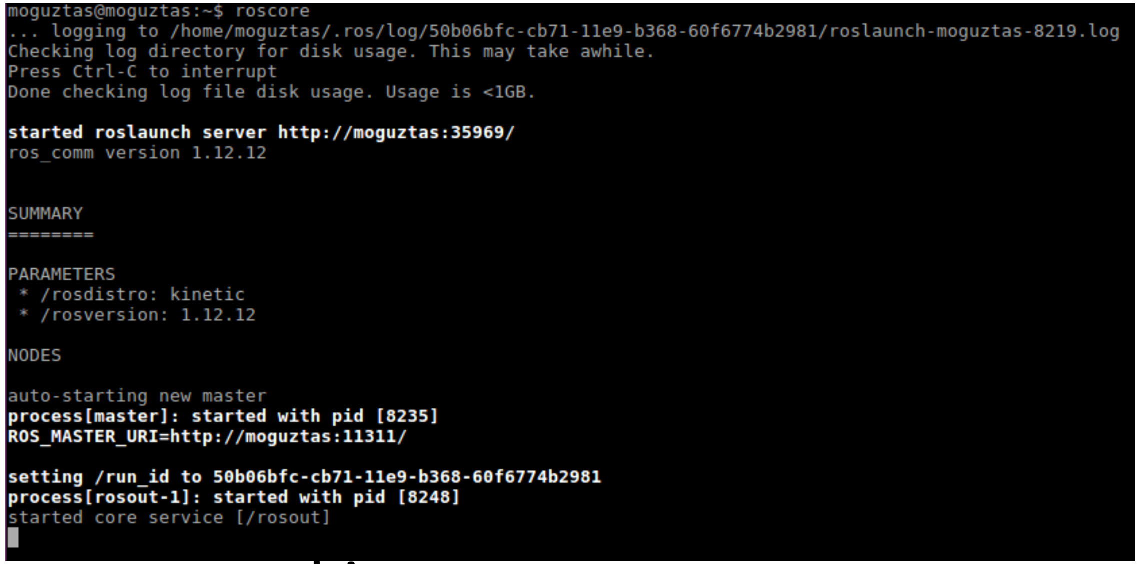
```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/
build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
Scanning dependencies of target beginner_tutorials_node
[ 50%] Building CXX object beginner_tutorials/CMakeFiles/beginner_tutorials_node
.dir/src/first_script.cpp.o
[100%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/beginner_tutorials_node
[100%] Built target beginner_tutorials_node
moguztas@moguztas:~/ros_ws$
```

Not: When the compilation is completed, build, devel and src subfolders will be installed in the src folder, the package will be ready for use.

Application 2: Creating a Catkin Package and Getting to Know the ROS Environment– 3/5

7. Two terminals are opened to run the written code. The following code is executed in the first terminal.

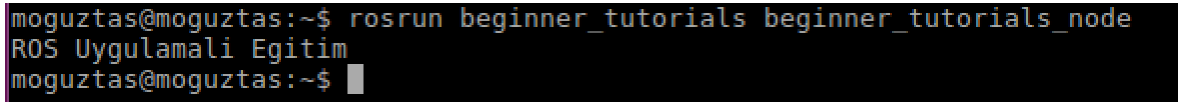
roscore



```
moguztas@moguztas:~$ roscore
... logging to /home/moguztas/.ros/log/50b06bfc-cb71-11e9-b368-60f6774b2981/roslaunch-moguztas-8219.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://moguztas:35969/
ros_comm version 1.12.12


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES

auto-starting new master
process[master]: started with pid [8235]
ROS_MASTER_URI=http://moguztas:11311/

setting /run_id to 50b06bfc-cb71-11e9-b368-60f6774b2981
process[rosout-1]: started with pid [8248]
started core service [/rosout]
```

8. In the other terminal, the ros package created is run.

rosrun beginner_tutorials beginner_tutorials_node

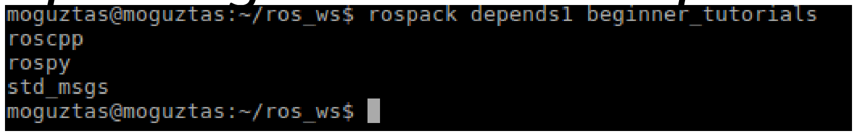


```
moguztas@moguztas:~$ rosrun beginner_tutorials beginner_tutorials_node
ROS Uygulamali Egitim
moguztas@moguztas:~$
```

Application 2: Creating a Catkin Package and Getting to Know the ROS Environment– 4/5

9.1. *To view the dependencies on the package with the rospack tool*:

rospack depends1 beginner_tutorials

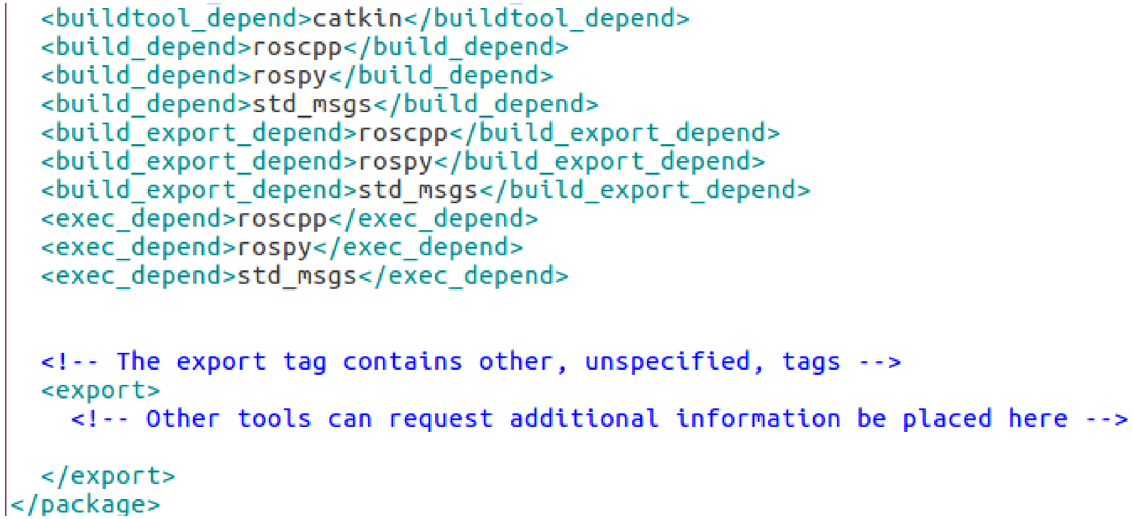


```
moguztas@moguztas:~/ros_ws$ rospack depends1 beginner_tutorials
roscpp
rospy
std_msgs
moguztas@moguztas:~/ros_ws$ 
```

9.2. These dependencies are also listed in the package.xml file.

roscd beginner_tutorials

gedit package.xml

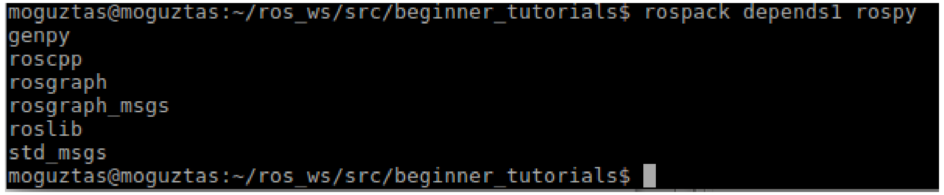


```
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

Application 2: Creating a Catkin Package and Getting to Know the ROS Environment– 5/5

10. Indirect dependencies can be viewed with the rospack tool. For example, to see rospy dependencies:

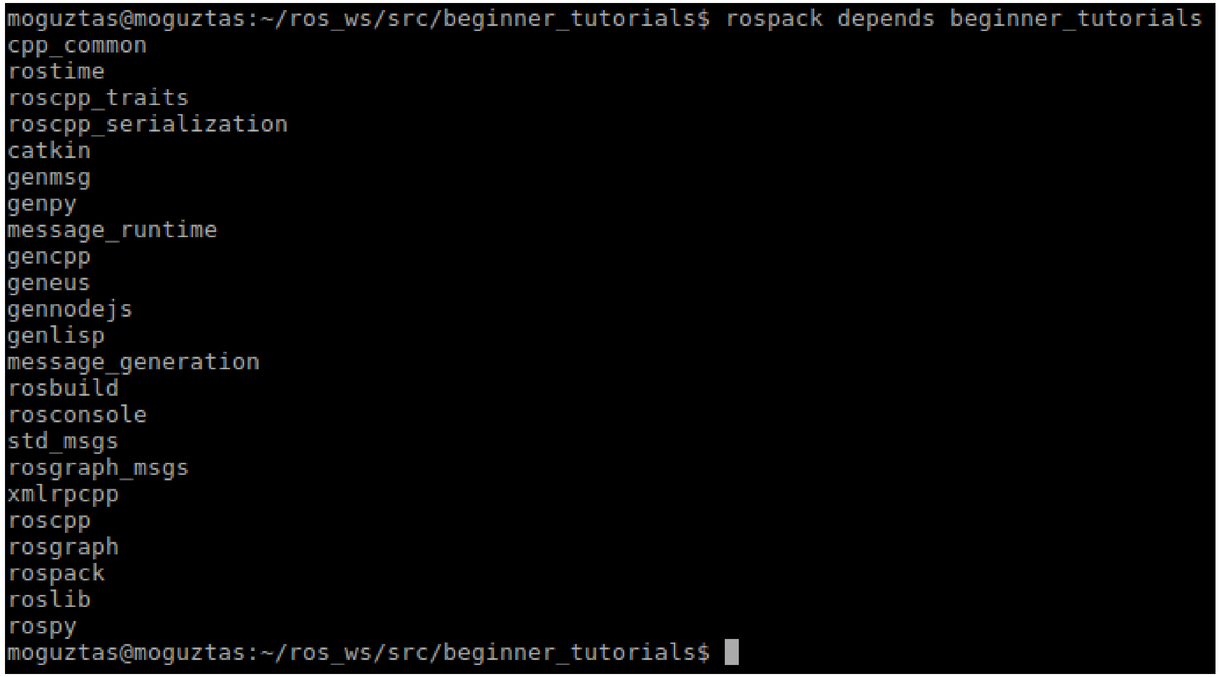
rospack depends1 rospy



```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials$ rospack depends1 rospy
genpy
roscpp
rosgraph
rosgraph_msgs
roslib
std_msgs
moguztas@moguztas:~/ros_ws/src/beginner_tutorials$ 
```

11. To see all the dependencies in the package:

rospack depends beginner_tutorials



```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials$ rospack depends beginner_tutorials
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
rospack
roslib
rospy
moguztas@moguztas:~/ros_ws/src/beginner_tutorials$ 
```

Application 3: TurtleSim – 1/4

1. For TurtleSim application:

sudo apt-get install ros-kinetic-ros-tutorials

2. To start the ROS Master:

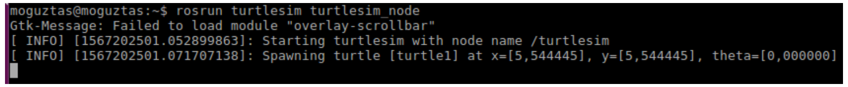
roscore

3. To run TurtleSim:

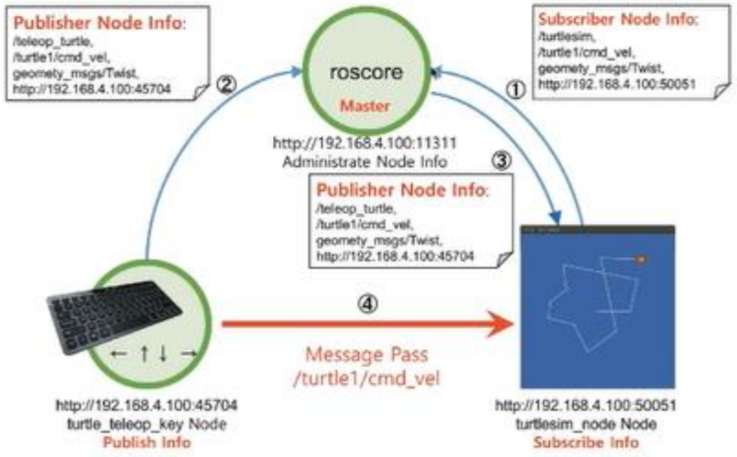
rosrun turtlesim turtlesim_node

4. For keyboard control:

rosrun turtlesim turtle_teleop_key



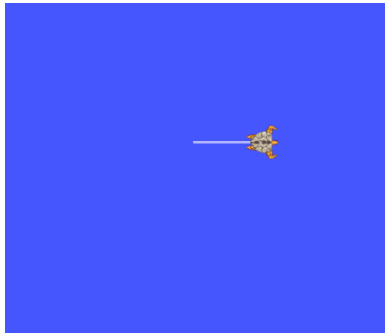
```
moguztas@moguztas:~$ rosrun turtlesim turtlesim_node
Gtk-Message: Failed to load module "overlay-scrollbar"
[ INFO] [1567202501.052899863]: Starting turtlesim with node name /turtlesim
[ INFO] [1567202501.071707138]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=[0,000000]
```



Publisher Node Info:
/teleop_turtle,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:45704

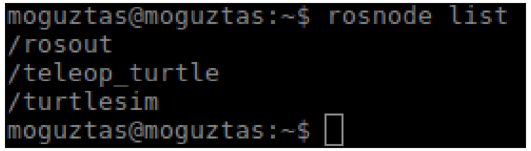
Subscriber Node Info:
/turtlesim,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:50051

Publisher Node Info:
/teleop_turtle,
/turtle1/cmd_vel,
geomety_msgs/Twist,
http://192.168.4.100:45704



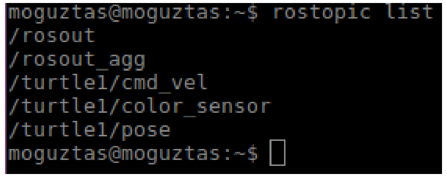
Application 3: TurtleSim – 2/4

5. *rosnode list*



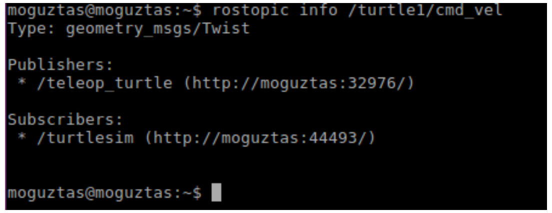
```
moguztas@moguztas:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
moguztas@moguztas:~$ 
```

6. *rostopic list*



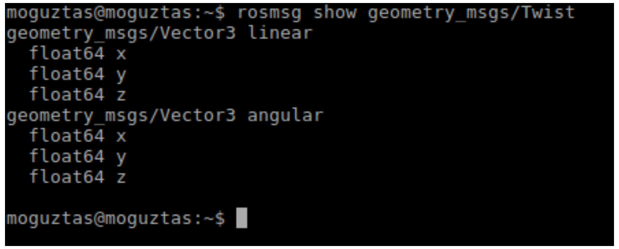
```
moguztas@moguztas:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
moguztas@moguztas:~$ 
```

7. *rostopic info /turtle1/cmd_vel*



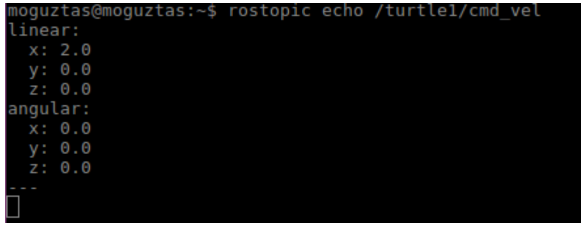
```
moguztas@moguztas:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://moguztas:32976/)

Subscribers:
 * /turtlesim (http://moguztas:44493/)


moguztas@moguztas:~$ 
```

8. *rosmsg show geometry_msgs/Twist*



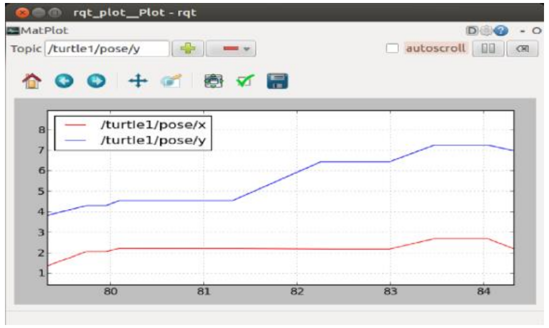
```
moguztas@moguztas:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

moguztas@moguztas:~$ 
```

9. *rostopic echo /turtle1/cmd_vel*

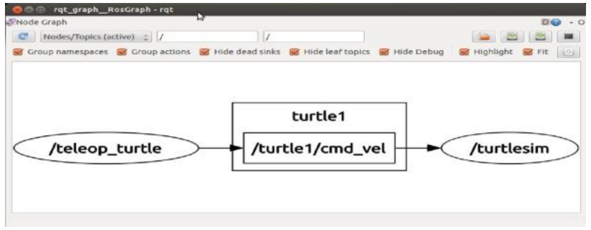


```
moguztas@moguztas:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---

```

10. *rqt_plot*

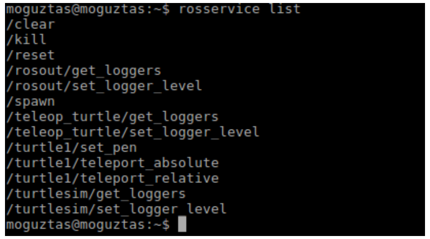


11. *rqt_graph*



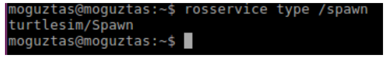
Application 3: TurtleSim – 3/4

12. *rosservice list*



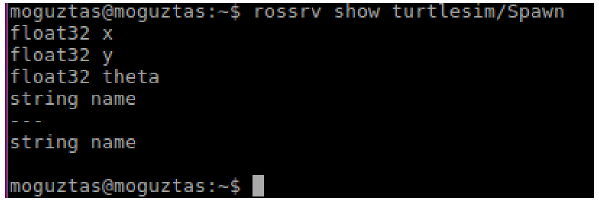
```
moguztas@moguztas:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
moguztas@moguztas:~$
```

13. *rosservice type/spawn*



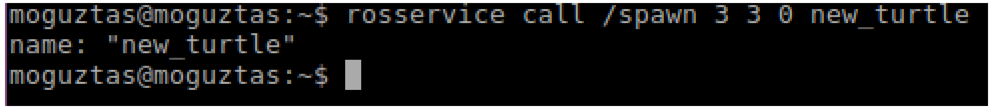
```
moguztas@moguztas:~$ rosservice type /spawn
turtlesim/Spawn
moguztas@moguztas:~$
```

14. *rossrv show turtlesim/Spawn*



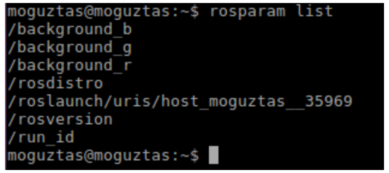
```
moguztas@moguztas:~$ rossrv show turtlesim/Spawn
float32 x
float32 y
float32 theta
string name
---
string name

moguztas@moguztas:~$
```

15. *rosservice call /spawn 3 3 0 new_turtle*



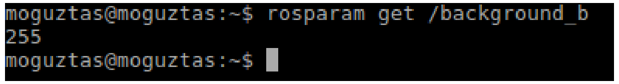
```
moguztas@moguztas:~$ rosservice call /spawn 3 3 0 new_turtle
name: "new_turtle"
moguztas@moguztas:~$
```

16. *rosparam list*



```
moguztas@moguztas:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_moguztas__35969
/rosversion
/run_id
moguztas@moguztas:~$
```

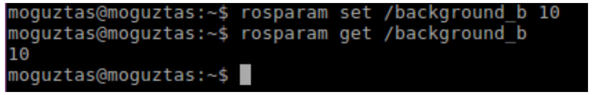
17. *rosparam get /background_b*



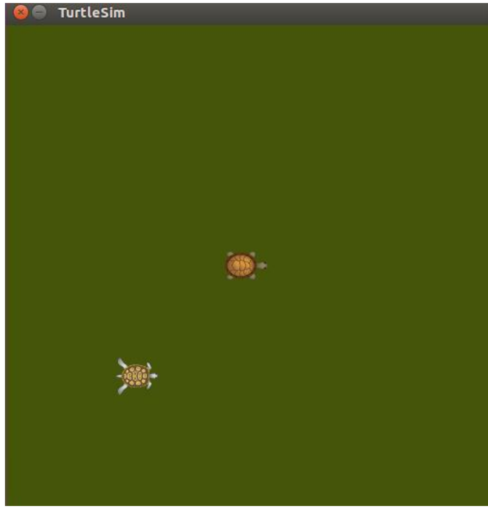
```
moguztas@moguztas:~$ rosparam get /background_b
255
moguztas@moguztas:~$
```

18. *rosparam set /background_b 10*

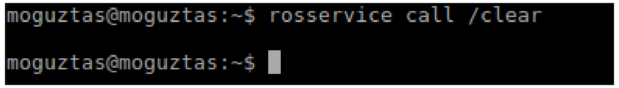
rosparam get /background_b



```
moguztas@moguztas:~$ rosparam set /background_b 10
moguztas@moguztas:~$ rosparam get /background_b
10
moguztas@moguztas:~$
```



19. *rosservice call /clear*



```
moguztas@moguztas:~$ rosservice call /clear

moguztas@moguztas:~$
```

Application 3: TurtleSim – 4/4

roslaunch, starts the specified run file. Its use is as follows:

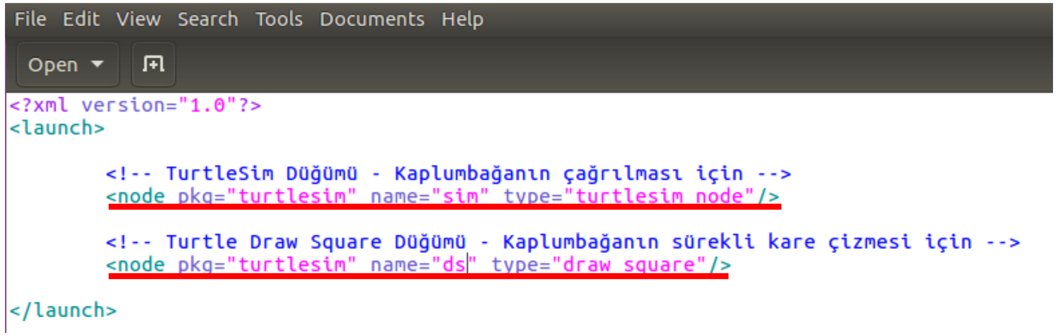
roslaunch [package] [filename.launch]

20. First, let's go to the package we created with the name beginner_tutorials and create a launch folder.

roscd beginner_tutorials

mkdir launch

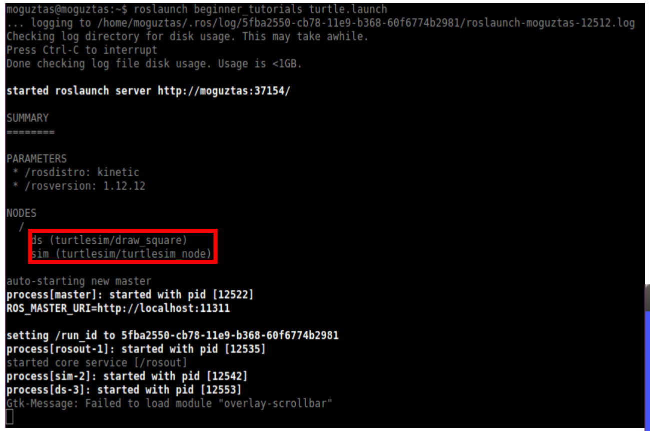
21. Let's create a startup file called *turtle.launch*.



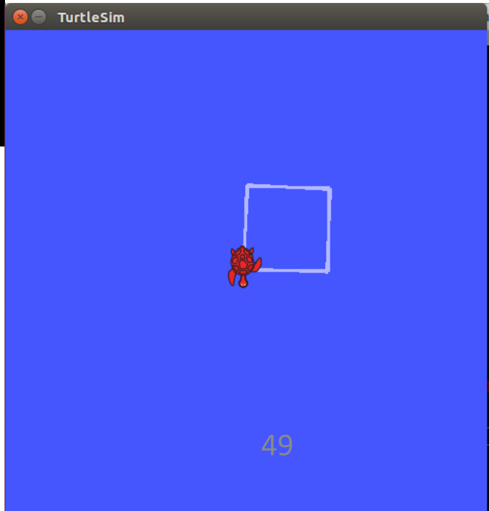
```
<?xml version="1.0"?>
<launch>

        <!-- TurtleSim Düğümü - Kaplumbağanın çağrılması için -->
        <node pkg="turtlesim" name="sim" type="turtlesim_node"/>

        <!-- Turtle Draw Square Düğümü - Kaplumbağanın sürekli kare çizmesi için -->
        <node pkg="turtlesim" name="ds" type="draw_square"/>

</launch>
```

22. Let's write the following code to the terminal to call the launch file.

roslaunch beginner_tutorials turtle.launch



```
moguztas@moguztas:~$ roslaunch beginner_tutorials turtle.launch
... logging to /home/moguztas/.ros/log/5fba2550-cb78-11e9-b368-60f6774b2981/roslaunch-moguztas-12512.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://moguztas:37154/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES
  /
    ds (turtlesim/draw_square)
    sim (turtlesim/turtlesim_node)

auto-starting new master
process[master]: started with pid [12522]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 5fba2550-cb78-11e9-b368-60f6774b2981
process[rosout-1]: started with pid [12535]
started core service [/rosout]
process[sim-2]: started with pid [12542]
process[ds-3]: started with pid [12553]
Gtk-Message: Failed to load module "overlay-scrollbar"
```



Application 4: Creating Messages and Services– 1/4
Creating Messages– 1/2

1. Let's create the msg folder in the beginner_tutorials folder in the workspace..

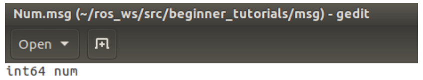
roscd beginner_tutorials

mkdir msg

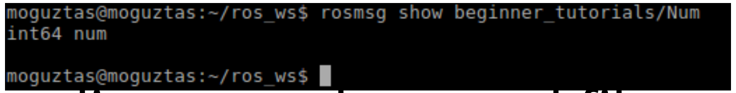
2. Let's create our message file and show the file we created in the terminal..

echo "int64 num" > msg/Num.msg

rosmsg show beginner_tutorials/Num



```
int64 num
```



```
moguztas@moguztas:~/ros_ws$ rosmsg show beginner_tutorials/Num
int64 num

moguztas@moguztas:~/ros_ws$ 
```

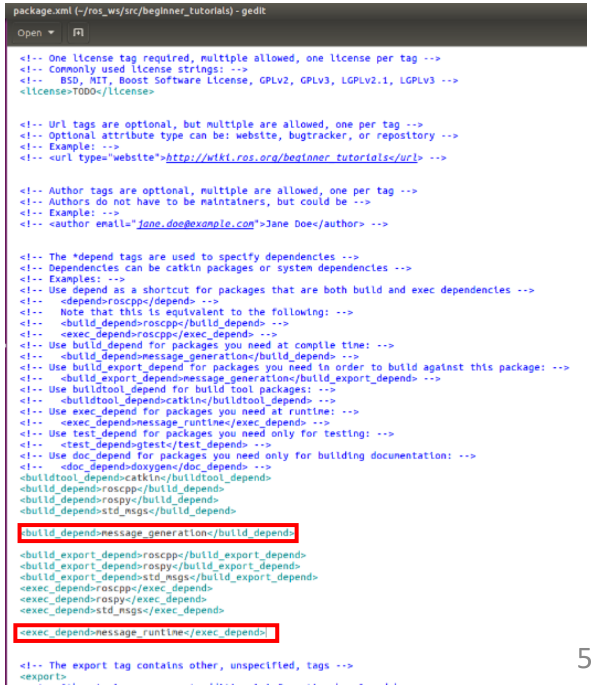
3. Let's edit our package.xml file.

roscd beginner_tutorials

gedit package.xml

<build_depend>message_generation</build_depend>

<exec_depend>message_runtime</exec_depend>



```
<!-- One license tag required, multiple allowed, one license per tag -->
<!-- Commonly used license strings: -->
<!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
<license>TODO</license>


<!-- Url tags are optional, but multiple are allowed, one per tag -->
<!-- Optional attribute type can be: website, bugtracker, or repository -->
<!-- Example: -->
<!-- <url type="website">http://wiki.ros.org/beginner_tutorials</url> -->


<!-- Author tags are optional, multiple are allowed, one per tag -->
<!-- Authors do not have to be maintainers, but could be -->
<!-- Example: -->
<!-- <author email="jane.doe@example.com">Jane Doe</author> -->


<!-- The *depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
<!--   <depend>roscpp</depend> -->
<!--   Note that this is equivalent to the following: -->
<!--   <build_depend>roscpp</build_depend> -->
<!--   <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile time: -->
<!--   <build_depend>message_generation</build_depend> -->
<!-- Use build_export_depend for packages you need in order to build against this package: -->
<!--   <build_export_depend>message_generation</build_export_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--   <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use exec_depend for packages you need at runtime: -->
<!--   <exec_depend>message_runtime</exec_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation: -->
<!--   <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>

<build_depend>message_generation</build_depend>

<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>

<exec_depend>message_runtime</exec_depend>


<!-- The export tag contains other, unspecified, tags -->
<export>
```

Application 4: Creating Messages and Services– 2/4
Creating Messages– 2/2

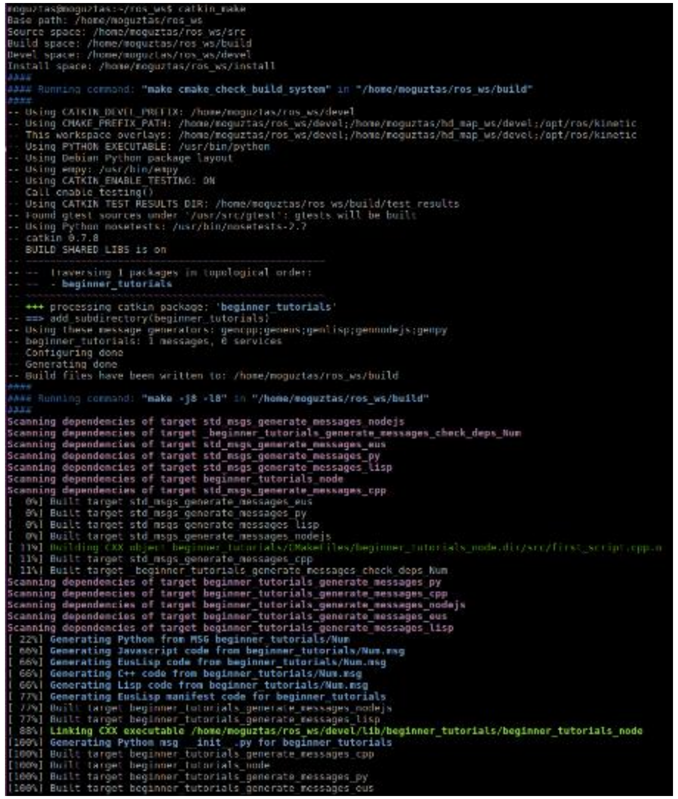
4. Let's edit our CMakeLists.txt file as follows.

gedit CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
...
message_generation
)

catkin_package(
...
CATKIN_DEPENDS message_runtime ...
...
)

add_message_files(
FILES
Num.msg
)

generate_messages(
DEPENDENCIES
std_msgs
)
```

5. Lets compile workspace.

cd ~/ros_ws
catkin_make



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 0 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
Scanning dependencies of target std_msgs_generate_messages_nodejs
Scanning dependencies of target _beginner_tutorials_generate_messages_check_deps_Num
Scanning dependencies of target std_msgs_generate_messages_eus
Scanning dependencies of target std_msgs_generate_messages_py
Scanning dependencies of target std_msgs_generate_messages_lisp
Scanning dependencies of target beginner_tutorials_node
Scanning dependencies of target std_msgs_generate_messages_cpp
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_nodejs
[ 11%] Building CXX object beginner_tutorials/CMakeFiles/beginner_tutorials_node.dir/src/first_script.cpp.o
[ 11%] Built target std_msgs_generate_messages_cpp
[ 11%] Built target _beginner_tutorials_generate_messages_check_deps_Num
Scanning dependencies of target beginner_tutorials_generate_messages_py
Scanning dependencies of target beginner_tutorials_generate_messages_cpp
Scanning dependencies of target beginner_tutorials_generate_messages_nodejs
Scanning dependencies of target beginner_tutorials_generate_messages_eus
Scanning dependencies of target beginner_tutorials_generate_messages_lisp
[ 22%] Generating Python from MSG beginner_tutorials/Num
[ 66%] Generating Javascript code from beginner_tutorials/Num.msg
[ 66%] Generating EusLisp code from beginner_tutorials/Num.msg
[ 66%] Generating C++ code from beginner_tutorials/Num.msg
[ 66%] Generating Lisp code from beginner_tutorials/Num.msg
[ 77%] Generating EusLisp manifest code for beginner_tutorials
[ 77%] Built target beginner_tutorials_generate_messages_nodejs
[ 77%] Built target beginner_tutorials_generate_messages_lisp
[ 88%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials/beginner_tutorials_node
[100%] Generating Python msg __init__.py for beginner_tutorials
[100%] Built target beginner_tutorials_generate_messages_cpp
[100%] Built target beginner_tutorials_node
[100%] Built target beginner_tutorials_generate_messages_py
[100%] Built target beginner_tutorials_generate_messages_eus
```

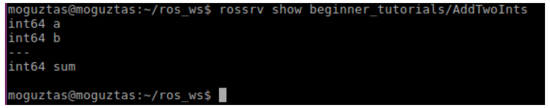
Application 4: Creating Messages and Services – 3/4 Creating Services – 1/2

1. Let's create the **srv** folder in the beginner_tutorials folder in the workspace..

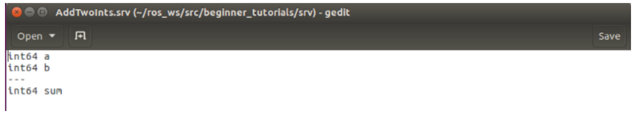
```
roscd beginner_tutorials
mkdir srv
```

2. Let's create our srv file and show the file we created in the terminal.

```
roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
rossrv show beginner_tutorials/AddTwoInts
```



```
moguztas@moguztas:~/ros_ws$ rossrv show beginner_tutorials/AddTwoInts
int64 a
int64 b
---
int64 sum

moguztas@moguztas:~/ros_ws$ 
```



AddTwoInts.srv (~/ros_ws/src/beginner_tutorials/srv) - gedit
Open
Save

```
int64 a
int64 b
---
int64 sum
```

Not: srv files are like **msg** files, except they contain two partitions.

3. Let's edit our package.xml file.

```
roscd beginner_tutorials
gedit package.xml
        <build_depend>message_generation</build_depend>
        <exec_depend>message_runtime</exec_depend>
```

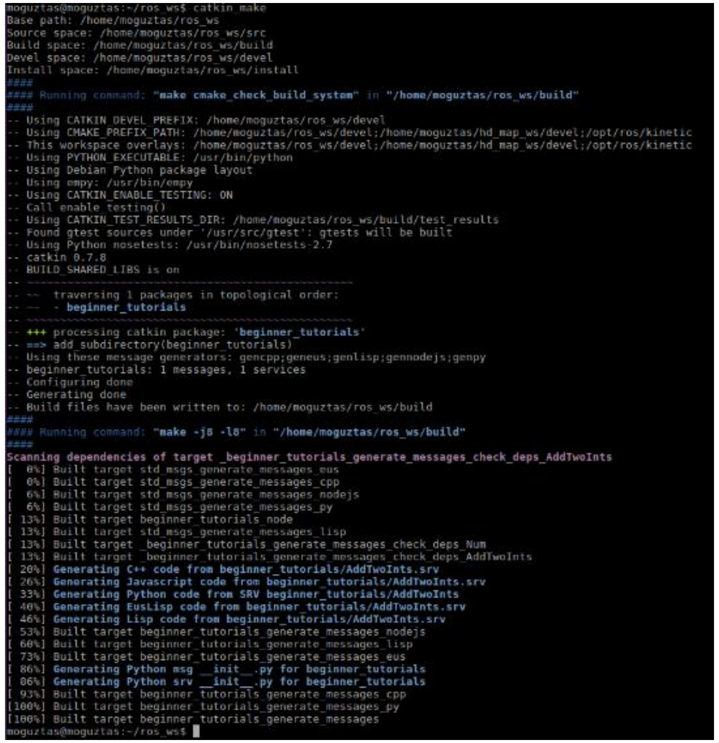
Application 4: Creating Messages and Services – 4/4 Creating Services – 2/2

4. Let's edit our CMakeLists.txt file as follows.

```
gedit CMakeLists.txt
        find_package(catkin REQUIRED COMPONENTS
        ...
        message_generation
        )

        catkin_package(
        ...
        CATKIN_DEPENDS message_runtime ...
        ...
        )

        add_service_files(
        FILES
        AddTwoInts.srv
        )

        generate_messages(
        DEPENDENCIES
        std_msgs
        )
```

5. Compile workspace.

```
cd ~/ros_ws
catkin_make
```



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 1 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
Scanning dependencies of target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_cpp
[  6%] Built target std_msgs_generate_messages_nodejs
[  6%] Built target std_msgs_generate_messages_py
[ 13%] Built target beginner_tutorials_node
[ 13%] Built target std_msgs_generate_messages_lisp
[ 13%] Built target _beginner_tutorials_generate_messages_check_deps_Num
[ 13%] Built target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[ 20%] Generating C++ code from beginner_tutorials/AddTwoInts.srv
[ 26%] Generating Javascript code from beginner_tutorials/AddTwoInts.srv
[ 33%] Generating Python code from SRV beginner_tutorials/AddTwoInts
[ 40%] Generating EusLisp code from beginner_tutorials/AddTwoInts.srv
[ 46%] Generating Lisp code from beginner_tutorials/AddTwoInts.srv
[ 53%] Built target beginner_tutorials_generate_messages_nodejs
[ 60%] Built target beginner_tutorials_generate_messages_lisp
[ 73%] Built target beginner_tutorials_generate_messages_eus
[ 86%] Generating Python msg __init__.py for beginner_tutorials
[ 86%] Generating Python srv __init__.py for beginner_tutorials
[ 93%] Built target beginner_tutorials_generate_messages_cpp
[100%] Built target beginner_tutorials_generate_messages_py
[100%] Built target beginner_tutorials_generate_messages
moguztas@moguztas:~/ros_ws$
```

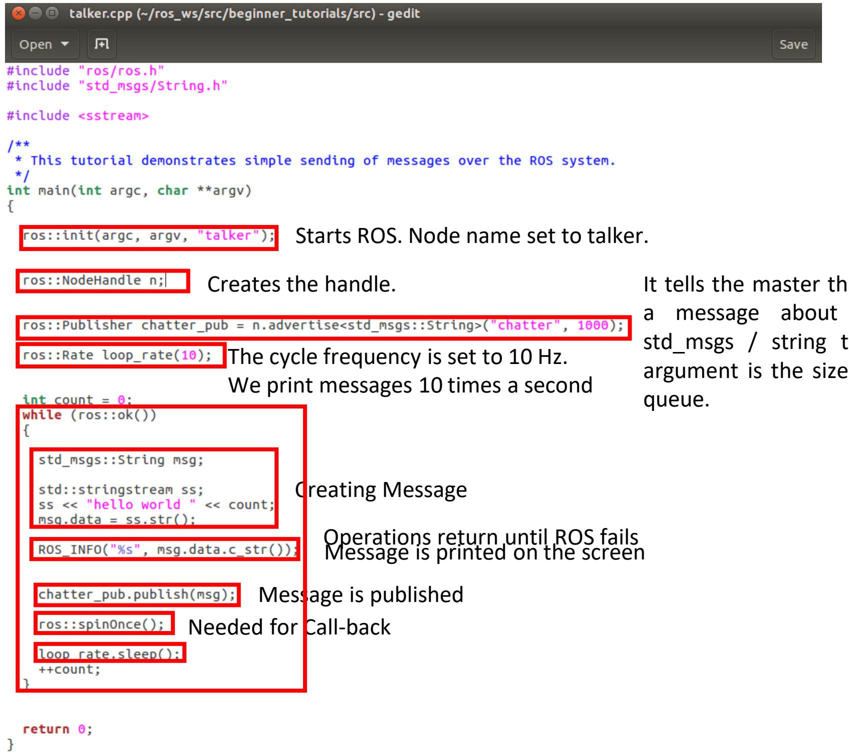
Application 5: Publisher-Subscirber Application – 1/5
C++ Application – 1/2

1. Let's go to the src folder under beginner_tutorials.

roscd beginner_tutorials/src

2. Let's create our Publisher file.

gedit talker.cpp



```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

/**
 * This tutorial demonstrates simple sending of messages over the ROS system.
 */
int main(int argc, char **argv)
{

  ros::init(argc, argv, "talker");

  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);

  int count = 0;
  while (ros::ok())
  {

    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());


    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

Starts ROS. Node name set to talker.

Creates the handle.

It tells the master that we will publish a message about the **chatter** in std_msgs / string type. The second argument is the size of the broadcast queue.

The cycle frequency is set to 10 Hz.
We print messages 10 times a second

Creating Message

Operations return until ROS fails

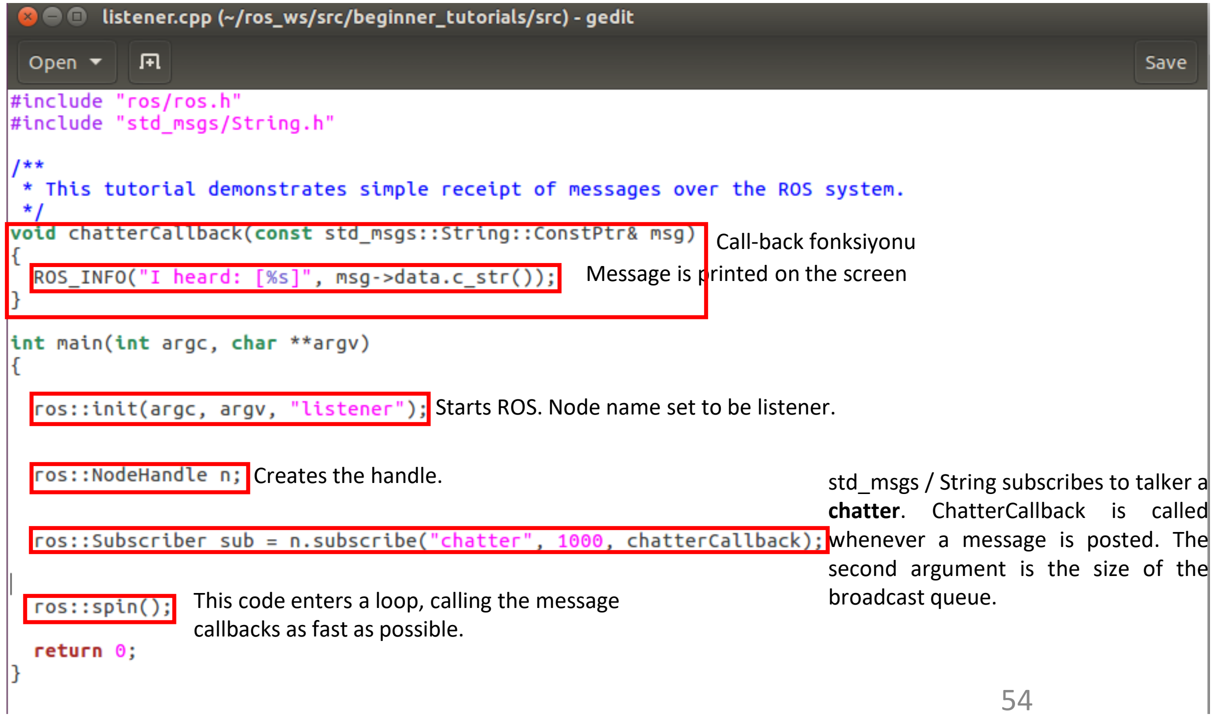
Message is printed on the screen

Message is published

Needed for Call-back

3. Let's create our subscriber file.

gedit listener.cpp



```
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{

  ros::init(argc, argv, "listener");


  ros::NodeHandle n;


  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);


  ros::spin();

  return 0;
}
```

Call-back fonksiyonu

Message is printed on the screen

Starts ROS. Node name set to be listener.

Creates the handle.

std_msgs / String subscribes to talker a **chatter**. ChatterCallback is called whenever a message is posted. The second argument is the size of the broadcast queue.

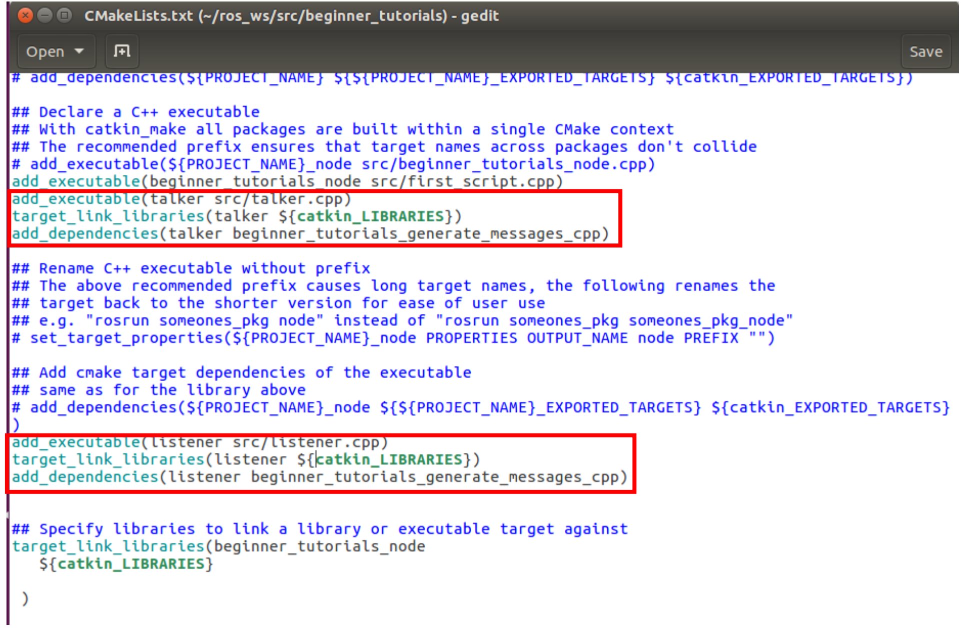
This code enters a loop, calling the message callbacks as fast as possible.

Application 5: Publisher-Subscirber Application – 2/5
C++ Application – 2/2

4. Let's edit our CMakeLists.txt file.

roscd beginner_tutorials

gedit CMakeLists.txt

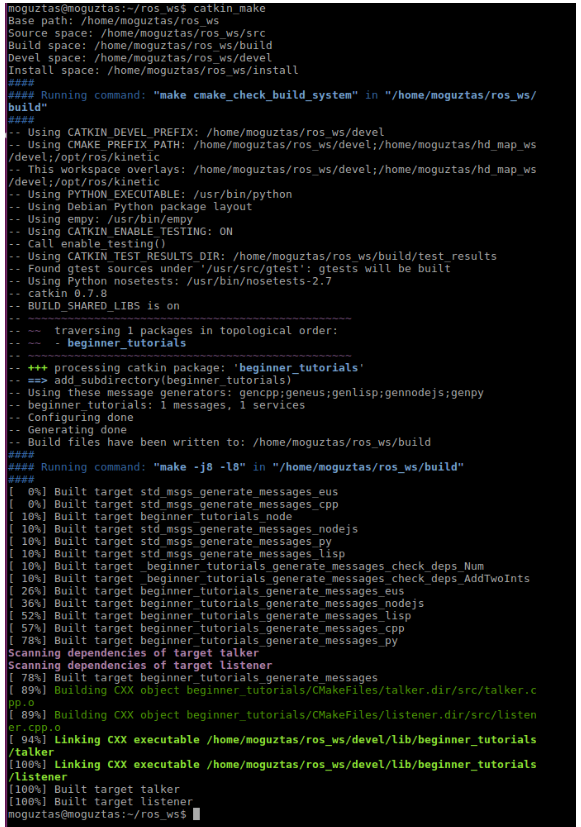


```
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/beginner_tutorials_node.cpp)
add_executable(beginner_tutorials_node src/first_script.cpp)
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS}
)
add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)


## Specify libraries to link a library or executable target against
target_link_libraries(beginner_tutorials_node
   ${catkin_LIBRARIES}

 )
```

5. Compile workspace.

cd ~/ros_ws

catkin_make



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/
build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 1 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_cpp
[ 10%] Built target beginner_tutorials_node
[ 10%] Built target std_msgs_generate_messages_nodejs
[ 10%] Built target std_msgs_generate_messages_py
[ 10%] Built target std_msgs_generate_messages_lisp
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_Num
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[ 26%] Built target beginner_tutorials_generate_messages_eus
[ 36%] Built target beginner_tutorials_generate_messages_nodejs
[ 52%] Built target beginner_tutorials_generate_messages_lisp
[ 57%] Built target beginner_tutorials_generate_messages_cpp
[ 78%] Built target beginner_tutorials_generate_messages_py
Scanning dependencies of target talker
Scanning dependencies of target listener
[ 78%] Built target beginner_tutorials_generate_messages
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/talker.dir/src/talker.c
pp.o
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/listener.dir/src/listen
er.cpp.o
[ 94%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/talker
[100%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/listener
[100%] Built target talker
[100%] Built target listener
moguztas@moguztas:~/ros_ws$ 
```

Application 5: Publisher-Subscirber Application– 3/5 Python Application – 1/2

1. Let's go to beginner_tutorials folder and create scripts folder.

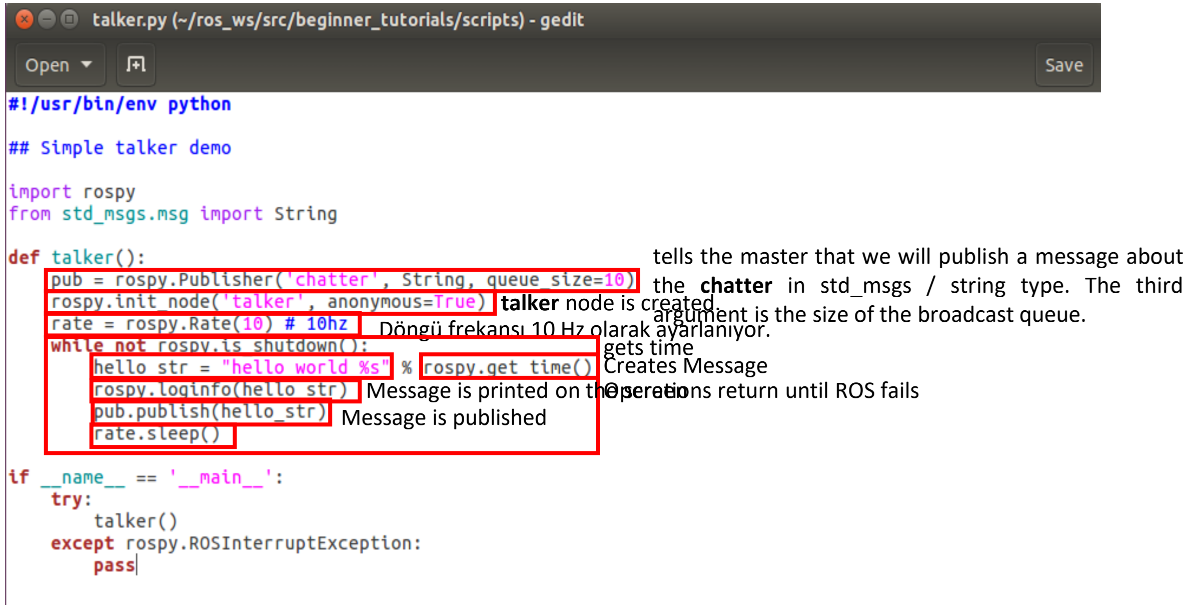
roscd beginner_tutorials/src

mkdir scripts

cd scripts

2. Let's create Publisher file.

gedit talker.py



```python
#!/usr/bin/env python

## Simple talker demo

import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

tells the master that we will publish a message about the **chatter** in std_msgs / string type. The third argument is the size of the broadcast queue.

talker node is created.

Döngü frekansı 10 Hz olarak ayarlanıyor.

gets time

Creates Message

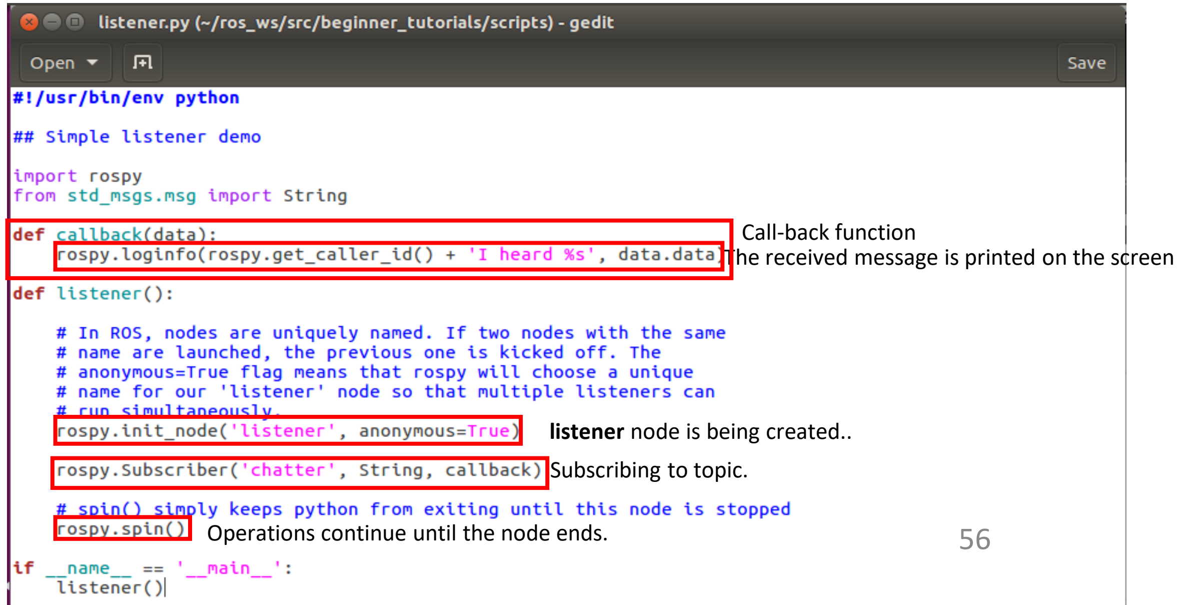
Message is printed on the screen

Operations return until ROS fails

Message is published

3. Let's create subscriber file.

gedit listener.py



listener.py (~/ros_ws/src/beginner_tutorials/scripts) - gedit
Open
Save

```python
#!/usr/bin/env python

## Simple listener demo

import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber('chatter', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

Call-back function

The received message is printed on the screen

listener node is being created..

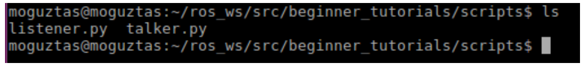
Subscribing to topic.

Operations continue until the node ends.

Application 5: Publisher-Subscirber Application– 4/5
Python Application – 2/2

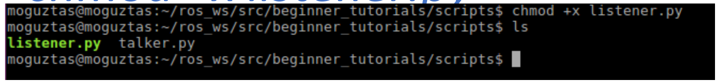
4. Let's make executable files.

ls



```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ ls
listener.py  talker.py
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ █
```

chmod +x listener.py

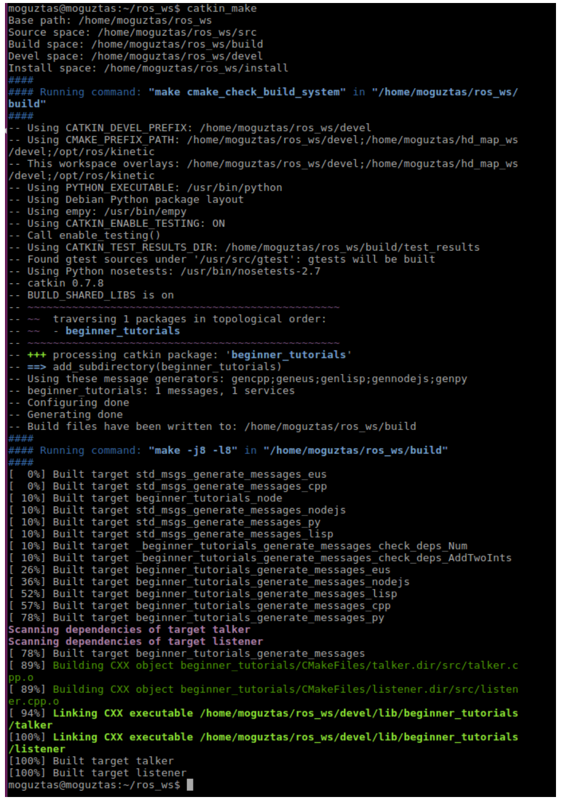


```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ chmod +x listener.py
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ ls
listener.py  talker.py
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ █
```

5. Compile workspace.

cd ~/ros_ws

catkin_make



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/
build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 1 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_cpp
[ 10%] Built target beginner_tutorials_node
[ 10%] Built target std_msgs_generate_messages_nodejs
[ 10%] Built target std_msgs_generate_messages_py
[ 10%] Built target std_msgs_generate_messages_lisp
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_Num
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[ 26%] Built target beginner_tutorials_generate_messages_eus
[ 36%] Built target beginner_tutorials_generate_messages_nodejs
[ 52%] Built target beginner_tutorials_generate_messages_lisp
[ 57%] Built target beginner_tutorials_generate_messages_cpp
[ 78%] Built target beginner_tutorials_generate_messages_py
Scanning dependencies of target talker
Scanning dependencies of target listener
[ 78%] Built target beginner_tutorials_generate_messages
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/talker.dir/src/talker.c
pp.o
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/listener.dir/src/listen
er.cpp.o
[ 94%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/talker
[100%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/listener
[100%] Built target talker
[100%] Built target listener
moguztas@moguztas:~/ros_ws$ █
```

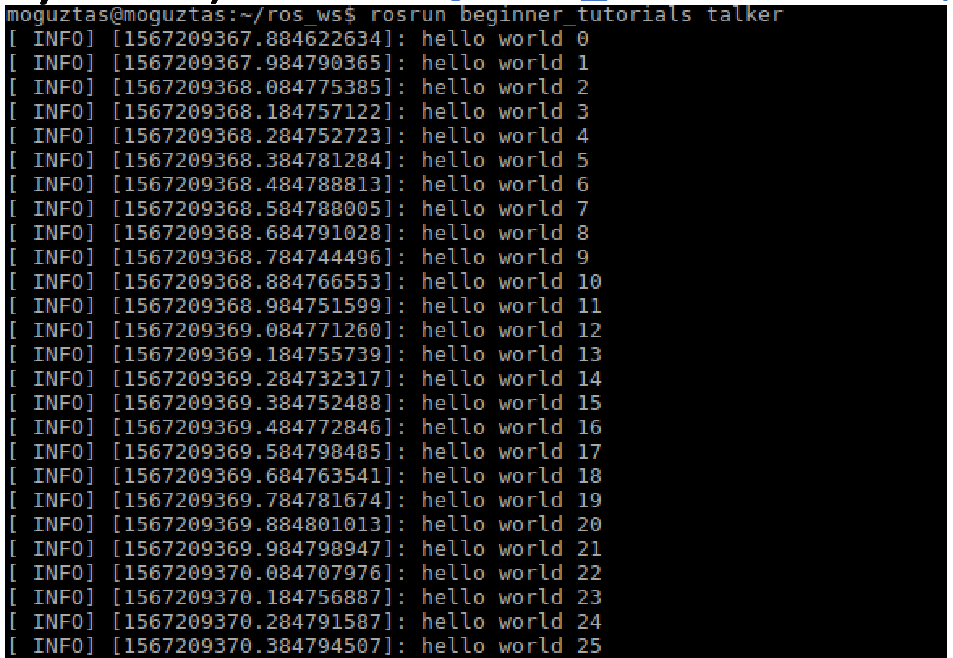
Application 5: Publisher-Subscirber Application– 5/5

6. Run the application

6.1. Terminal 1: *roscore*

6.2. Terminal 2:

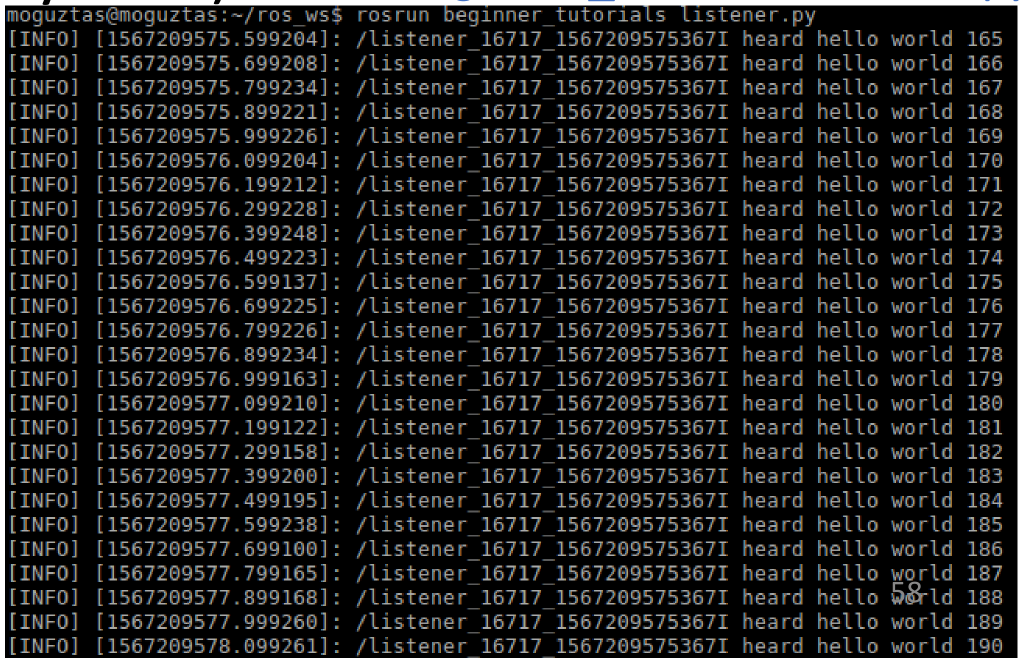
- (C++) *rosrun beginner_tutorials talker*
- (Python) *rosrun beginner_tutorials talker.py*



```
moguztas@moguztas:~/ros_ws$ rosrun beginner_tutorials talker
[ INFO] [1567209367.884622634]: hello world 0
[ INFO] [1567209367.984790365]: hello world 1
[ INFO] [1567209368.084775385]: hello world 2
[ INFO] [1567209368.184757122]: hello world 3
[ INFO] [1567209368.284752723]: hello world 4
[ INFO] [1567209368.384781284]: hello world 5
[ INFO] [1567209368.484788813]: hello world 6
[ INFO] [1567209368.584788005]: hello world 7
[ INFO] [1567209368.684791028]: hello world 8
[ INFO] [1567209368.784744496]: hello world 9
[ INFO] [1567209368.884766553]: hello world 10
[ INFO] [1567209368.984751599]: hello world 11
[ INFO] [1567209369.084771260]: hello world 12
[ INFO] [1567209369.184755739]: hello world 13
[ INFO] [1567209369.284732317]: hello world 14
[ INFO] [1567209369.384752488]: hello world 15
[ INFO] [1567209369.484772846]: hello world 16
[ INFO] [1567209369.584798485]: hello world 17
[ INFO] [1567209369.684763541]: hello world 18
[ INFO] [1567209369.784781674]: hello world 19
[ INFO] [1567209369.884801013]: hello world 20
[ INFO] [1567209369.984798947]: hello world 21
[ INFO] [1567209370.084707976]: hello world 22
[ INFO] [1567209370.184756887]: hello world 23
[ INFO] [1567209370.284791587]: hello world 24
[ INFO] [1567209370.384794507]: hello world 25
```

6.3. Terminal 3:

- (C++) *rosrun beginner_tutorials listener*
- (Python) *rosrun beginner_tutorials listener.py*



```
moguztas@moguztas:~/ros_ws$ rosrun beginner_tutorials listener.py
[INFO] [1567209575.599204]: /listener_16717_1567209575367I heard hello world 165
[INFO] [1567209575.699208]: /listener_16717_1567209575367I heard hello world 166
[INFO] [1567209575.799234]: /listener_16717_1567209575367I heard hello world 167
[INFO] [1567209575.899221]: /listener_16717_1567209575367I heard hello world 168
[INFO] [1567209575.999226]: /listener_16717_1567209575367I heard hello world 169
[INFO] [1567209576.099204]: /listener_16717_1567209575367I heard hello world 170
[INFO] [1567209576.199212]: /listener_16717_1567209575367I heard hello world 171
[INFO] [1567209576.299228]: /listener_16717_1567209575367I heard hello world 172
[INFO] [1567209576.399248]: /listener_16717_1567209575367I heard hello world 173
[INFO] [1567209576.499223]: /listener_16717_1567209575367I heard hello world 174
[INFO] [1567209576.599137]: /listener_16717_1567209575367I heard hello world 175
[INFO] [1567209576.699225]: /listener_16717_1567209575367I heard hello world 176
[INFO] [1567209576.799226]: /listener_16717_1567209575367I heard hello world 177
[INFO] [1567209576.899234]: /listener_16717_1567209575367I heard hello world 178
[INFO] [1567209576.999163]: /listener_16717_1567209575367I heard hello world 179
[INFO] [1567209577.099210]: /listener_16717_1567209575367I heard hello world 180
[INFO] [1567209577.199122]: /listener_16717_1567209575367I heard hello world 181
[INFO] [1567209577.299158]: /listener_16717_1567209575367I heard hello world 182
[INFO] [1567209577.399200]: /listener_16717_1567209575367I heard hello world 183
[INFO] [1567209577.499195]: /listener_16717_1567209575367I heard hello world 184
[INFO] [1567209577.599238]: /listener_16717_1567209575367I heard hello world 185
[INFO] [1567209577.699100]: /listener_16717_1567209575367I heard hello world 186
[INFO] [1567209577.799165]: /listener_16717_1567209575367I heard hello world 187
[INFO] [1567209577.899168]: /listener_16717_1567209575367I heard hello world 188
[INFO] [1567209577.999260]: /listener_16717_1567209575367I heard hello world 189
[INFO] [1567209578.099261]: /listener_16717_1567209575367I heard hello world 190
```

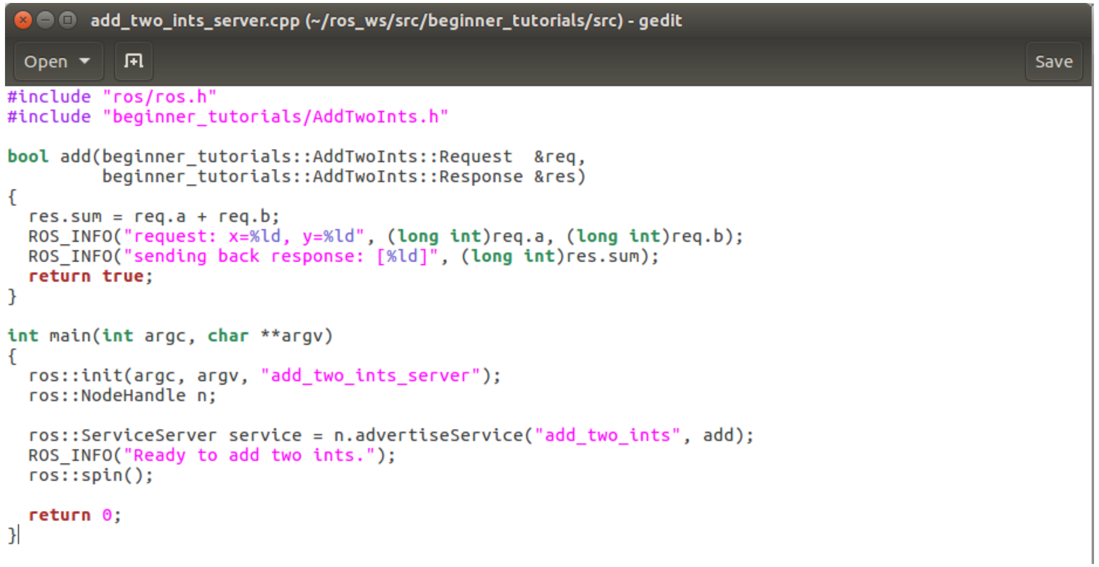
Application 6: Service-Client Application– 1/5
C++ Application – 1/2

1. Let's go to the src folder under beginner_tutorials.

roscd beginner_tutorials/src

2. Let's create Server file.

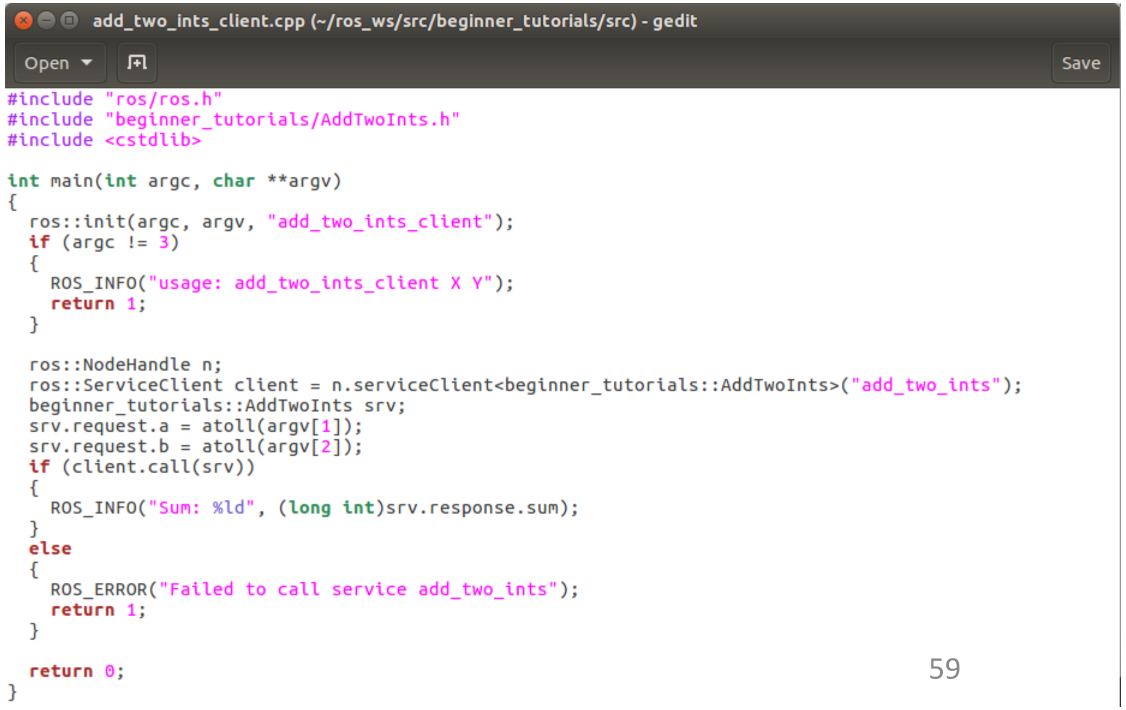
gedit add_two_ints_server.cpp


add_two_ints_server.cpp (~/ros_ws/src/beginner_tutorials/src) - gedit
Open
Save

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"

bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
  ros::NodeHandle n;

  ros::ServiceServer service = n.advertiseService("add_two_ints", add);
  ROS_INFO("Ready to add two ints.");
  ros::spin();

  return 0;
}
```

3. Let’s create Client file

gedit add_two_ints_client.cpp


add_two_ints_client.cpp (~/ros_ws/src/beginner_tutorials/src) - gedit
Open
Save

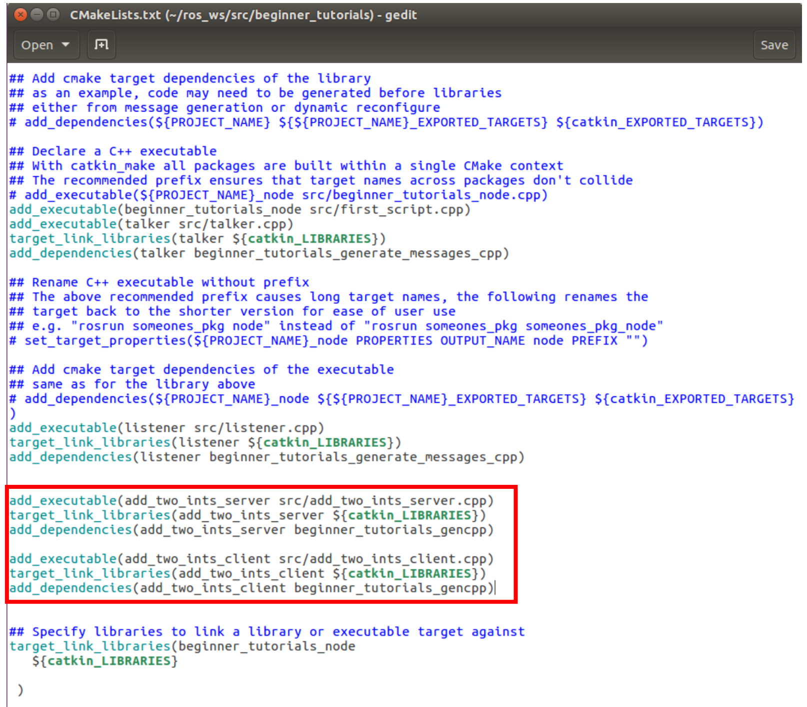
```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_client");
  if (argc != 3)
  {
    ROS_INFO("usage: add_two_ints_client X Y");
    return 1;
  }

  ros::NodeHandle n;
  ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
  if (client.call(srv))
  {
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

Application 6: Service-Client Application– 2/5
C++ Application – 2/2

4. Let's edit CMakeLists.txt file.

roscd beginner_tutorials

gedit CMakeLists.txt

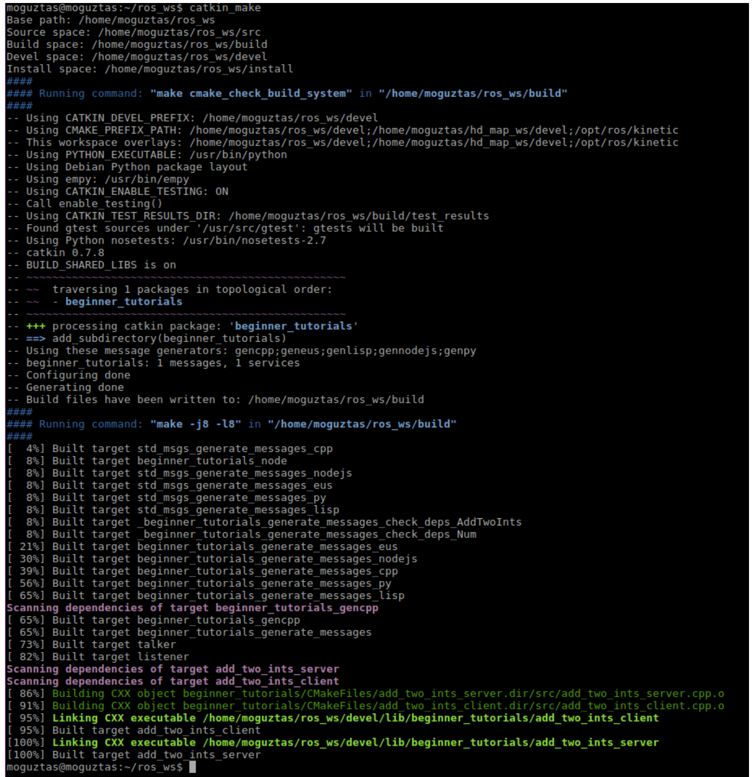


```
## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/beginner_tutorials_node.cpp)
add_executable(beginner_tutorials_node src/first_script.cpp)
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS}
)
add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)


add_executable(add_two_ints_server src/add_two_ints_server.cpp)
target_link_libraries(add_two_ints_server ${catkin_LIBRARIES})
add_dependencies(add_two_ints_server beginner_tutorials_gencpp)

add_executable(add_two_ints_client src/add_two_ints_client.cpp)
target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
add_dependencies(add_two_ints_client beginner_tutorials_gencpp)


## Specify libraries to link a library or executable target against
target_link_libraries(beginner_tutorials_node
   ${catkin_LIBRARIES}

 )
```

5. Compile workspace.

cd ~/ros_ws

catkin_make



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 1 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
[  4%] Built target std_msgs_generate_messages_cpp
[  8%] Built target beginner_tutorials_node
[  8%] Built target std_msgs_generate_messages_nodejs
[  8%] Built target std_msgs_generate_messages_eus
[  8%] Built target std_msgs_generate_messages_py
[  8%] Built target std_msgs_generate_messages_lisp
[  8%] Built target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[  8%] Built target _beginner_tutorials_generate_messages_check_deps_Num
[ 21%] Built target beginner_tutorials_generate_messages_eus
[ 30%] Built target beginner_tutorials_generate_messages_nodejs
[ 39%] Built target beginner_tutorials_generate_messages_cpp
[ 56%] Built target beginner_tutorials_generate_messages_py
[ 65%] Built target beginner_tutorials_generate_messages_lisp
Scanning dependencies of target beginner_tutorials_gencpp
[ 65%] Built target beginner_tutorials_gencpp
[ 65%] Built target beginner_tutorials_generate_messages
[ 73%] Built target talker
[ 82%] Built target listener
Scanning dependencies of target add_two_ints_server
Scanning dependencies of target add_two_ints_client
[ 86%] Building CXX object beginner_tutorials/CMakeFiles/add_two_ints_server.dir/src/add_two_ints_server.cpp.o
[ 91%] Building CXX object beginner_tutorials/CMakeFiles/add_two_ints_client.dir/src/add_two_ints_client.cpp.o
[ 95%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials/add_two_ints_client
[ 95%] Built target add_two_ints_client
[100%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials/add_two_ints_server
[100%] Built target add_two_ints_server
moguztas@moguztas:~/ros_ws$
```

Application 6: Service-Client Application– 3/5 Python Application – 1/2

1. Let's go to beginner_tutorials folder and create scripts folder.

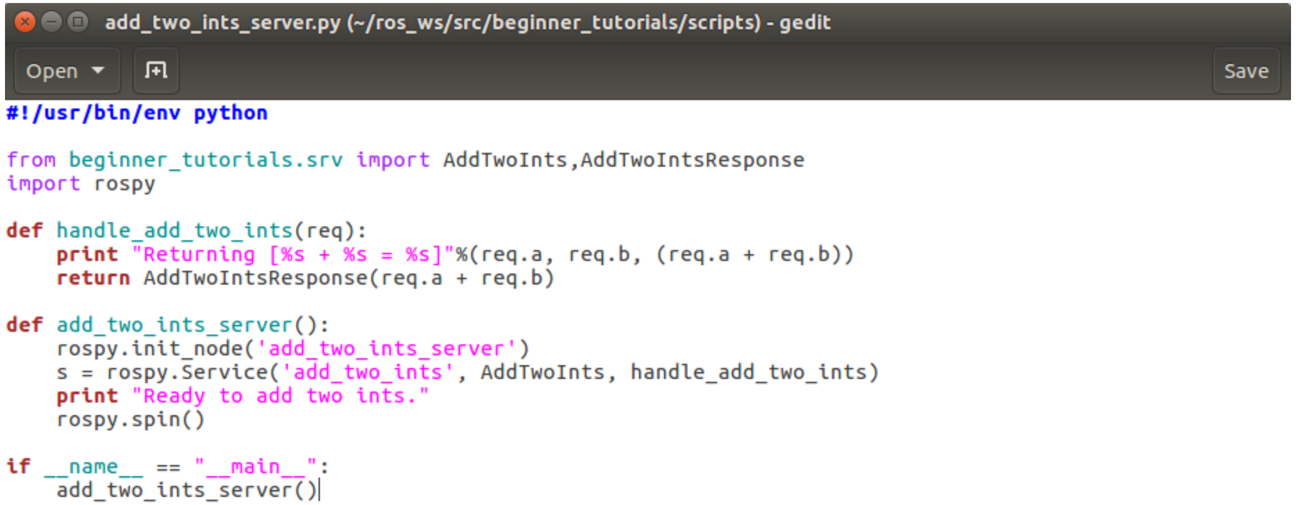
roscd beginner_tutorials/src

mkdir scripts

cd scripts

2. Let's create Server file.

gedit add_two_ints_server.py

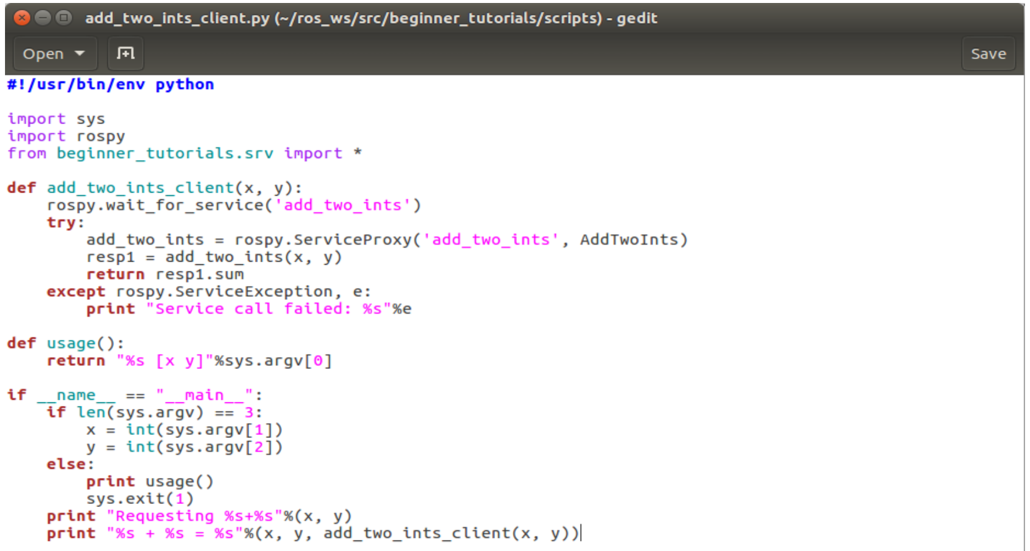


add_two_ints_server.py (~/ros_ws/src/beginner_tutorials/scripts) - gedit
Open
Save

```
#!/usr/bin/env python

from beginner_tutorials.srv import AddTwoInts,AddTwoIntsResponse
import rospy

def handle_add_two_ints(req):
    print "Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b))
    return AddTwoIntsResponse(req.a + req.b)

def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print "Ready to add two ints."
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

3. Let’s create Client file.

gedit add_two_ints_client.py



add_two_ints_client.py (~/ros_ws/src/beginner_tutorials/scripts) - gedit
Open
Save

```
#!/usr/bin/env python

import sys
import rospy
from beginner_tutorials.srv import *

def add_two_ints_client(x, y):
    rospy.wait_for_service('add_two_ints')
    try:
        add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException, e:
        print "Service call failed: %s"%e

def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        print usage()
        sys.exit(1)
    print "Requesting %s+%s"%(x, y)
    print "%s + %s = %s"%(x, y, add_two_ints_client(x, y))
```

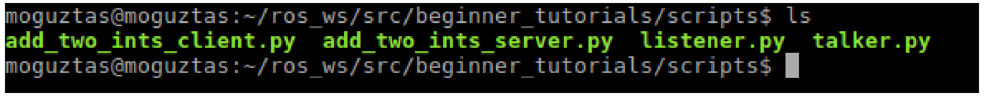
Application 6: Service-Client Application – 4/5 Python Uygulaması – 2/2

4. Let's make our files executable.

chmod +x add_two_ints_server.py

chmod +x add_two_ints_client.py

ls

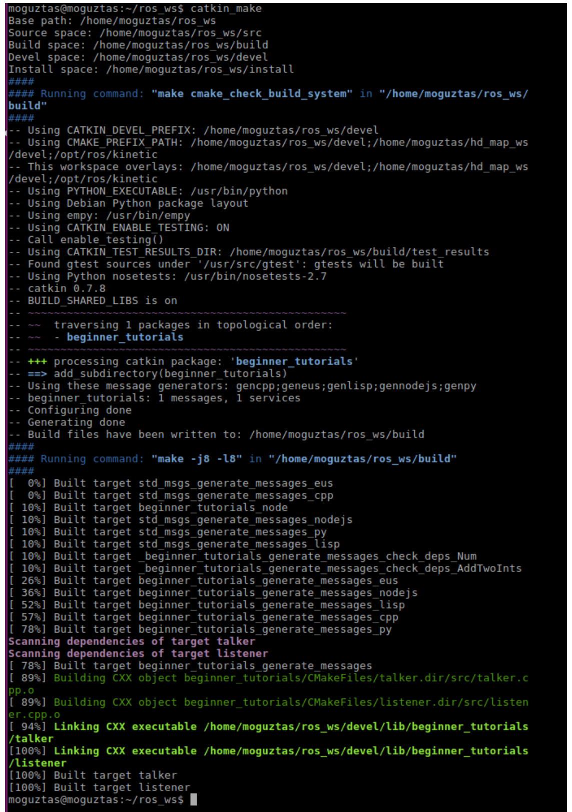


```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ ls
add_two_ints_client.py  add_two_ints_server.py  listener.py  talker.py
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/scripts$ 
```

5. Compile workspace.

cd ~/ros_ws

catkin_make



```
moguztas@moguztas:~/ros_ws$ catkin_make
Base path: /home/moguztas/ros_ws
Source space: /home/moguztas/ros_ws/src
Build space: /home/moguztas/ros_ws/build
Devel space: /home/moguztas/ros_ws/devel
Install space: /home/moguztas/ros_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/moguztas/ros_ws/
build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/moguztas/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- This workspace overlays: /home/moguztas/ros_ws/devel;/home/moguztas/hd_map_ws
/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/moguztas/ros_ws/build/test_results
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.8
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 1 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moguztas/ros_ws/build
####
#### Running command: "make -j8 -l8" in "/home/moguztas/ros_ws/build"
####
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_cpp
[ 10%] Built target beginner_tutorials_node
[ 10%] Built target std_msgs_generate_messages_nodejs
[ 10%] Built target std_msgs_generate_messages_py
[ 10%] Built target std_msgs_generate_messages_lisp
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_Num
[ 10%] Built target _beginner_tutorials_generate_messages_check_deps_AddTwoInts
[ 26%] Built target beginner_tutorials_generate_messages_eus
[ 36%] Built target beginner_tutorials_generate_messages_nodejs
[ 52%] Built target beginner_tutorials_generate_messages_lisp
[ 57%] Built target beginner_tutorials_generate_messages_cpp
[ 78%] Built target beginner_tutorials_generate_messages_py
Scanning dependencies of target talker
Scanning dependencies of target listener
[ 78%] Built target beginner_tutorials_generate_messages
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/talker.dir/src/talker.c
pp.o
[ 89%] Building CXX object beginner_tutorials/CMakeFiles/listener.dir/src/listen
er.cpp.o
[ 94%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/talker
[100%] Linking CXX executable /home/moguztas/ros_ws/devel/lib/beginner_tutorials
/listener
[100%] Built target talker
[100%] Built target listener
moguztas@moguztas:~/ros_ws$ 
```

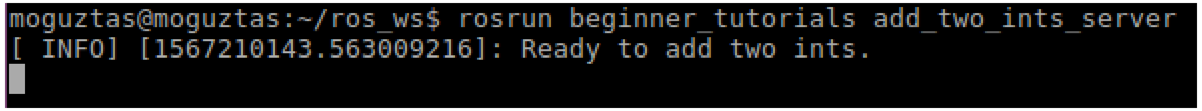
Application 6: Service-Client Application – 5/5

6. Running Application

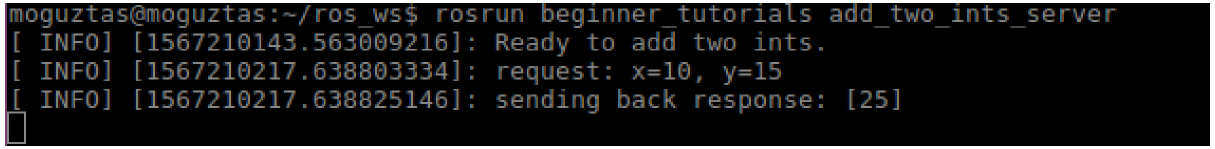
6.1. Terminal 1: *roscore*

6.2. Terminal 2:

- (C++) *rosrun beginner_tutorials add_two_ints_server*
- (Python) *rosrun beginner_tutorials add_two_ints_server.py*



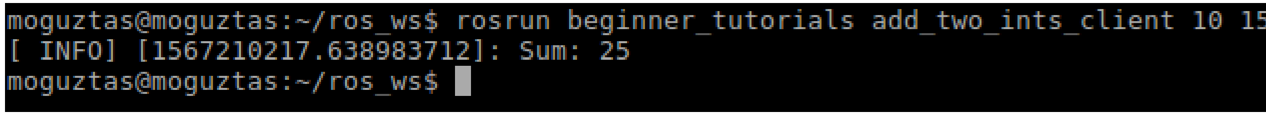
```
moguztas@moguztas:~/ros_ws$ rosrun beginner_tutorials add_two_ints_server
[ INFO] [1567210143.563009216]: Ready to add two ints.
```



```
moguztas@moguztas:~/ros_ws$ rosrun beginner_tutorials add_two_ints_server
[ INFO] [1567210143.563009216]: Ready to add two ints.
[ INFO] [1567210217.638803334]: request: x=10, y=15
[ INFO] [1567210217.638825146]: sending back response: [25]
```

6.3. Terminal 3:

- (C++) *rosrun beginner_tutorials add_two_ints_client 10 15*
- (Python) *rosrun beginner_tutorials add_two_ints_client.py 10 15*



```
moguztas@moguztas:~/ros_ws$ rosrun beginner_tutorials add_two_ints_client 10 15
[ INFO] [1567210217.638983712]: Sum: 25
moguztas@moguztas:~/ros_ws$
```

Application 7: Saving and Playing Data

1. Let's run roscore.

roscore

2. Let's open TurtleSim.

rosrun turtlesim turtlesim_node

3. Let's open the keyboard control node.

rosrun turtlesim turtle_teleop_key

4. Open the folder named bagfiles under the beginner_tutorials folder.

mkdir ~/bagfiles

cd ~/bagfiles

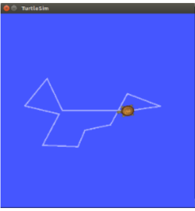
5.1. To save all published topics:

rosbag record –a

5.2. To record some topics:

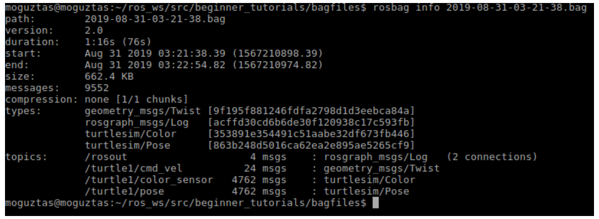
rosbag record -O subset /turtle1/cmd_vel /turtle1/pose

6. Let's move our robot with the help of the keyboard.

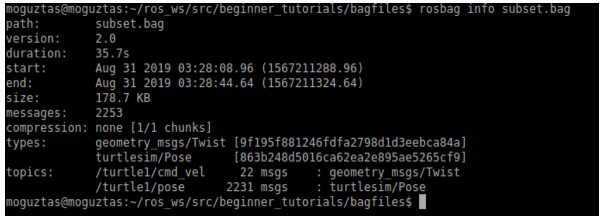


7. Let's check the content of our bag file.

rosbag info bag_file

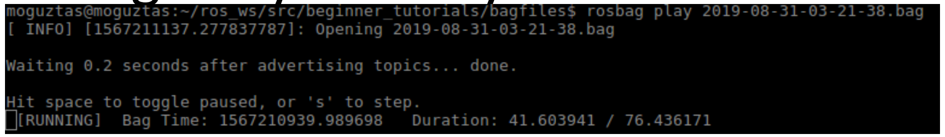


```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/bagfiles$ rosbag info 2019-08-31-03-21-38.bag
path:        2019-08-31-03-21-38.bag
version:     2.0
duration:    1:16s (76s)
start:       Aug 31 2019 03:21:38.39 (1567210898.39)
end:         Aug 31 2019 03:22:54.82 (1567210974.82)
size:        662.4 KB
messages:    9552
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
             turtlesim/Color     [353891e354491c51aabe32df673fb446]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /rosout                    4 msgs    : rosgraph_msgs/Log   (2 connections)
             /turtle1/cmd_vel          24 msgs    : geometry_msgs/Twist
             /turtle1/color_sensor   4762 msgs    : turtlesim/Color
             /turtle1/pose           4762 msgs    : turtlesim/Pose
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/bagfiles$ 
```



```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/bagfiles$ rosbag info subset.bag
path:        subset.bag
version:     2.0
duration:    35.7s
start:       Aug 31 2019 03:28:08.96 (1567211288.96)
end:         Aug 31 2019 03:28:44.64 (1567211324.64)
size:        178.7 KB
messages:    2253
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /turtle1/cmd_vel     22 msgs    : geometry_msgs/Twist
             /turtle1/pose      2231 msgs    : turtlesim/Pose
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/bagfiles$ 
```

8. Bag dosyamızı oynatalım.



```
moguztas@moguztas:~/ros_ws/src/beginner_tutorials/bagfiles$ rosbag play 2019-08-31-03-21-38.bag
[ INFO] [1567211137.277837787]: Opening 2019-08-31-03-21-38.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING]  Bag Time: 1567210939.989698   Duration: 41.603941 / 76.436171
```

- **O** argument tells the rosbag record command to just follow and write these two topics in a file called **subset.bag.**

Question & Answer