ROS

ROBOT OPERATING SYSTEM

Amirreza Kabiri
Fatemeh Pahlevan Aghababa

Autumn 2017

- **Why ROS?**
- **Understanding ROS community level**
- **Levels of development in ROS**
- **Understanding the ROS file system level**
- **Understanding the ROS computation graph level**
- **Understanding ROS nodes, messages, topics, services, bags**
- **Understanding ROS Master**
- **Using ROS Parameter**
- **Running ROS Master and ROS Parameter server**
- **Creating a ROS package**
- **Working with ROS topics**
- **Adding custom msg and srv files**
- **Working with ROS services**
- **Working with ROS actionlib**
- **Creating launch files**
- **Applications of topics, services, and actionlib**

INTRODUCTION TO ROS

ROS-PART1

Amirreza Kabiri & Fatemeh Pahlevan @ Autumn 2017

WHY ROS?

- Robot Operating System (ROS)
  - supported by the **Open Source Robotics Foundation** (**OSRF**),
    - in 2007 with the name Switchyard
      - Willow Garage

- A meta operating system

✓ performing many functions of an operating system but it requires a computer's operating system such as Linux

✓ Provides communication between the user, the computer's operating system, and equipment external to the computer
including sensors, cameras, as well as robots

✓ and the ability to control a robot without the user having to know all of the details of the robot

- Other robot frameworks are such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

WHY WE PREFER ROS FOR ROBOTS?

- ✓ High-end capabilities
- ✓ Tons of tools
- ✓ Support high-end sensors and actuators
- ✓ Inter-platform operability
- ✓ Modularity
- ✓ Concurrent resource handling
- ✓ Active community

WHICH ROBOTS ARE USING ROS?

✓ **More than one hundered robots**

❑ For example

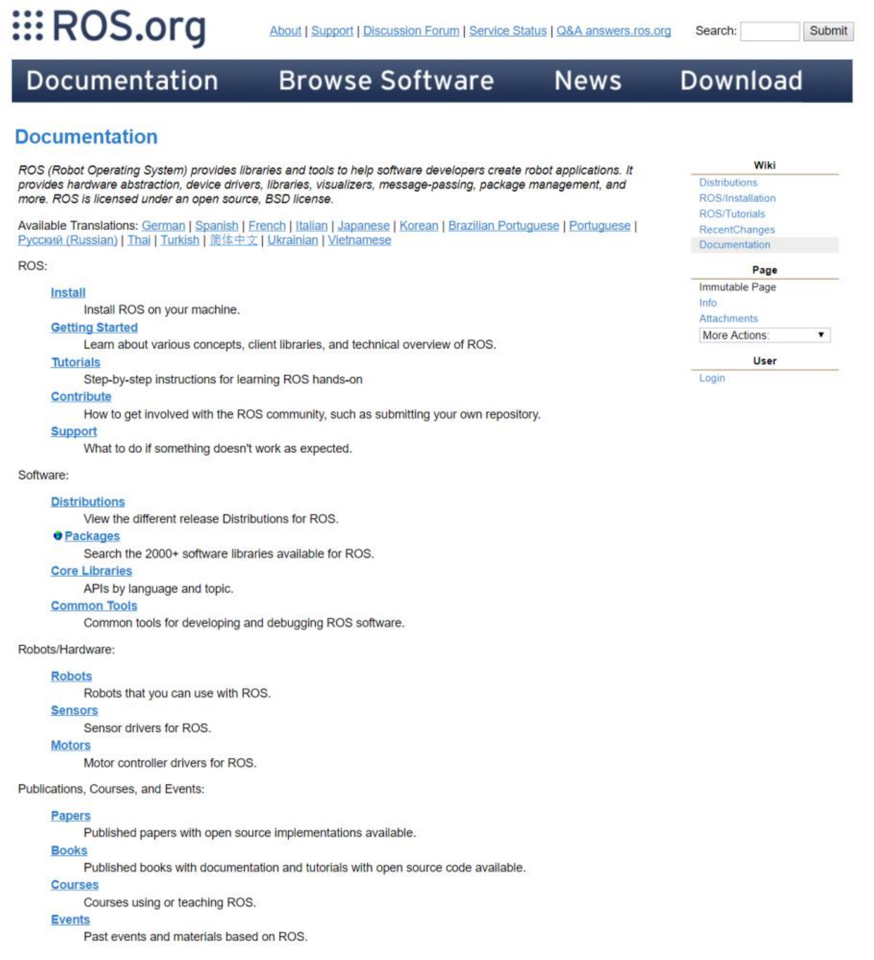
TurtleBot, a mobile robot
Baxter, a friendly two-armed robot
Crazyflie and Bebop, flying robots

❑ Complete list of robots

http://robots.ros.org/all/

WWW.ROS.ORG - THE ROS HUB

- ❑ A centralized location for ROS users and developers



Documentation

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

Available Translations: German | Spanish | French | Italian | Japanese | Korean | Brazilian Portuguese | Portuguese | Русский (Russian) | Thai | Turkish | 简体中文 | Ukrainian | Vietnamese

ROS:

Install
Install ROS on your machine.

Getting Started
Learn about various concepts, client libraries, and technical overview of ROS.

Tutorials
Step-by-step instructions for learning ROS hands-on

Contribute
How to get involved with the ROS community, such as submitting your own repository.

Support
What to do if something doesn't work as expected.

Software:

Distributions
View the different release Distributions for ROS.

Packages
Search the 2000+ software libraries available for ROS.

Core Libraries
APIs by language and topic.

Common Tools
Common tools for developing and debugging ROS software.

Robots/Hardware:

Robots
Robots that you can use with ROS.

Sensors
Sensor drivers for ROS.

Motors
Motor controller drivers for ROS.

Publications, Courses, and Events:

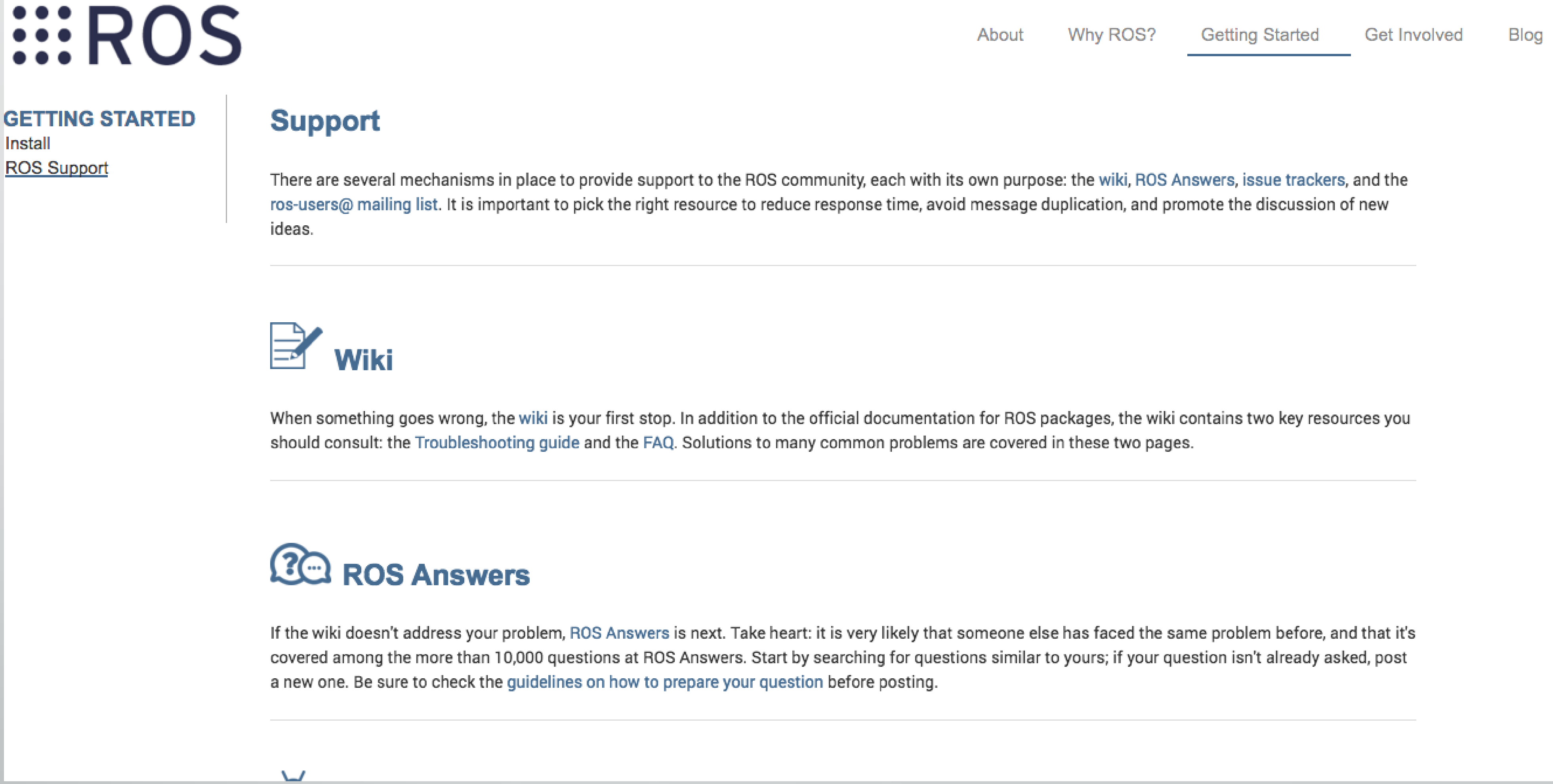
Papers
Published papers with open source implementations available.

Books
Published books with documentation and tutorials with open source code available.

Courses
Courses using or teaching ROS.

Events
Past events and materials based on ROS.

WWW.ROS.ORG - THE ROS HUB



GETTING STARTED
Install
ROS Support

Support

There are several mechanisms in place to provide support to the ROS community, each with its own purpose: the wiki, ROS Answers, issue trackers, and the ros-users@ mailing list. It is important to pick the right resource to reduce response time, avoid message duplication, and promote the discussion of new ideas.

Wiki

When something goes wrong, the wiki is your first stop. In addition to the official documentation for ROS packages, the wiki contains two key resources you should consult: the Troubleshooting guide and the FAQ. Solutions to many common problems are covered in these two pages.

ROS Answers

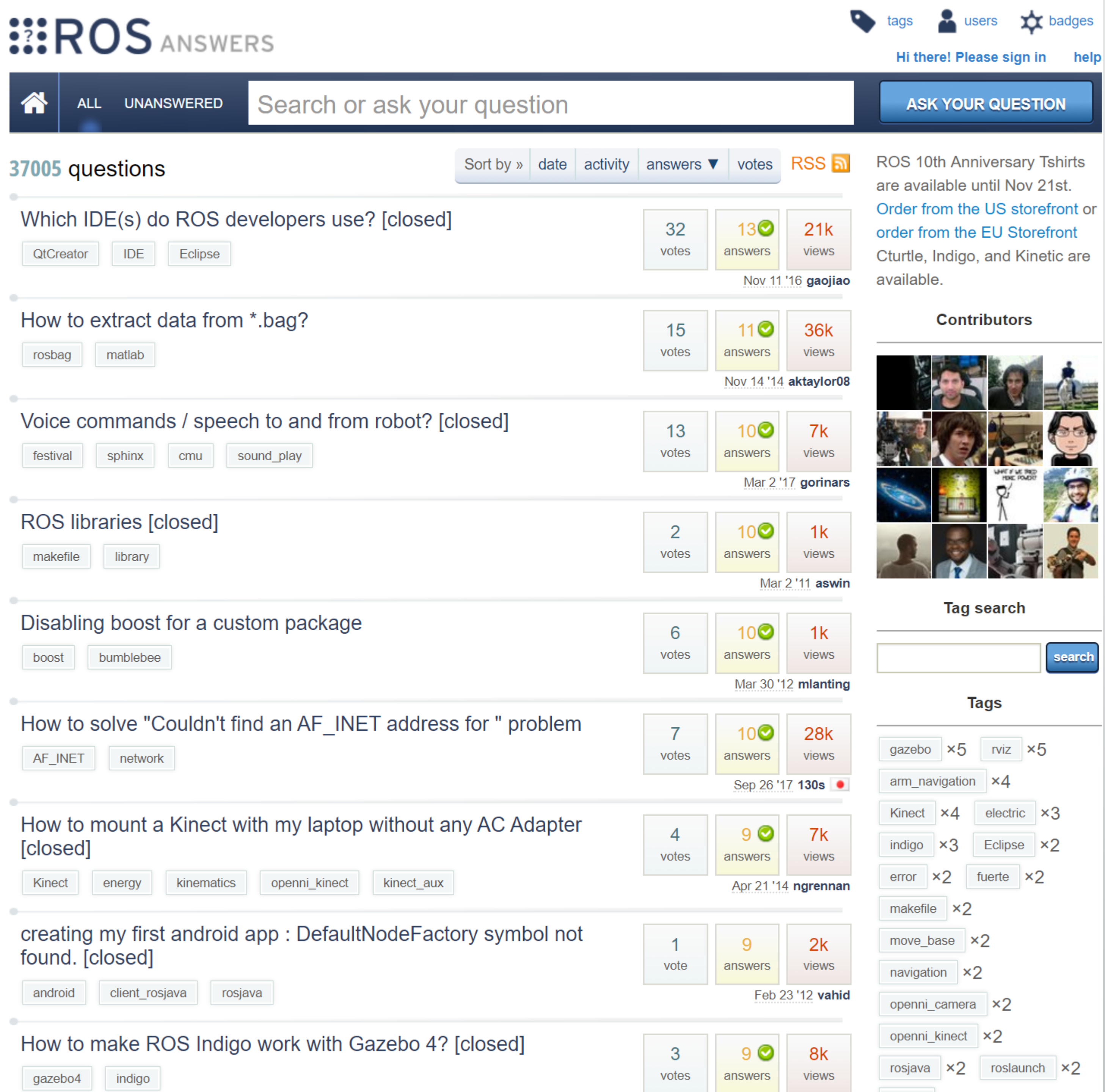
If the wiki doesn't address your problem, ROS Answers is next. Take heart: it is very likely that someone else has faced the same problem before, and that it's covered among the more than 10,000 questions at ROS Answers. Start by searching for questions similar to yours; if your question isn't already asked, post a new one. Be sure to check the guidelines on how to prepare your question before posting.

GROWING COMMUNITY

- ➢ answers.ros.org - ROS Questions & Answers

- ➢ industrial robotics trend:
- ➢ switching from proprietary robotic application to ROS

- ➢ Community-supported help for ROS users
  - ❑ 37005 question and answers
  - ❑ more than 15000 active users



ROS ANSWERS

tags users badges

Hi there! Please sign in help

ALL UNANSWERED Search or ask your question ASK YOUR QUESTION

37005 questions Sort by » date activity answers ▼ votes RSS

Question	Tags	Votes	Answers	Views	Asked
Which IDE(s) do ROS developers use? [closed]	QtCreator, IDE, Eclipse	32 votes	13 ✓ answers	21k views	Nov 11 '16 gaojiao
How to extract data from *.bag?	rosbag, matlab	15 votes	11 ✓ answers	36k views	Nov 14 '14 aktaylor08
Voice commands / speech to and from robot? [closed]	festival, sphinx, cmu, sound_play	13 votes	10 ✓ answers	7k views	Mar 2 '17 gorinars
ROS libraries [closed]	makefile, library	2 votes	10 ✓ answers	1k views	Mar 2 '11 aswin
Disabling boost for a custom package	boost, bumblebee	6 votes	10 ✓ answers	1k views	Mar 30 '12 mlanting
How to solve "Couldn't find an AF_INET address for " problem	AF_INET, network	7 votes	10 ✓ answers	28k views	Sep 26 '17 130s ●
How to mount a Kinect with my laptop without any AC Adapter [closed]	Kinect, energy, kinematics, openni_kinect, kinect_aux	4 votes	9 ✓ answers	7k views	Apr 21 '14 ngrennan
creating my first android app : DefaultNodeFactory symbol not found. [closed]	android, client_rosjava, rosjava	1 vote	9 answers	2k views	Feb 23 '12 vahid
How to make ROS Indigo work with Gazebo 4? [closed]	gazebo4, indigo	3 votes	9 ✓ answers	8k views	

ROS 10th Anniversary Tshirts are available until Nov 21st. Order from the US storefront or order from the EU Storefront Cturtle, Indigo, and Kinetic are available.

Contributors

Tag search

search

Tags

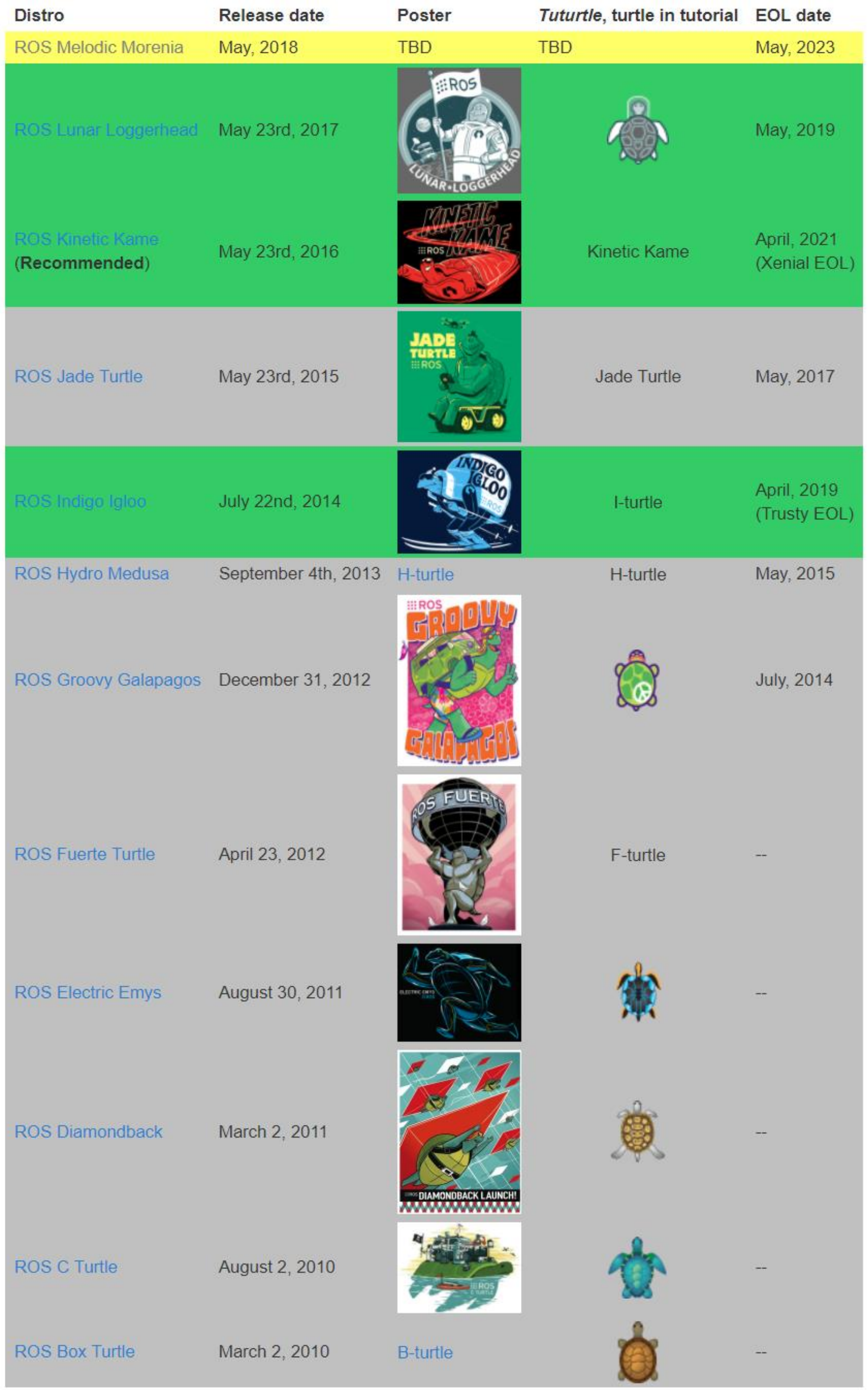
Tag	Count	Tag	Count
gazebo	×5	rviz	×5
arm_navigation	×4		
Kinect	×4	electric	×3
indigo	×3	Eclipse	×2
error	×2	fuerte	×2
makefile	×2		
move_base	×2		
navigation	×2		
openni_camera	×2		
openni_kinect	×2		
rosjava	×2	roslaunch	×2

Distributions

❑ A ROS distribution is a versioned set of ROS packages.

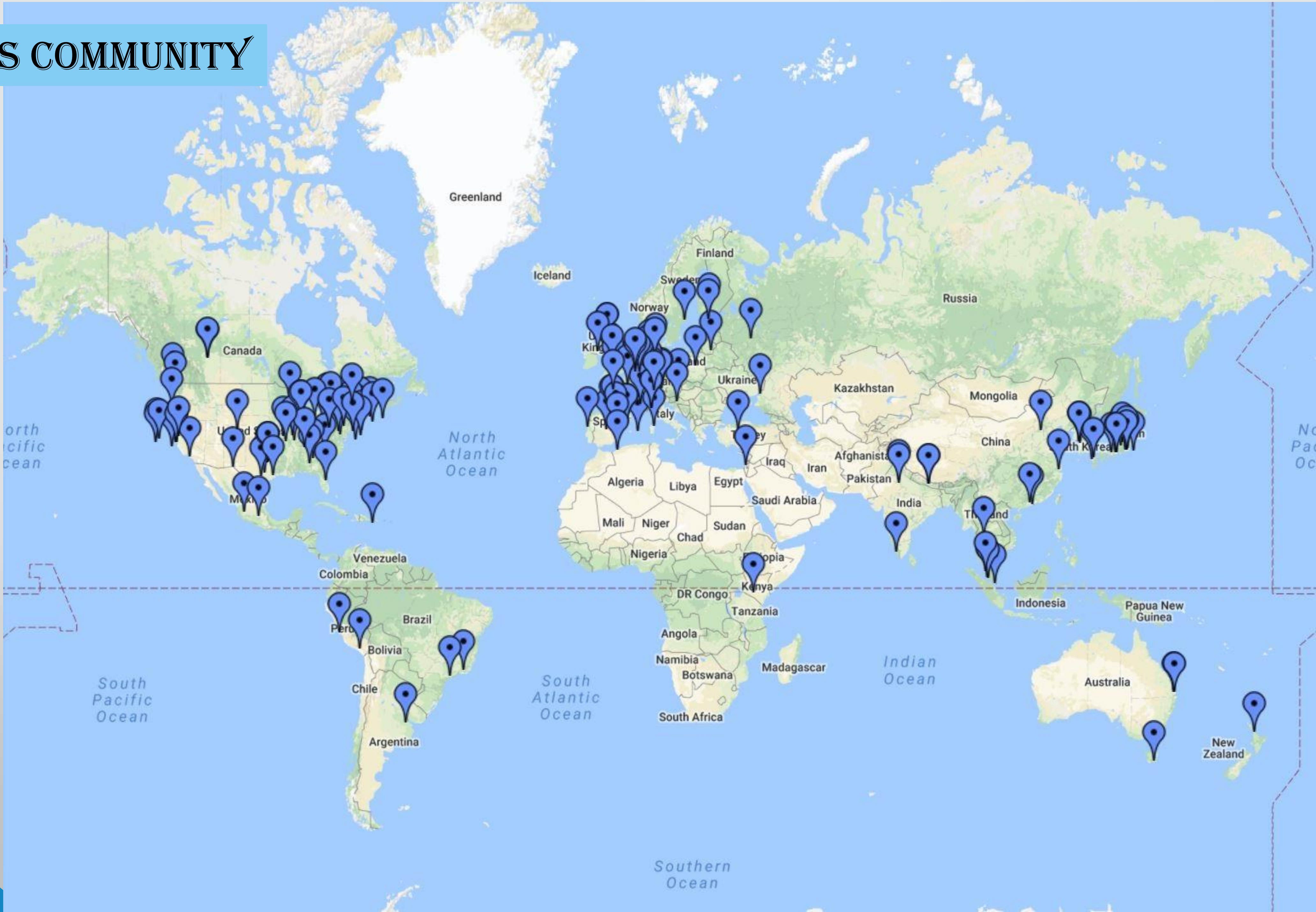
❑ Release rules:

- There is a ROS release every year in May.
- Releases on even numbered years will be a LTS release, supported for five years.
- Releases on odd numbered years are normal ROS releases, supported for two years.
- ROS releases will drop support for EOL Ubuntu distributions, even if the ROS release is still supported.



Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Melodic Morenia	May, 2018	TBD	TBD	May, 2023
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame (**Recommended**)	May 23rd, 2016		Kinetic Kame	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	H-turtle	May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012		F-turtle	--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010	B-turtle		--

THE ROS COMMUNITY



ROS MAILING LISTS

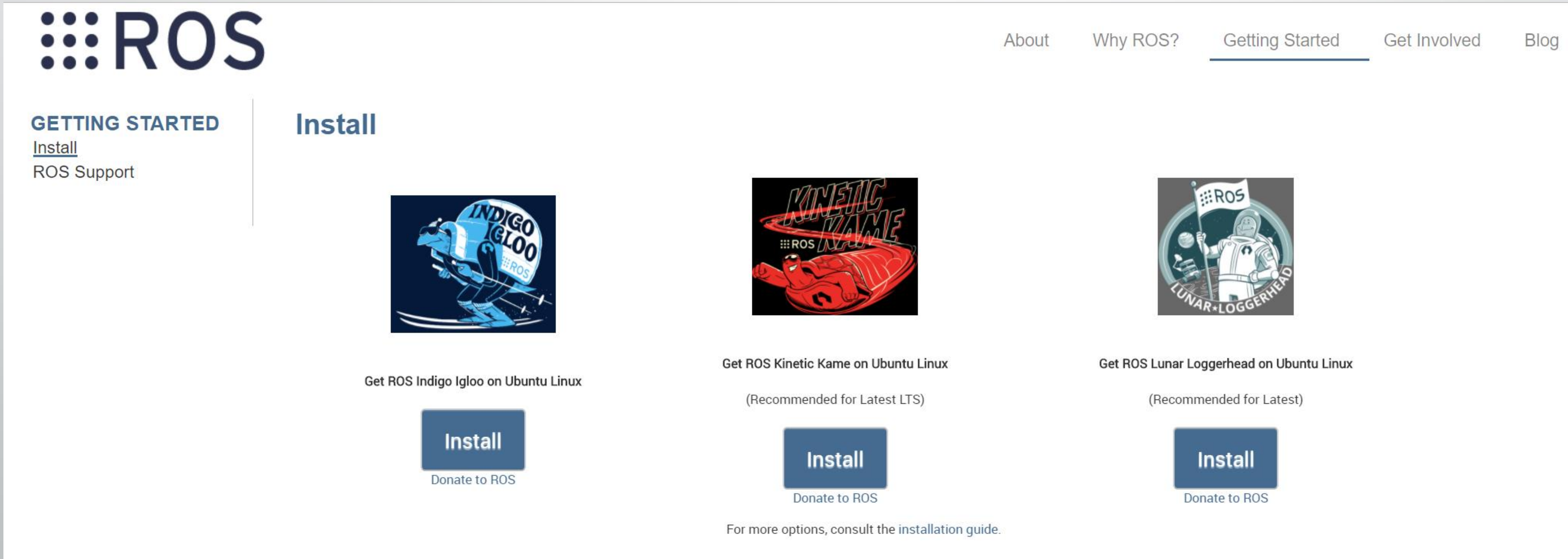
Getting in touch with the developer community

- ❑ http://lists.ros.org/lurker/list/ros-release.en.html — *ROS release maintainers*
- ❑ http://lists.ros.org/lurker/list/ros-users.en.html — *Discussions among ROS users.*

- ➢ To post a message to all the list members, send email to ros-users@lists.ros.org.

ROS INSTALLATION GUIDE

http://www.ros.org/install/



GETTING STARTED

Install

ROS Support

Install

Get ROS Indigo Igloo on Ubuntu Linux

Install

Donate to ROS

Get ROS Kinetic Kame on Ubuntu Linux

(Recommended for Latest LTS)

Install

Donate to ROS

Get ROS Lunar Loggerhead on Ubuntu Linux

(Recommended for Latest)

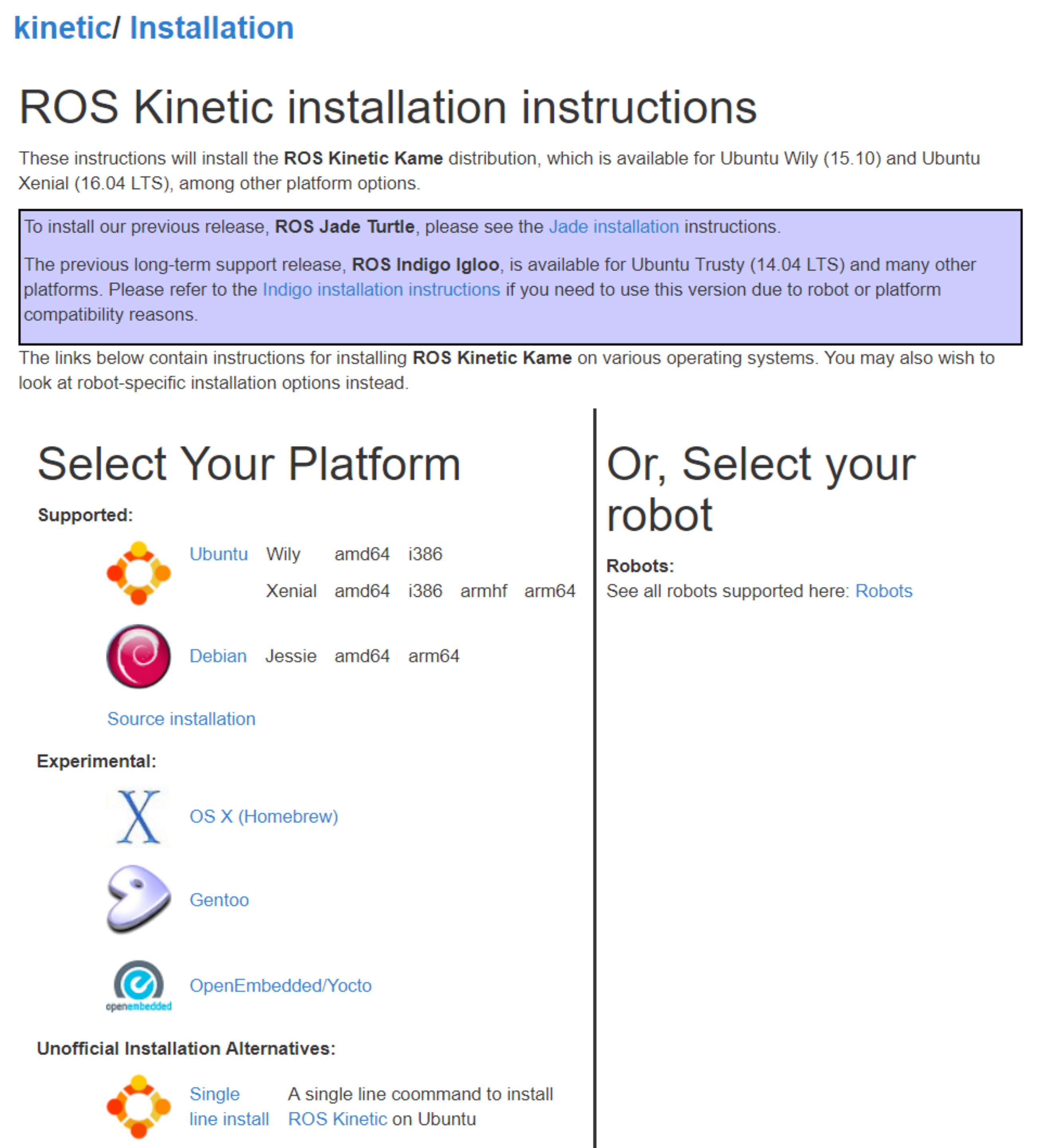
Install

Donate to ROS

For more options, consult the installation guide.

KINETIC INSTALLATION GUIDE

- http://wiki.ros.org/kinetic/Installation



kinetic/ Installation

ROS Kinetic installation instructions

These instructions will install the **ROS Kinetic Kame** distribution, which is available for Ubuntu Wily (15.10) and Ubuntu Xenial (16.04 LTS), among other platform options.

To install our previous release, **ROS Jade Turtle**, please see the Jade installation instructions.

The previous long-term support release, **ROS Indigo Igloo**, is available for Ubuntu Trusty (14.04 LTS) and many other platforms. Please refer to the Indigo installation instructions if you need to use this version due to robot or platform compatibility reasons.

The links below contain instructions for installing **ROS Kinetic Kame** on various operating systems. You may also wish to look at robot-specific installation options instead.

Select Your Platform

Supported:

Ubuntu	Wily	amd64	i386		
	Xenial	amd64	i386	armhf	arm64
Debian	Jessie	amd64	arm64		

Source installation

Experimental:

OS X (Homebrew)

Gentoo

OpenEmbedded/Yocto

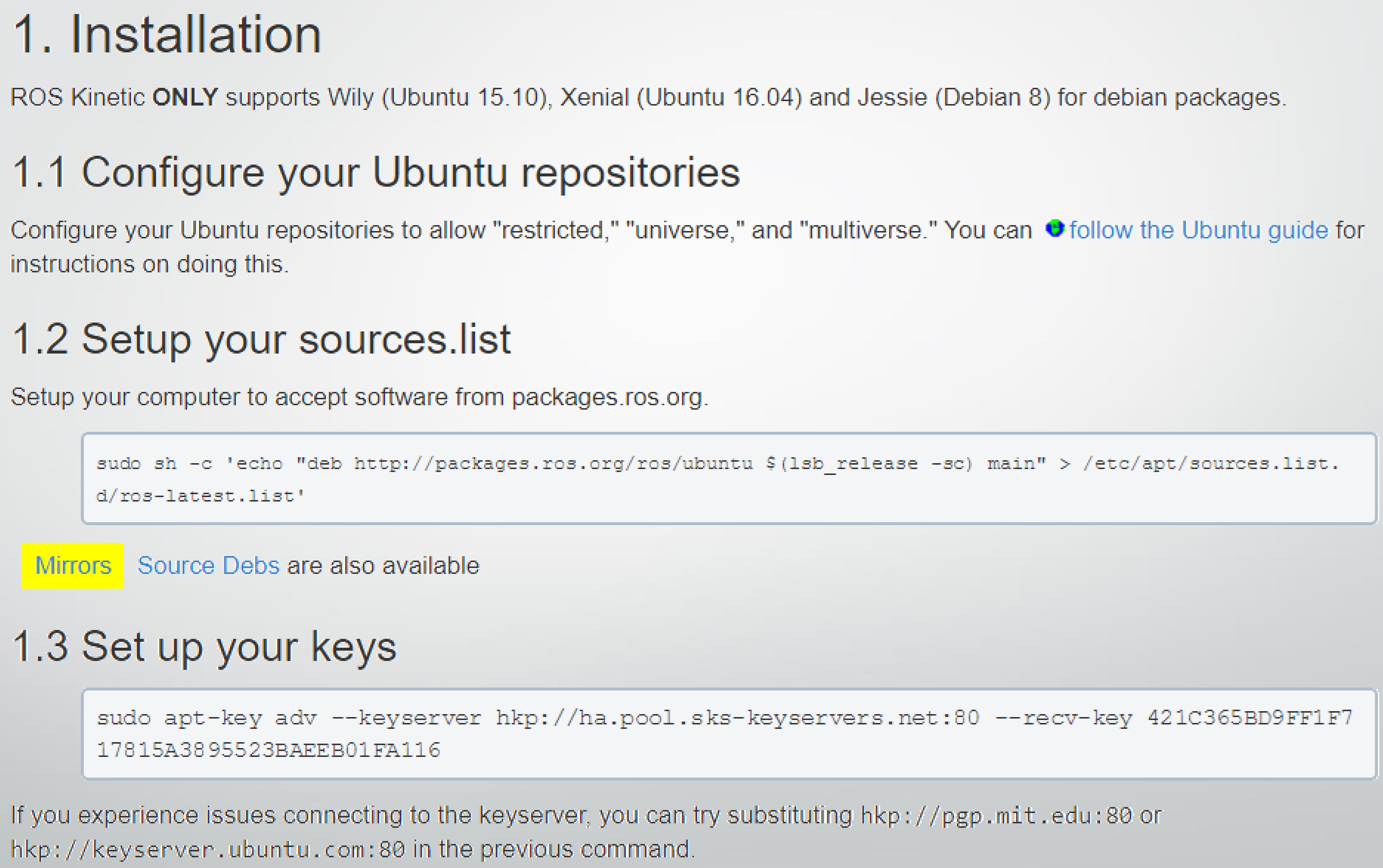
Unofficial Installation Alternatives:

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

Or, Select your robot

Robots:

See all robots supported here: Robots



1. Installation

ROS Kinetic **ONLY** supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages.

1.1 Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can follow the Ubuntu guide for instructions on doing this.

1.2 Setup your sources.list

Setup your computer to accept software from packages.ros.org.

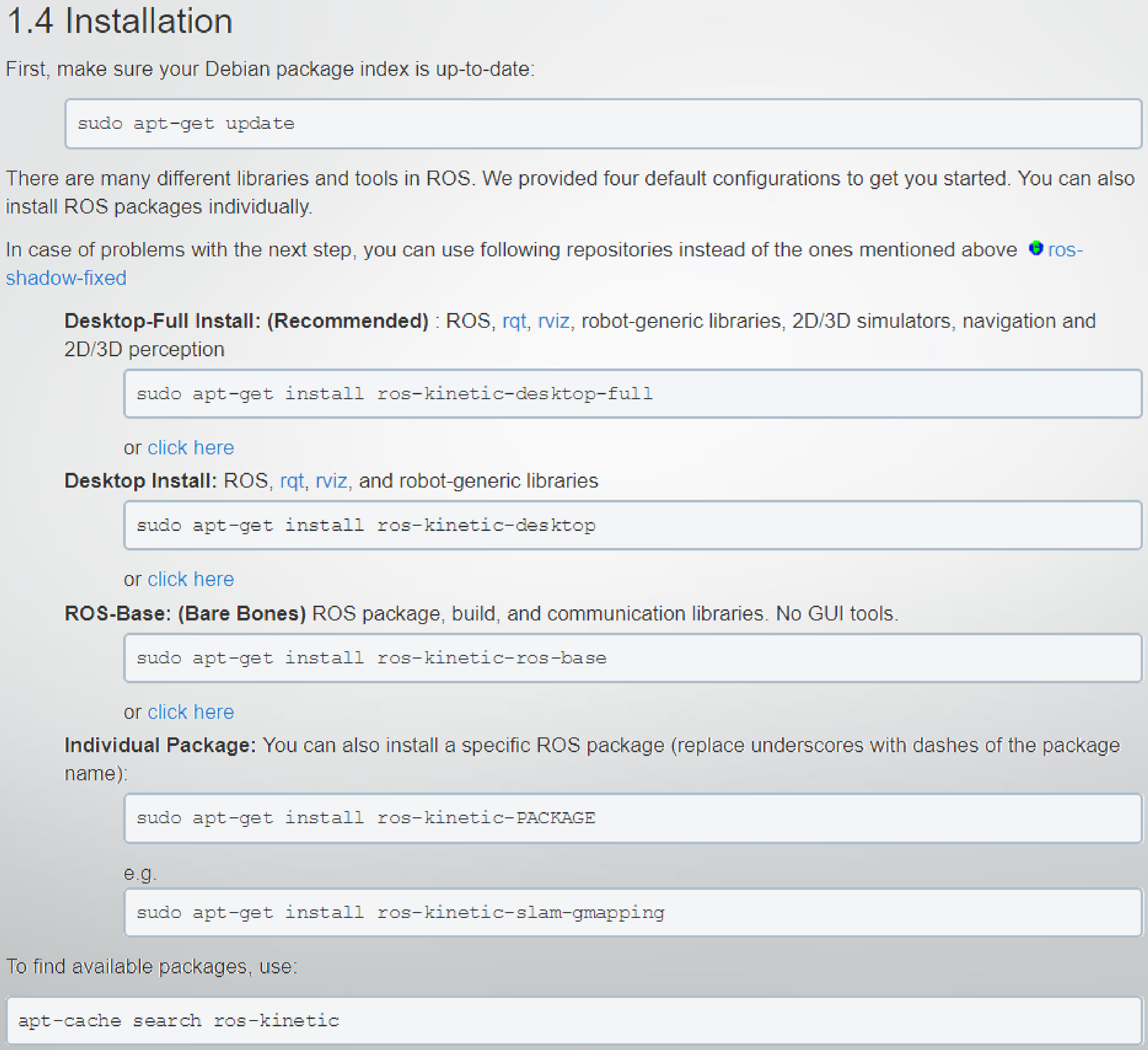
```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.
d/ros-latest.list'
```

Mirrors Source Debs are also available

1.3 Set up your keys

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F7
17815A3895523BAEEB01FA116
```

If you experience issues connecting to the keyserver, you can try substituting `hkp://pgp.mit.edu:80` or `hkp://keyserver.ubuntu.com:80` in the previous command.



1.4 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt-get update
```

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

In case of problems with the next step, you can use following repositories instead of the ones mentioned above ros-shadow-fixed

Desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

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

or click here

Desktop Install: ROS, rqt, rviz, and robot-generic libraries

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

or click here

ROS-Base: (Bare Bones) ROS package, build, and communication libraries. No GUI tools.

```
sudo apt-get install ros-kinetic-ros-base
```

or click here

Individual Package: You can also install a specific ROS package (replace underscores with dashes of the package name):

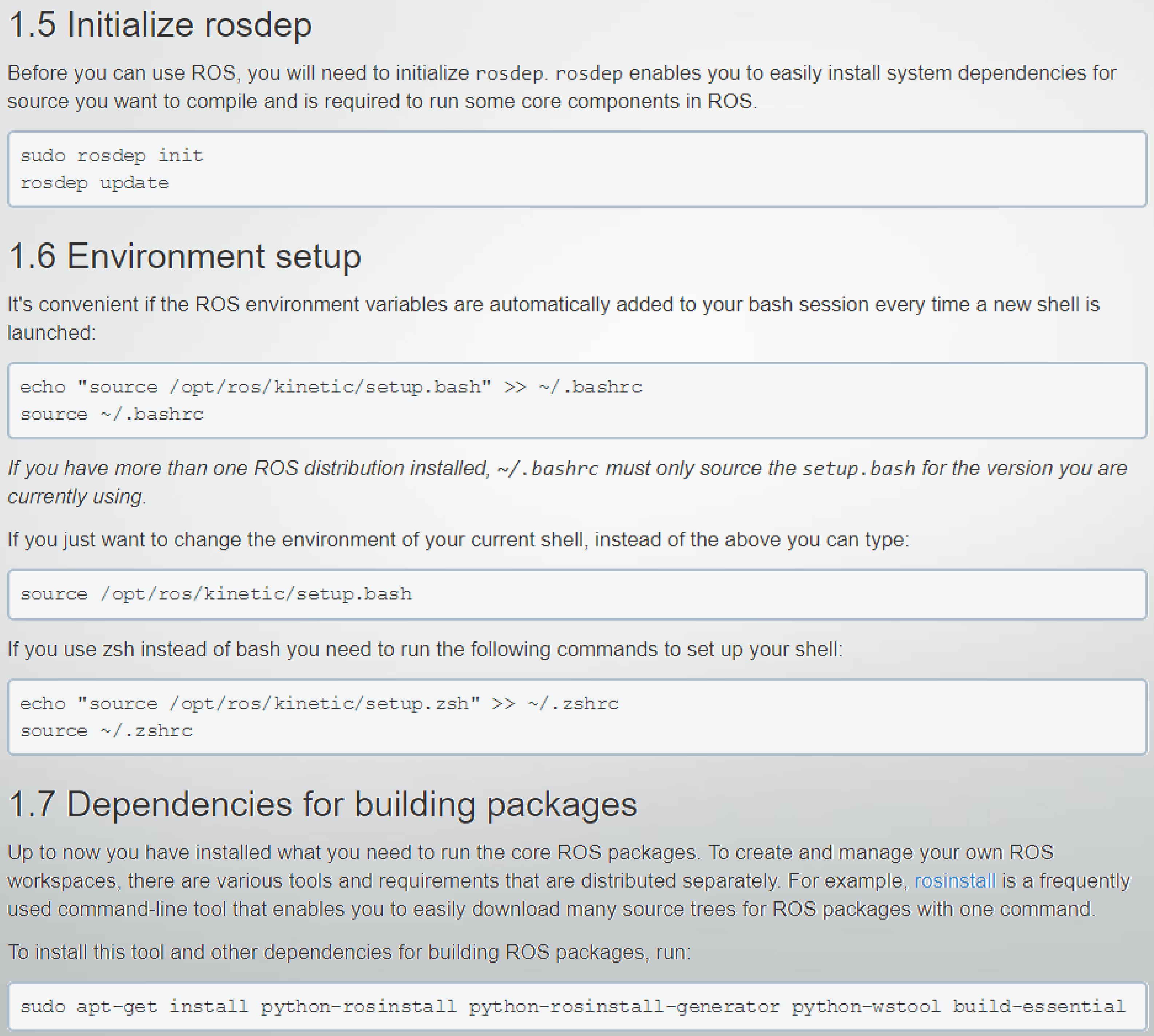
```
sudo apt-get install ros-kinetic-PACKAGE
```

e.g.

```
sudo apt-get install ros-kinetic-slam-gmapping
```

To find available packages, use:

```
apt-cache search ros-kinetic
```



1.5 Initialize rosdep

Before you can use ROS, you will need to initialize `rosdep`. `rosdep` enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

```
sudo rosdep init
rosdep update
```

1.6 Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

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

If you have more than one ROS distribution installed, `~/.bashrc` must only source the `setup.bash` for the version you are currently using.

If you just want to change the environment of your current shell, instead of the above you can type:

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

If you use zsh instead of bash you need to run the following commands to set up your shell:

```
echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

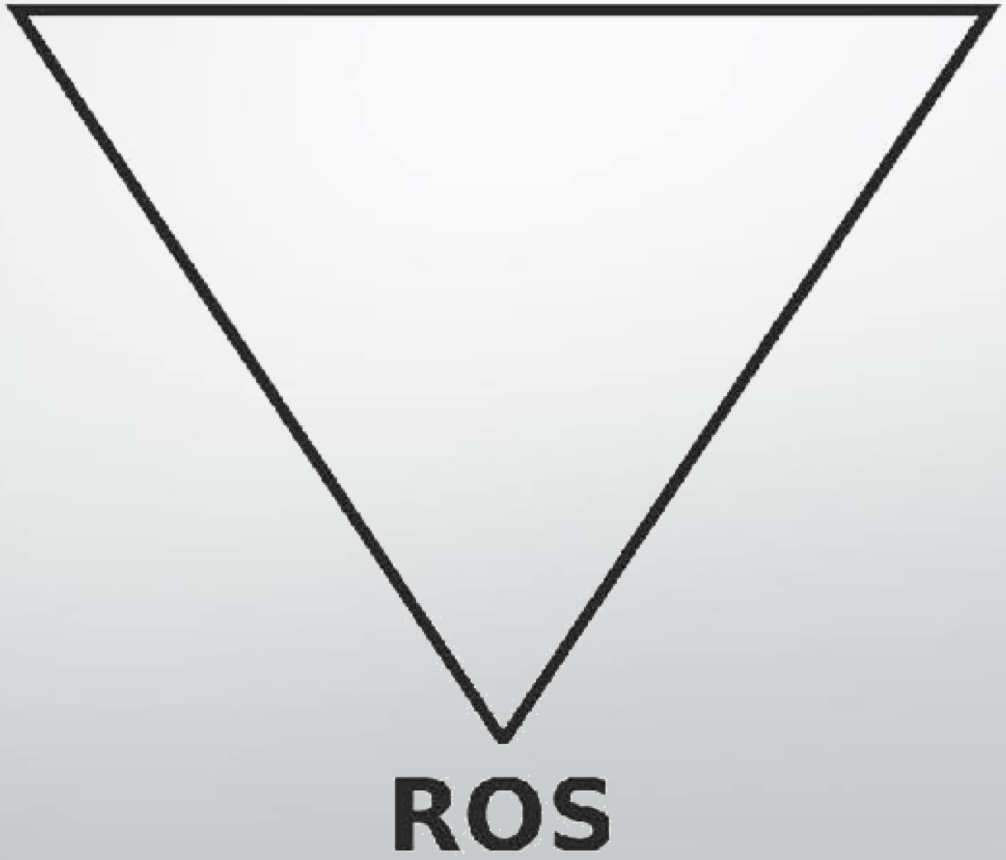
1.7 Dependencies for building packages

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, rosinstall is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

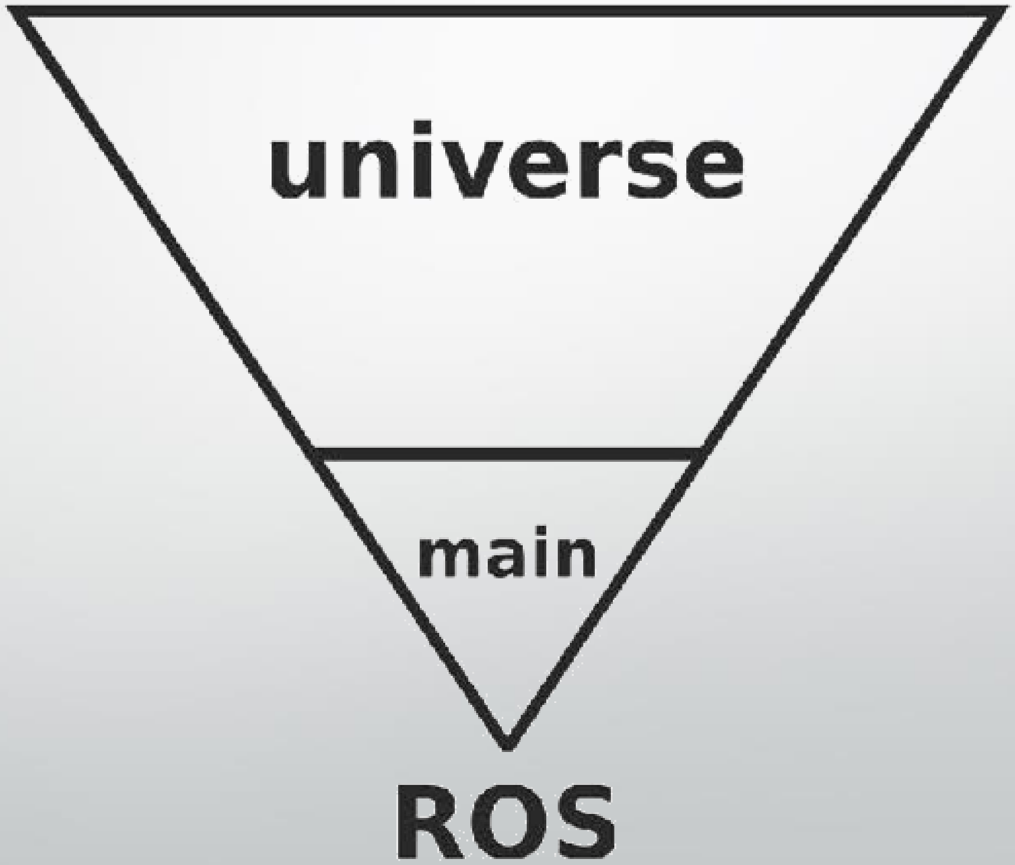
To install this tool and other dependencies for building ROS packages, run:

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

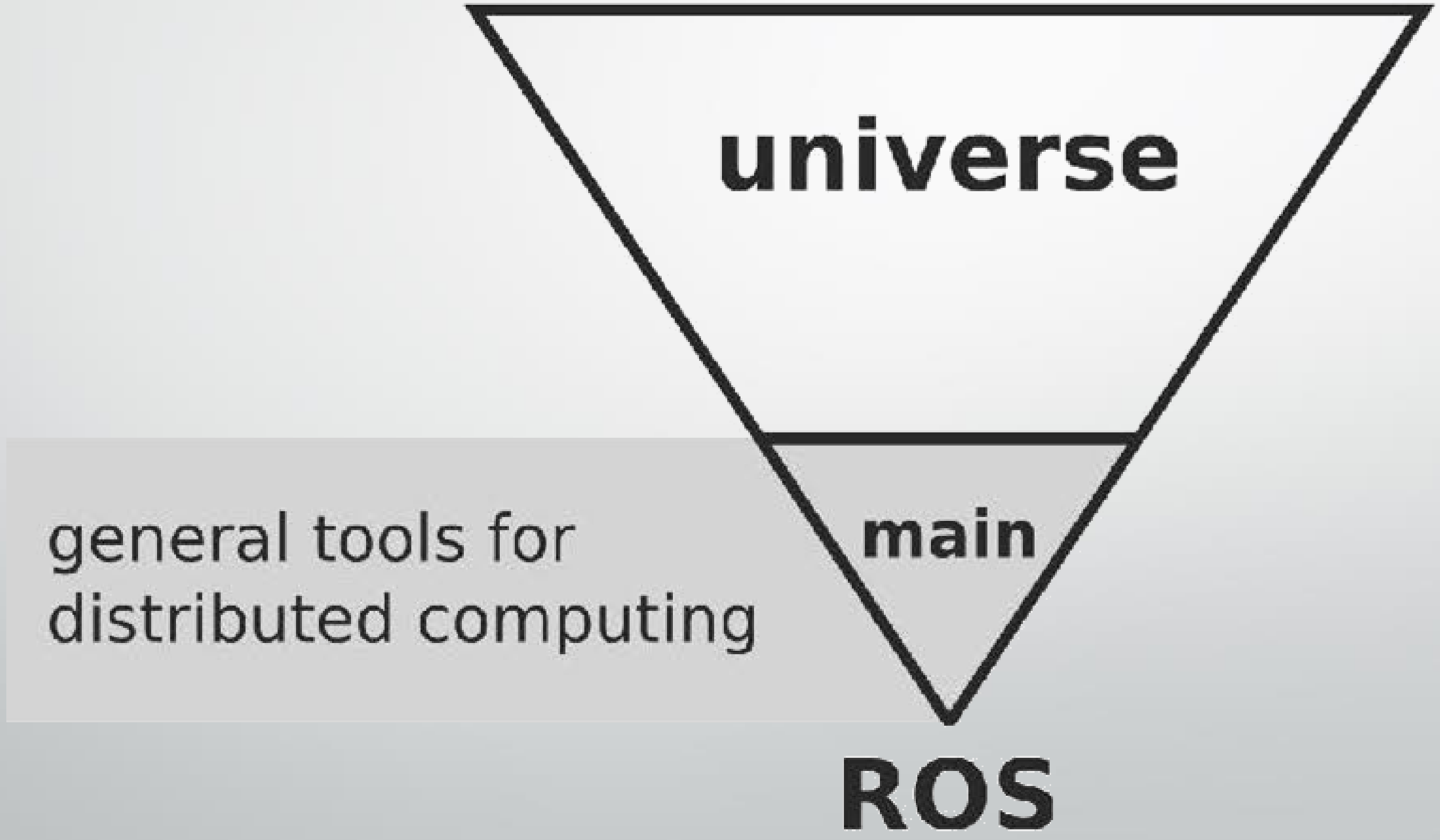
LEVELS OF DEVELOPMENT IN ROS



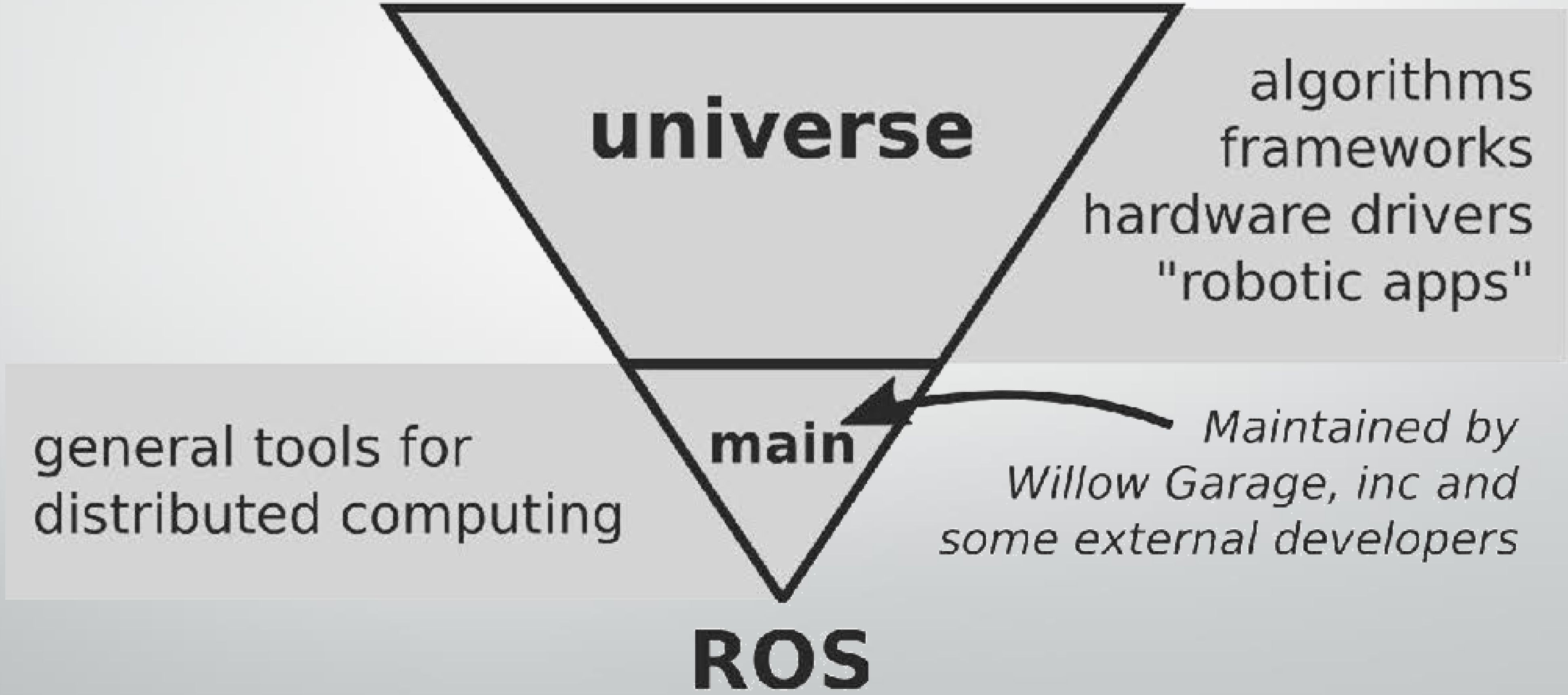
Levels of Development in ROS



Levels of Development in ROS

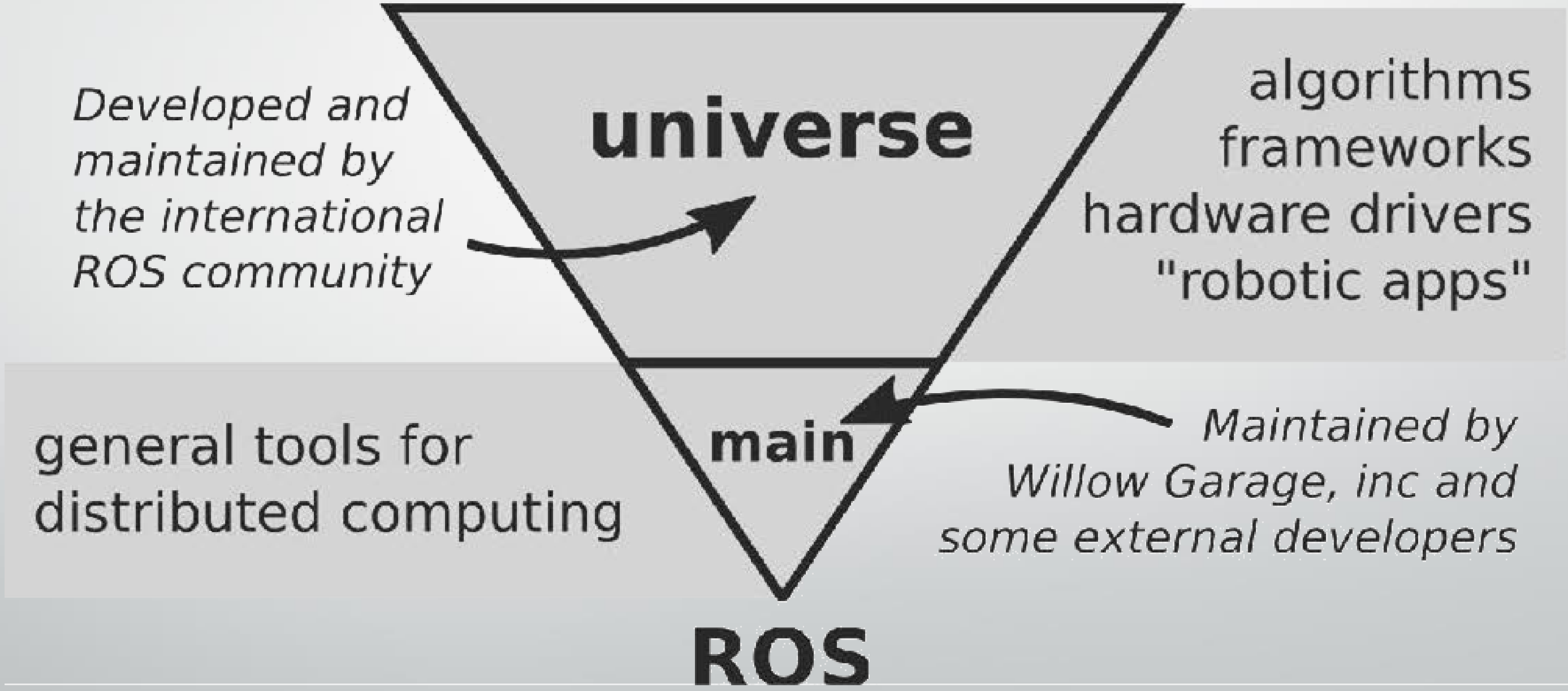


Levels of Development in ROS



universe
algorithms
frameworks
hardware drivers
"robotic apps"
main
general tools for distributed computing
Maintained by Willow Garage, inc and some external developers
ROS

Levels of Development in ROS



Developed and maintained by the international ROS community

universe

algorithms
frameworks
hardware drivers
"robotic apps"

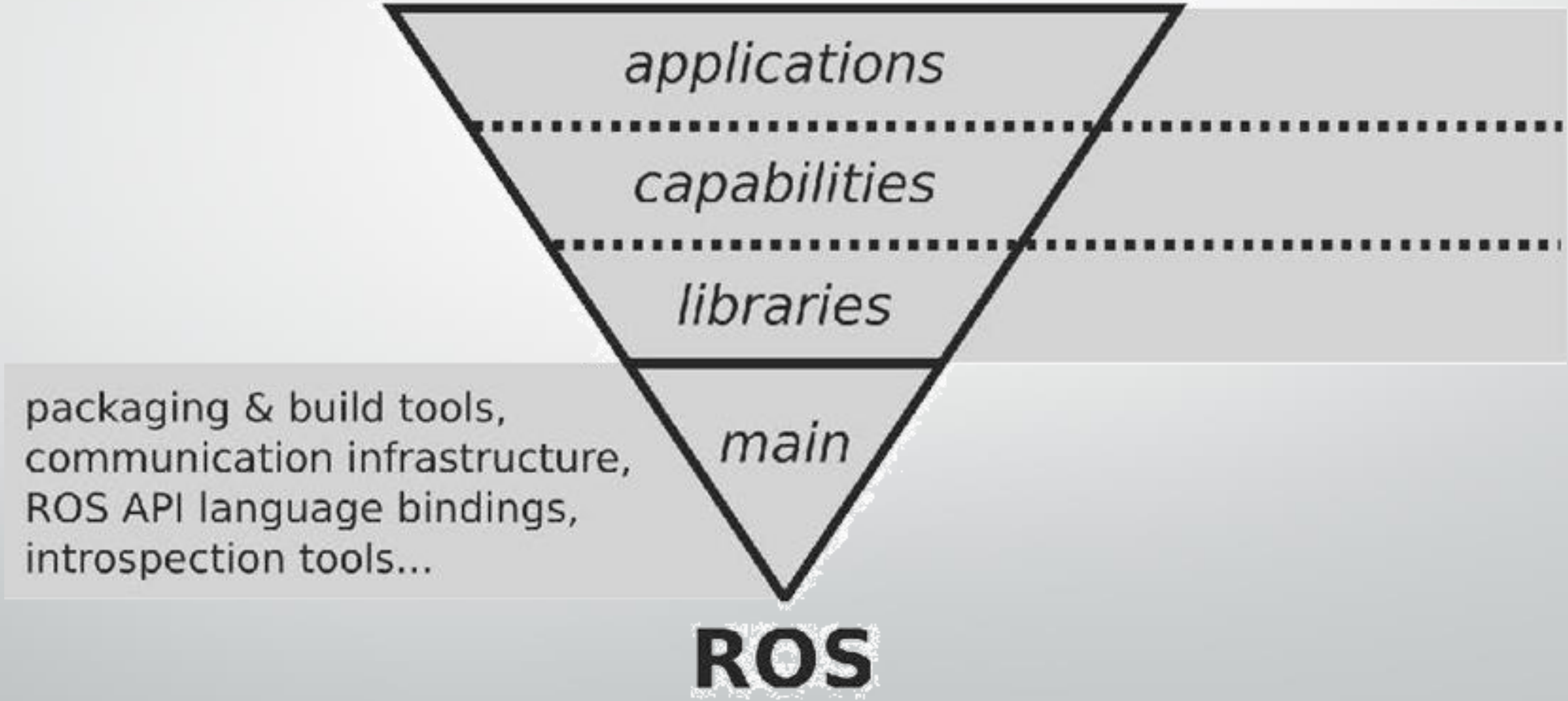
general tools for distributed computing

main

Maintained by Willow Garage, inc and some external developers

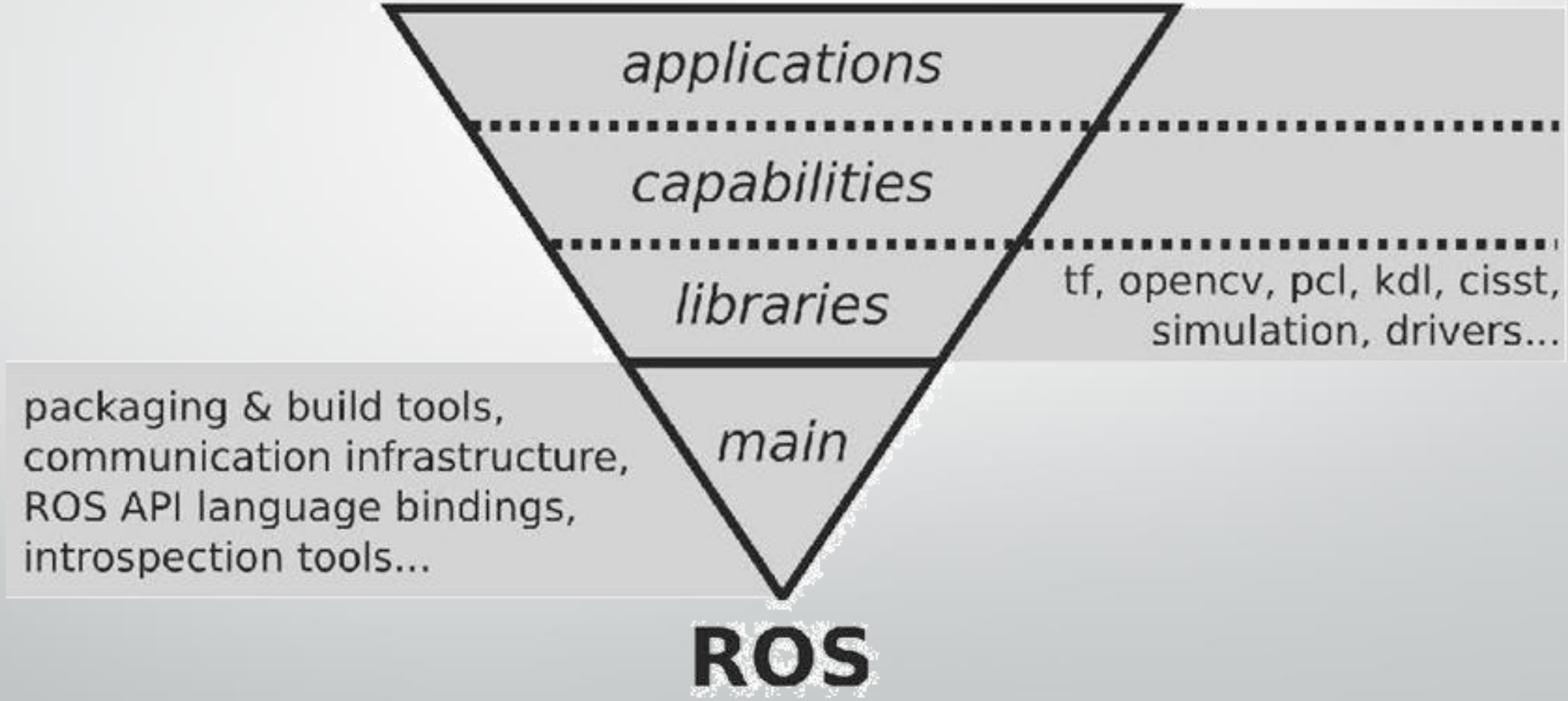
ROS

Levels of Development in ROS

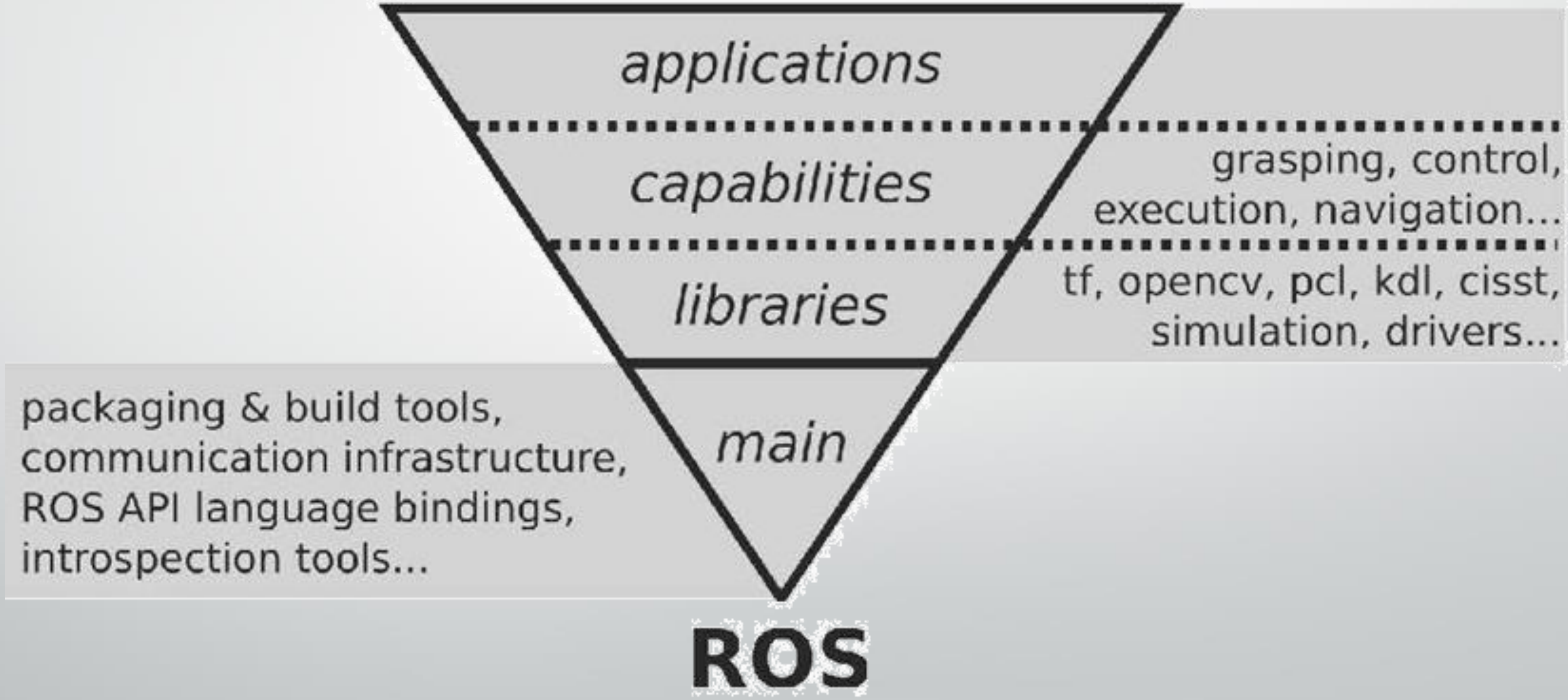


packaging & build tools,
communication infrastructure,
ROS API language bindings,
introspection tools...

Levels of Development in ROS

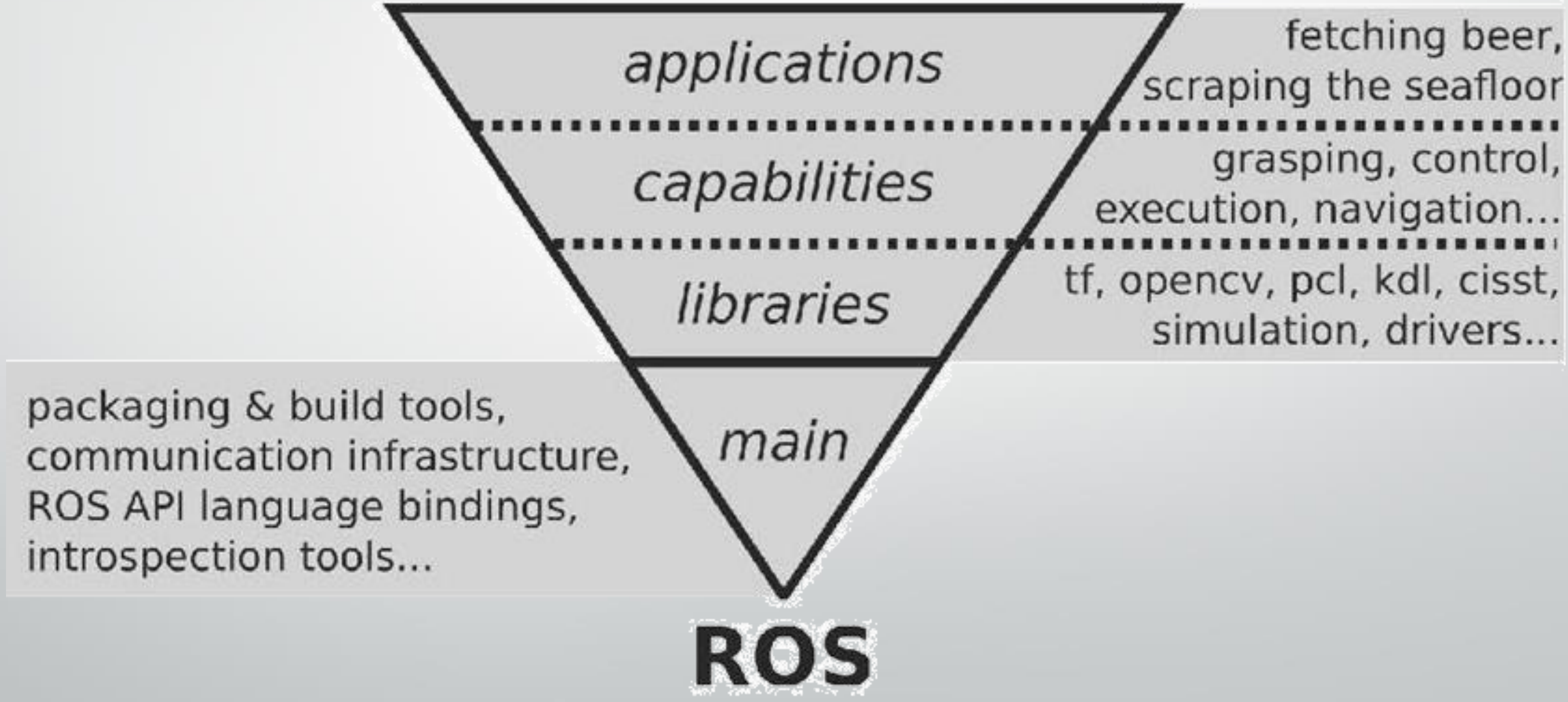


Levels of Development in ROS



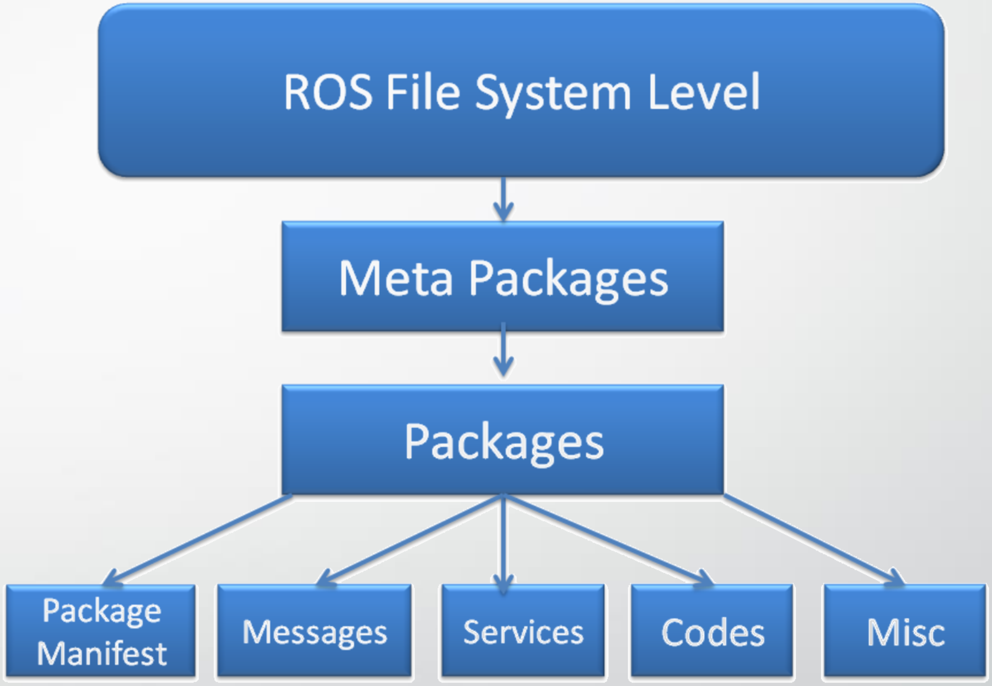
applications
capabilities
grasping, control, execution, navigation...
libraries
tf, opencv, pcl, kdl, cisst, simulation, drivers...
main
packaging & build tools, communication infrastructure, ROS API language bindings, introspection tools...
ROS

Levels of Development in ROS


applications
fetching beer, scraping the seafloor
capabilities
grasping, control, execution, navigation...
libraries
tf, opencv, pcl, kdl, cisst, simulation, drivers...
main
packaging & build tools, communication infrastructure, ROS API language bindings, introspection tools...
ROS

UNDERSTANDING THE ROS FILE SYSTEM LEVEL

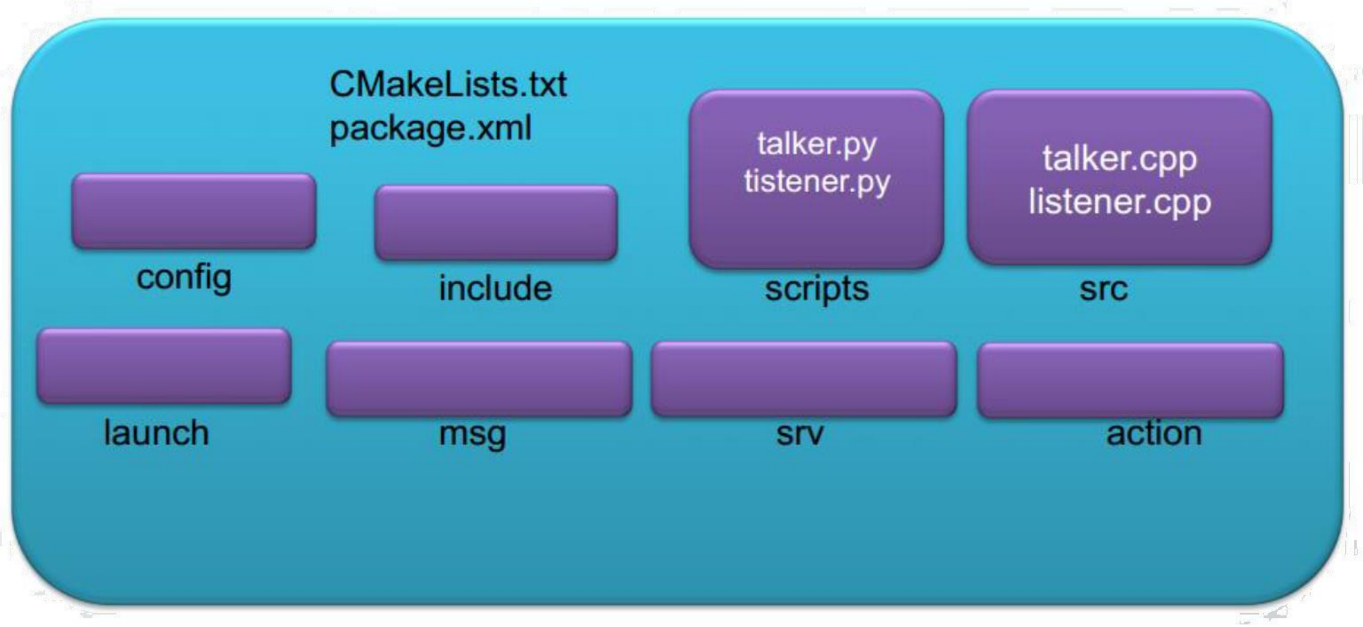
- ❑ **Packages**
- ❑ **Package manifest**
- ❑ **Meta packages**
- ❑ **Meta packages manifest**
- ❑ **Messages (.msg)**
- ❑ **Services (.srv)**
- ❑ **Repositories**



ROS File system level

ROS PACKAGES

- config
- include/package_name
- scripts
- src
- launch
- msg
- srv
- action
- package.xml
- CMakeLists.txt



Structure of a typical ROS package

ROS PACKAGES

- Some of the commands

  - ❑ catkin_create_pkg
  - ❑ rospack
  - ❑ catkin_make
  - ❑ rosdep
  - ❑ rosbash
  - ❑ rosrun
  - ❑ roscp
  - ❑ rosed
  - ❑ roscd

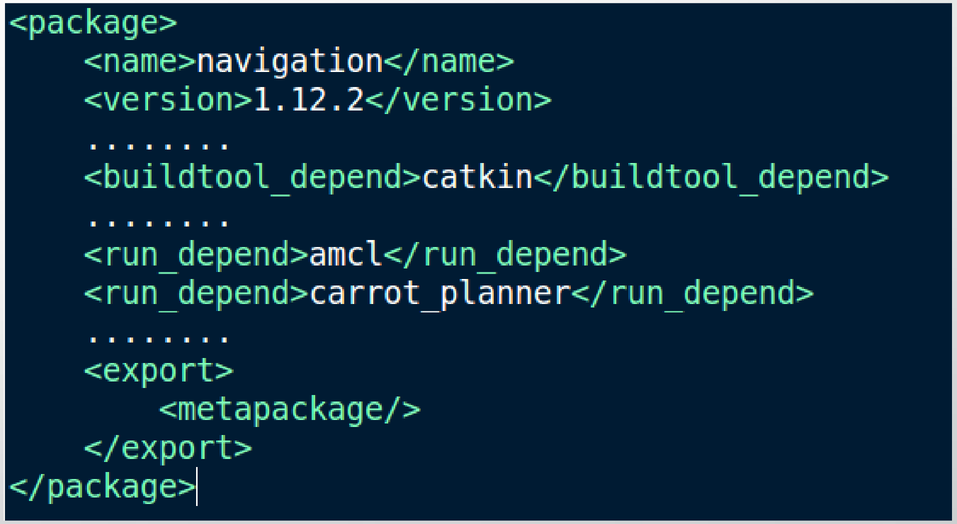
ROS META PACKAGES

- Specialized packages with just a package.xml file.
  - do not contain any tests, code, files

- Grouping a set of multiple packages
  - ROS navigation stack

- an export tag

```
<export>
    <metapackage/>
</export>
```



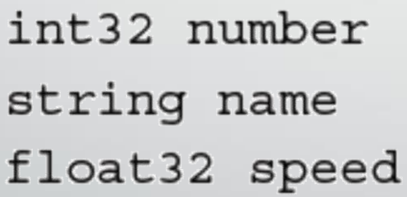
```
<package>
    <name>navigation</name>
    <version>1.12.2</version>
    ........
    <buildtool_depend>catkin</buildtool_depend>
    ........
    <run_depend>amcl</run_depend>
    <run_depend>carrot_planner</run_depend>
    ........
    <export>
        <metapackage/>
    </export>
</package>
```

ROS navigation stack

ROS MESSAGES

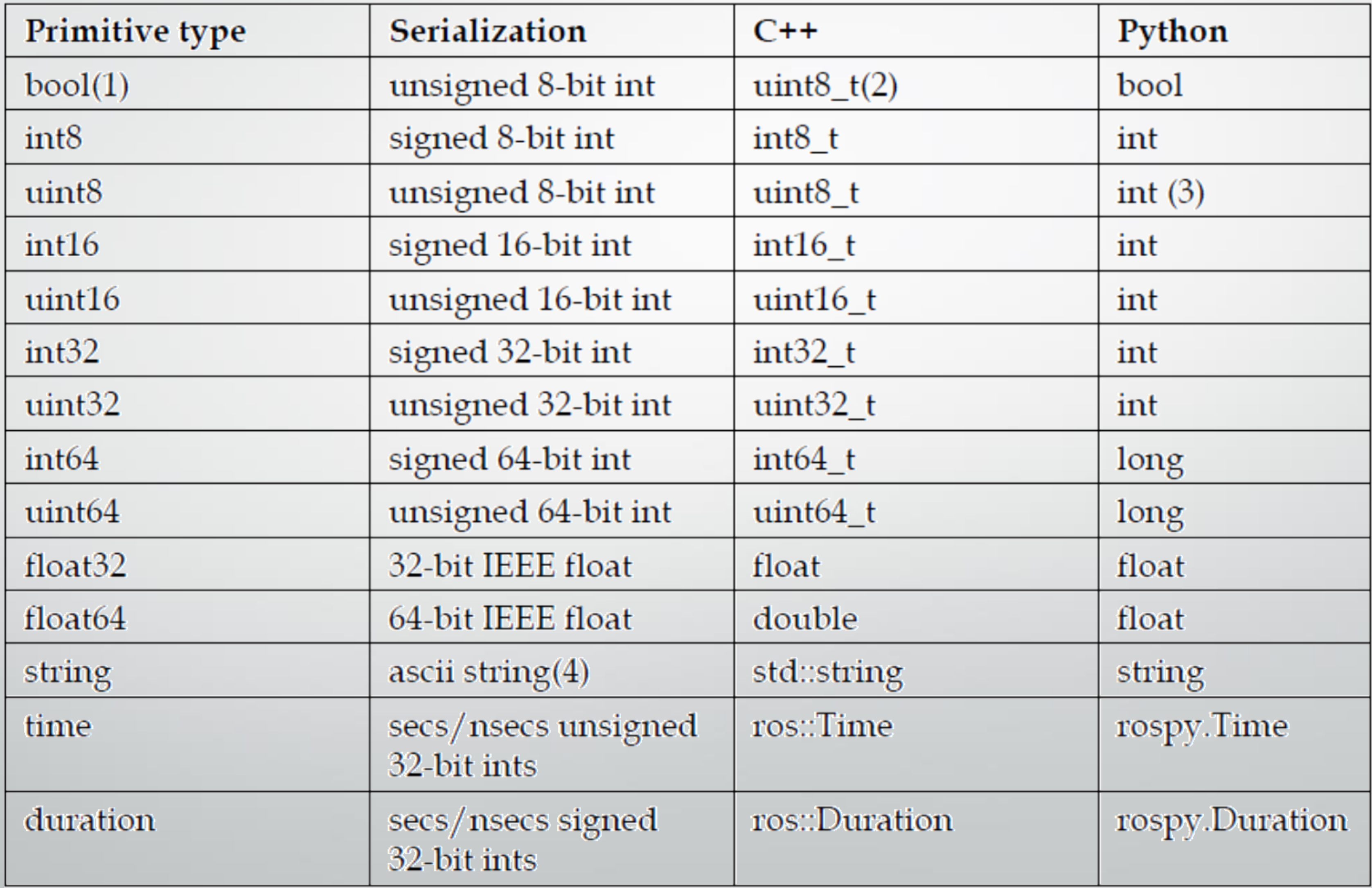
- ❑ Describing types of publishing data
  - ➢ As a list of data field descriptions and constant definitions
    - ▪ field types and field name
  - ➢ stored in .msg files

- ❑ Here is an example of message definitions:



```
int32 number
string name
float32 speed
```

THE BUILT-IN FIELD TYPES



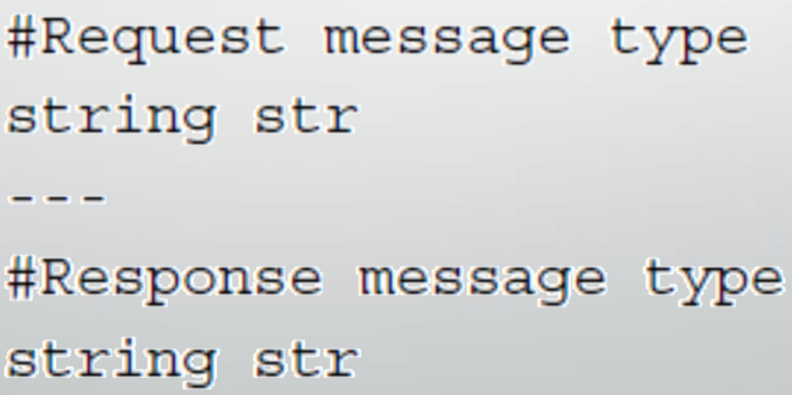
Primitive type	Serialization	C++	Python
bool(1)	unsigned 8-bit int	uint8_t(2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int (3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	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 unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

THE ROS SERVICES

- a request/response communication type between ROS nodes
  - Similar to the message definition

- definitions in a .srv file

- An example service description format



```
#Request message type
string str
---
#Response message type
string str
```

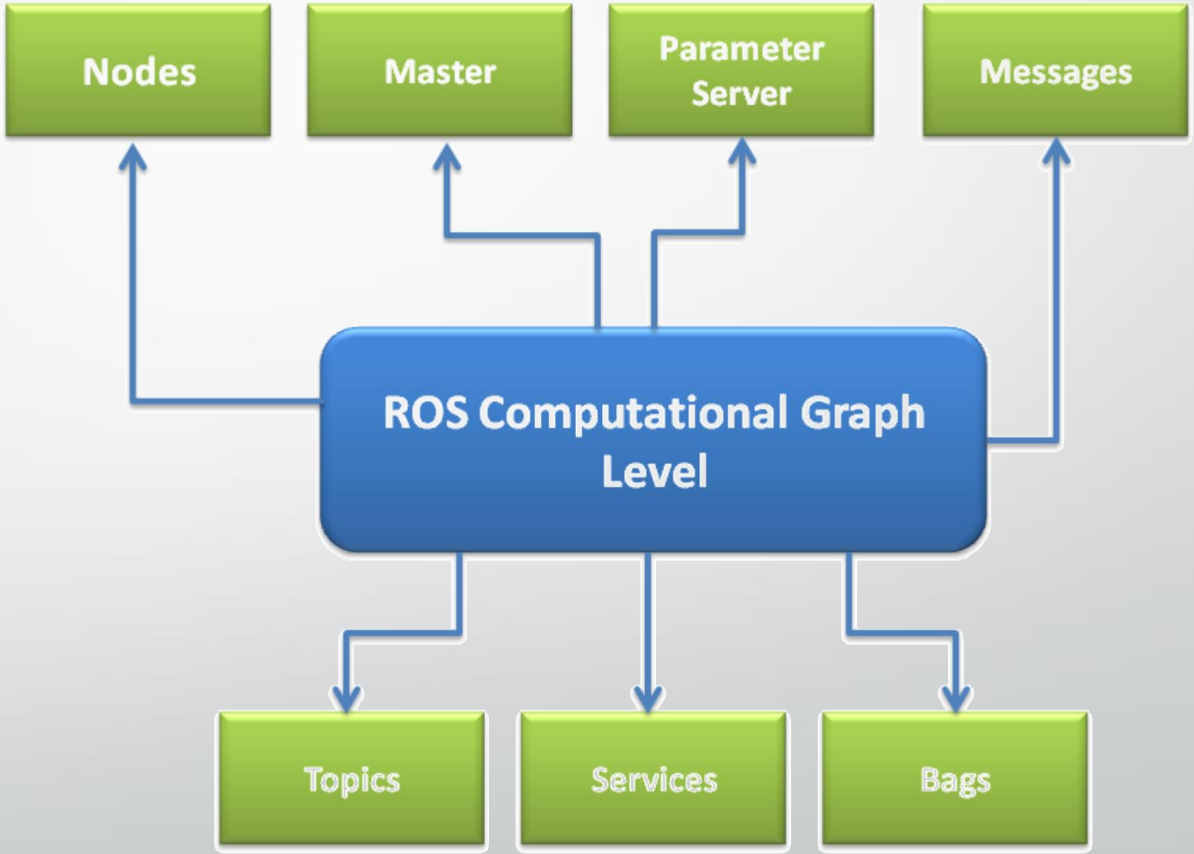
UNDERSTANDING THE ROS COMPUTATION GRAPH LEVEL

- ❑ Computation is done by using a network of process
  - ✓ Computation graph
- ❑ The main concepts of the network
  - ▪ ROS Nodes
  - ▪ Master
  - ▪ Parameter server
  - ▪ Messages,
  - ▪ Topics
  - ▪ Services
  - ▪ Bags
- ✓ ros_comm
  - ➢ http://wiki.ros.org/ros_comm

❖ ROS Graph layer

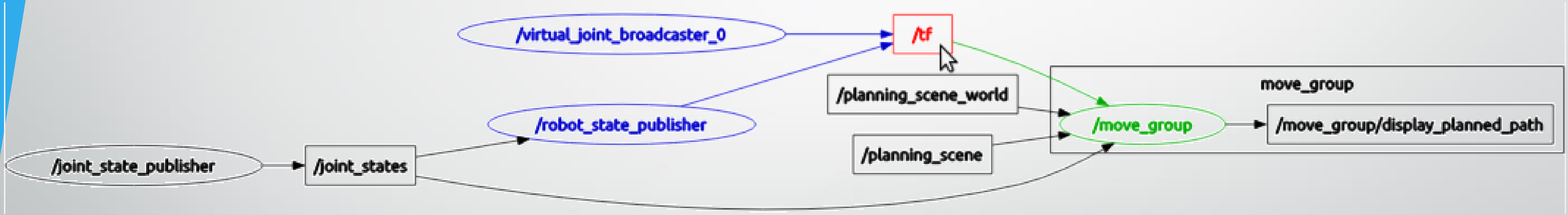
ROS GRAPH LAYER

- ❑ **Nodes:**
- ❑ **Master:**
- ❑ **Parameter Server:**
- ❑ **Messages:**
- ❑ **Topics:**
- ❑ **Services:**
- ❑ **Bags:**



ROS GRAPH LAYER

- rqt_graph (http://wiki.ros.org/rqt_graph)



Graph of communication between nodes using topics

UNDERSTANDING ROS NODES

❑ performing computation
- ➢ using ROS client libraries
  - ✓ roscpp
  - ✓ rospy

❑ Communicating by using
- ➢ ROS Topics,
- ➢ ROS Services,
- ➢ ROS Parameters

❖ Benefits:
- ✓ Fault tolerant system
- ✓ Reduce the complexity
- ✓ Increase debug-ability

❑ Rosbash
- ➢ introspect ROS nodes

❑ Rosnode

```
$ rosnode info [node_name]
$ rosnode kill [node_name]
$ rosnode list
$ rosnode machine [machine_name]
$ rosnode ping
$ rosnode cleanup
```

ROS TOPICS

❑ buses in which ROS nodes exchange messages

- ✓ Anonymously publish and subscribe
- ✓ Asynchronous many-to-many communication streams
- ✓ topics are unidirectional,
- ✓ TCP/IP-based transport (**TCPROS**)

❑ ROS topic tool

```
$ rostopic bw /topic
$ rostopic echo /topic:
$ rostopic find /message_type:
$ rostopic hz /topic:
$ rostopic info /topic:
$ rostopic list:
$ rostopic pub /topic message_type args
$ rostopic type /topic
```

❑ Request/response communications

  ❑ ROS services

ROS MESSAGES

- ❑ ROS nodes communicate with each other by publishing messages to a topic.
  - ➢ messages are a simple data structure
  - ➢ standard primitive datatypes and arrays of primitive types
  - ➢ MD5 checksum comparison
- ❑ rosmsg

```
$ rosmsg show [message]
$ rosmsg list
$ rosmsg md5 [message]
$ rosmsg package [package_name]
$ rosmsg packages [package_1] [package_2]
```

ROS SERVICES

- ❑ using a pair of messages for request/response communications
- ❑ .srv file
  - ✓ request/response datatypes
- ❑ ROS server and service client
  - ✓ Synchronous one-to-many network-based functions.
  - ✓ MD5 checksum
- ❑ Two ROS tools
  - ➢ `Rossrv` similar to `rosmsg`
  - ➢ `rosservice` tool

```
$ rosservice call /service args
$ rosservice find service_type
$ rosservice info /services
$ rosservice list
$ rosservice type /service
$ rosservice uri /service
```

ROS BAGS

- ❑ Storing ROS messages
  - ➢ The .bag extension
- ❑ Rosbag
  - ➢ data logging

```
$ rosbag record [topic_1] [topic_2] -o [bag_name]
$ rosbag play [bag_name]
```

rqt_bag

❑ a GUI tool for recording and managing bag files

- show bag message contents
- display image messages (optionally as thumbnails on a timeline)
- plot configurable time-series of message values
- publish/record messages on selected topics to/from ROS
- export messages in a time range to a new bag

UNDERSTANDING ROS MASTER

- ❑ like a DNS server

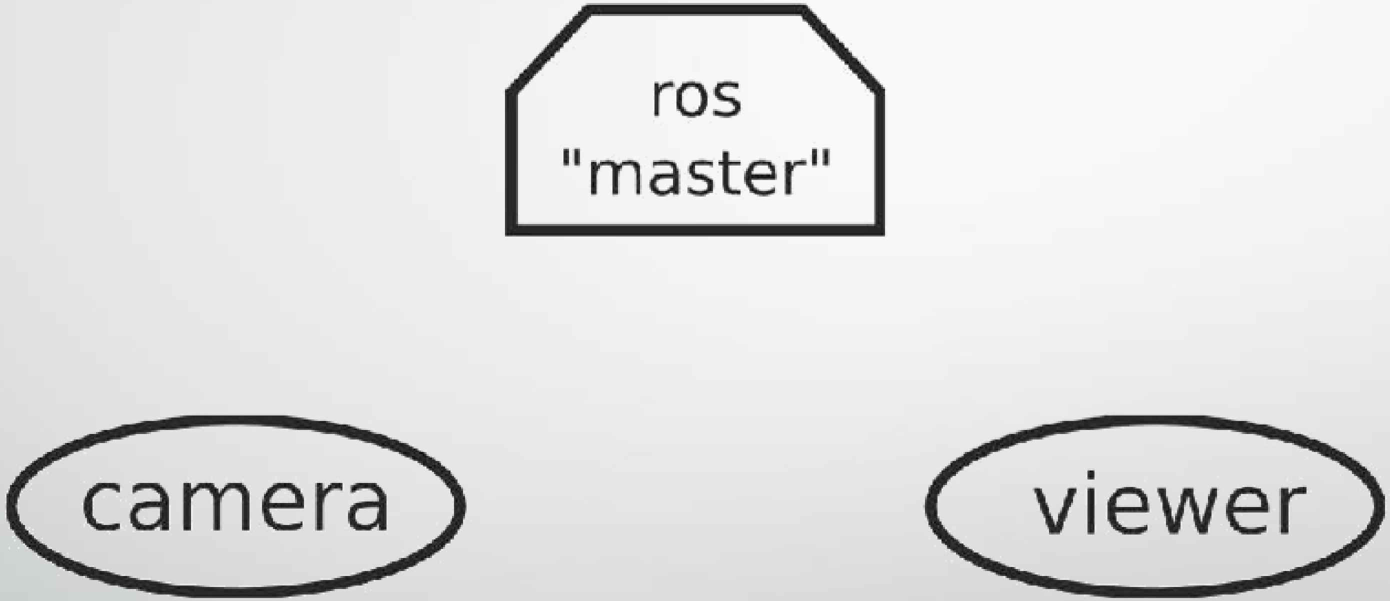
  the details of all nodes currently running

- ❑ A centralized XML-RPC(**Remote Procedure Call)** server
  - ✓ Negotiates communication connections
  - ✓ Registers and looks up names for ROS graph resources

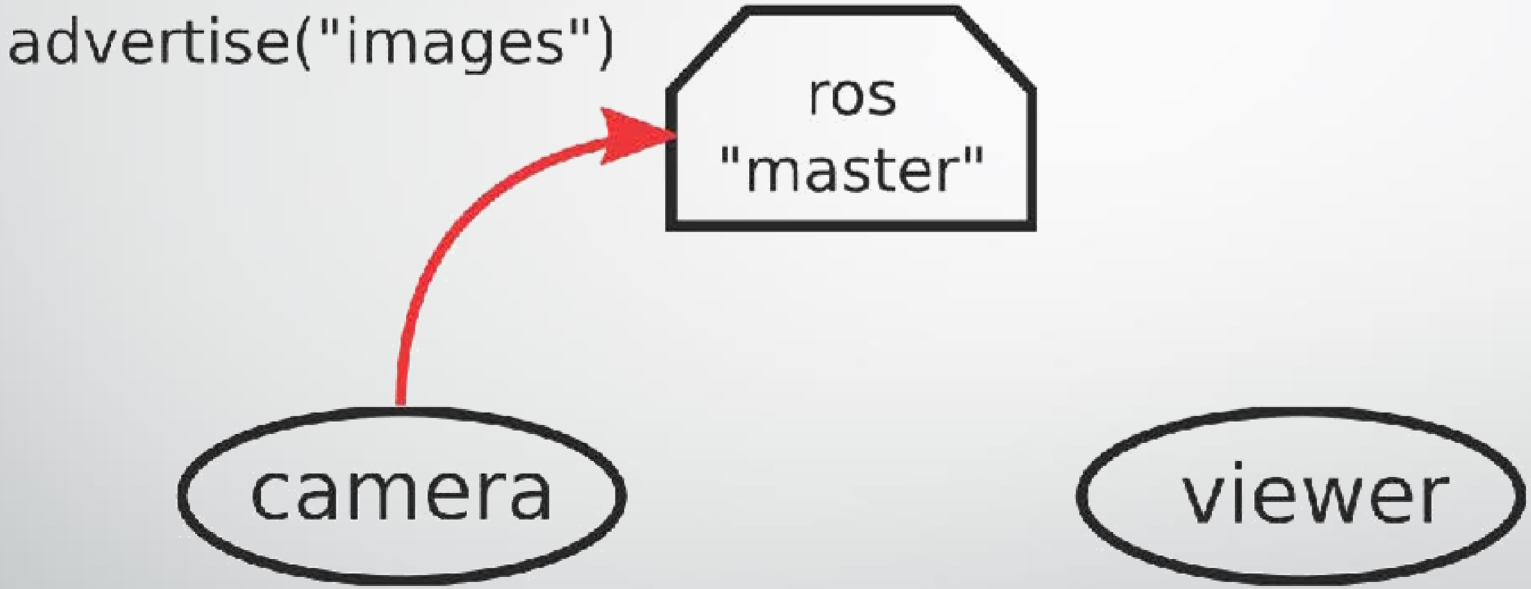
❑ single system localhost

❑ distributed network only one Master `ROS_MASTER_URI`

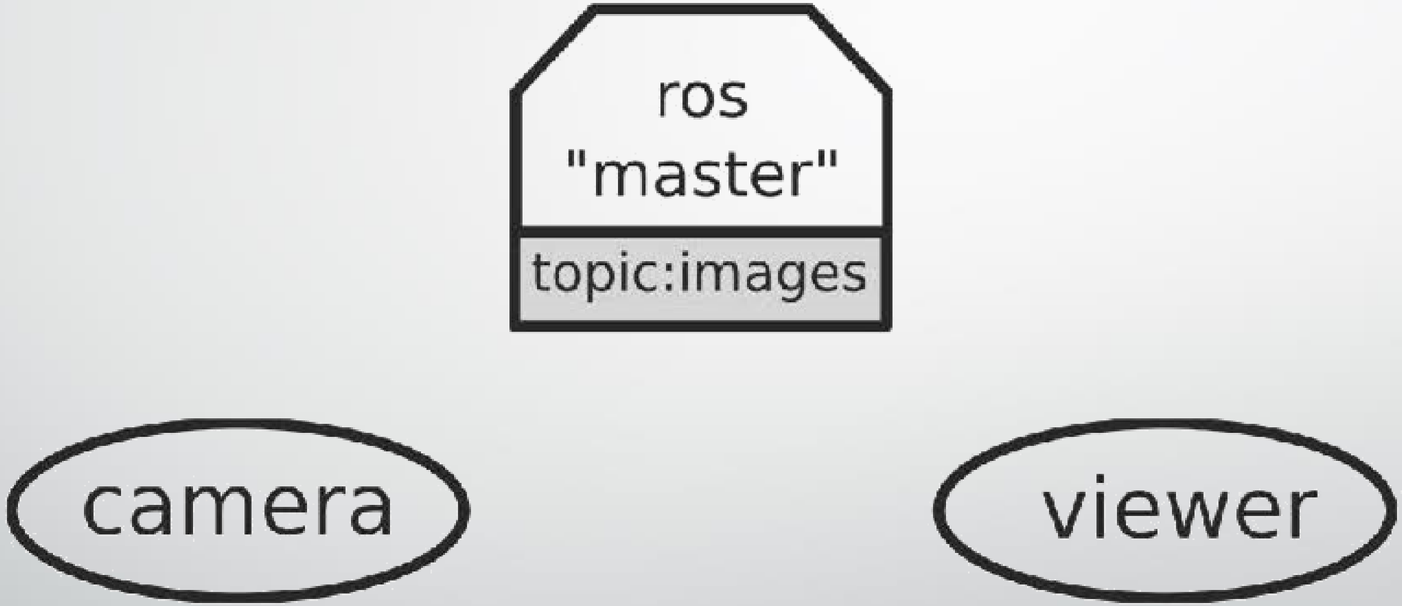
UNDERSTANDING ROS MASTER



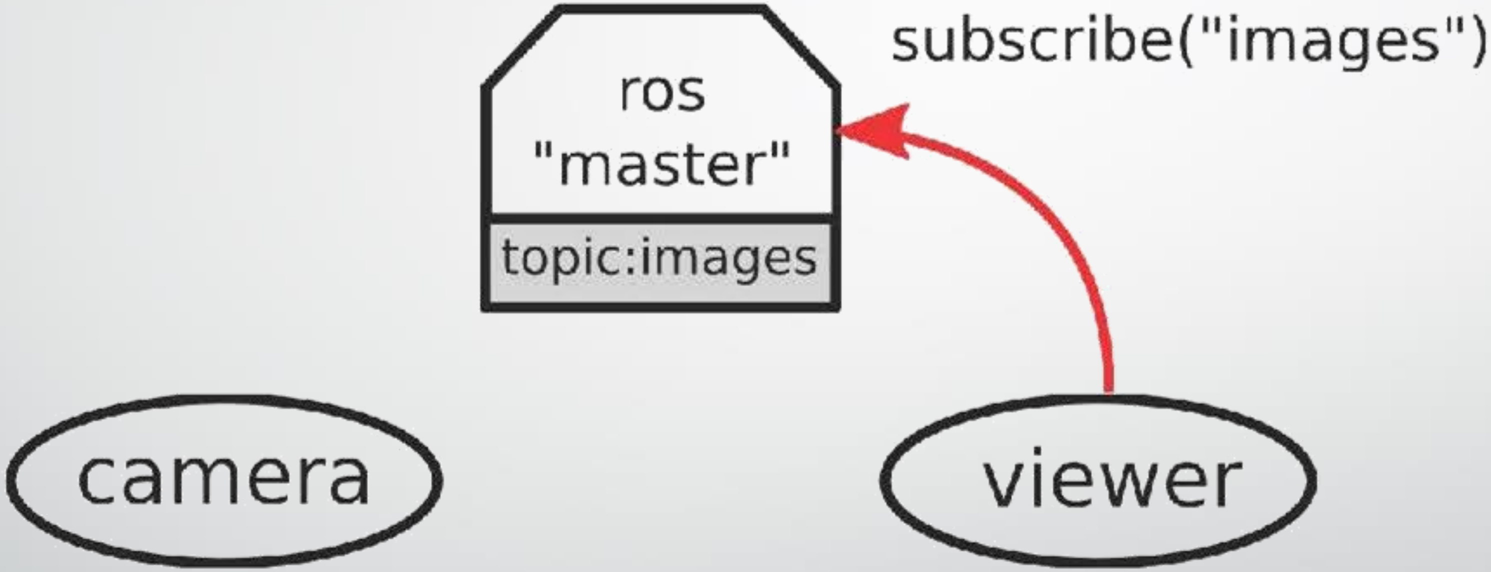
UNDERSTANDING ROS MASTER



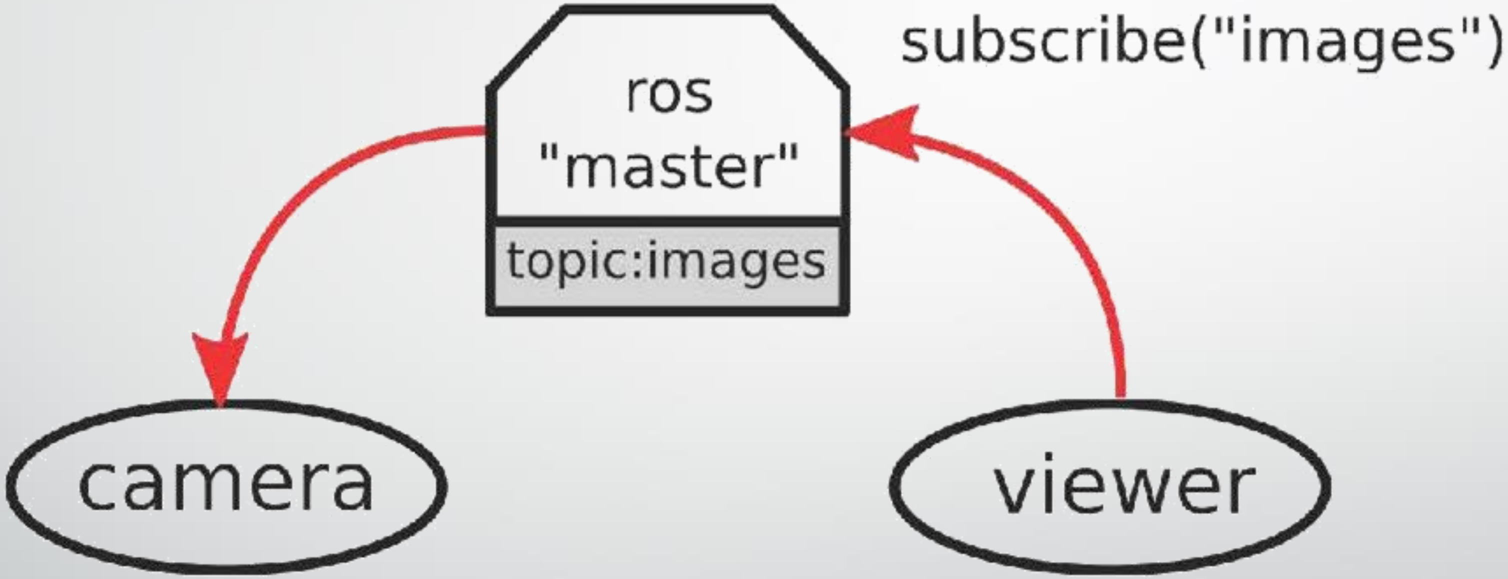
UNDERSTANDING ROS MASTER



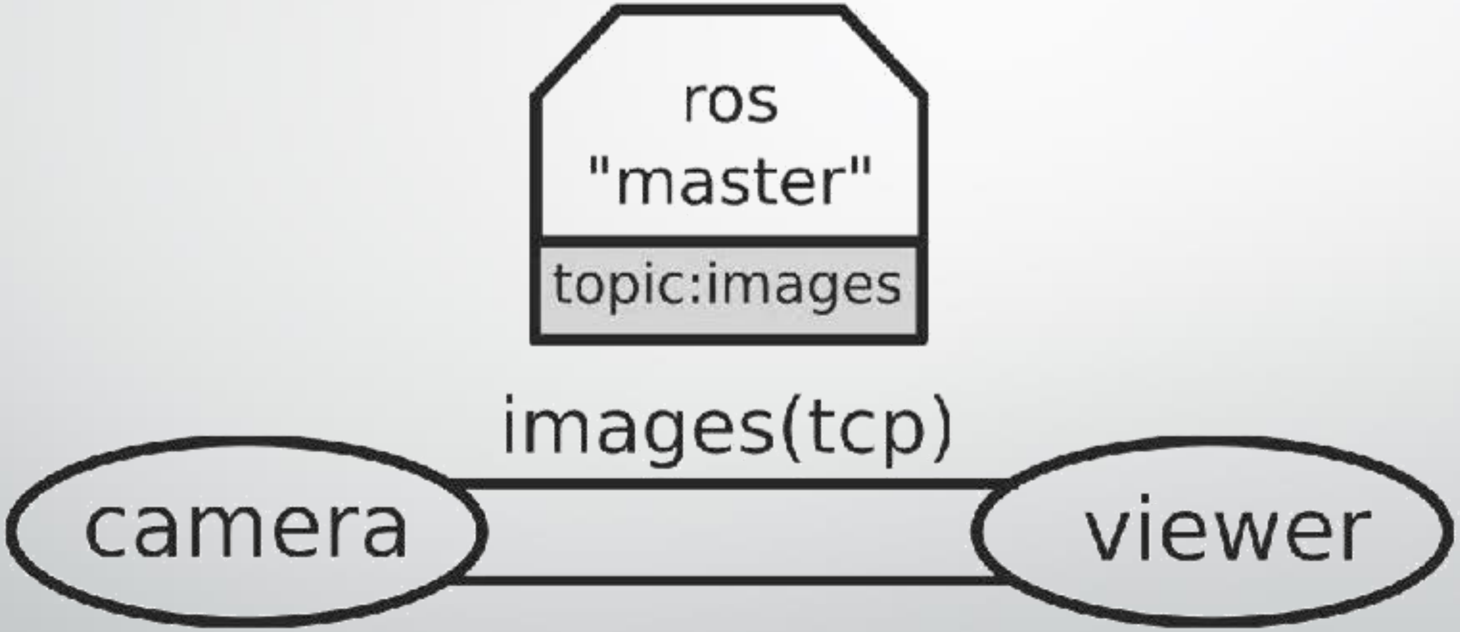
UNDERSTANDING ROS MASTER



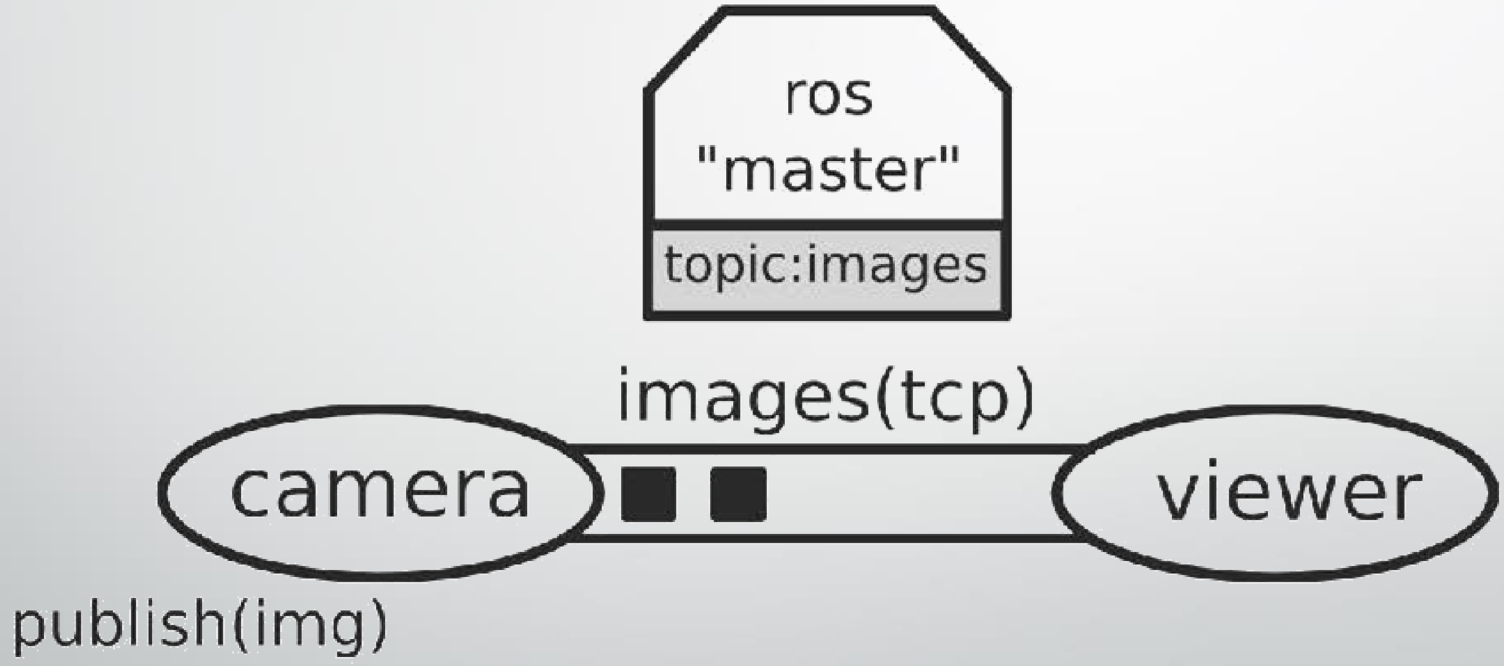
UNDERSTANDING ROS MASTER



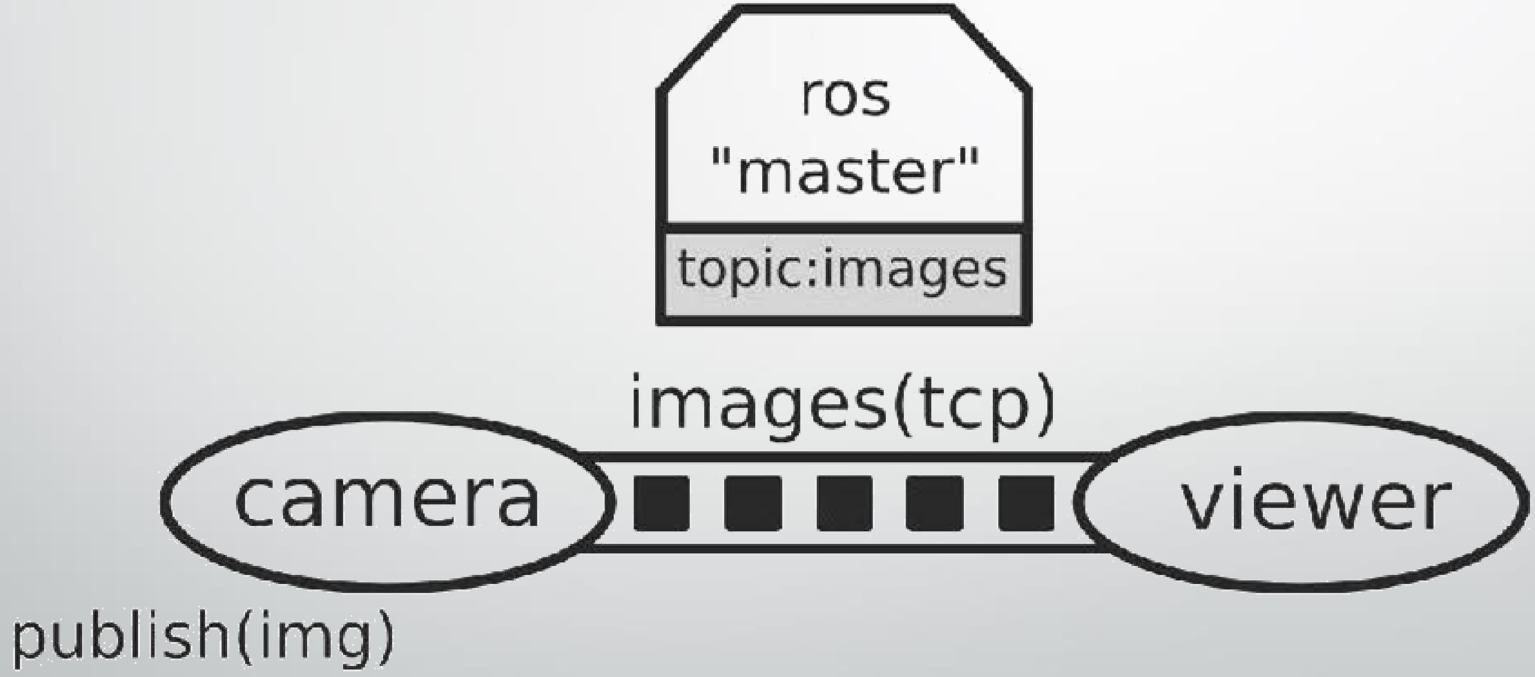
UNDERSTANDING ROS MASTER



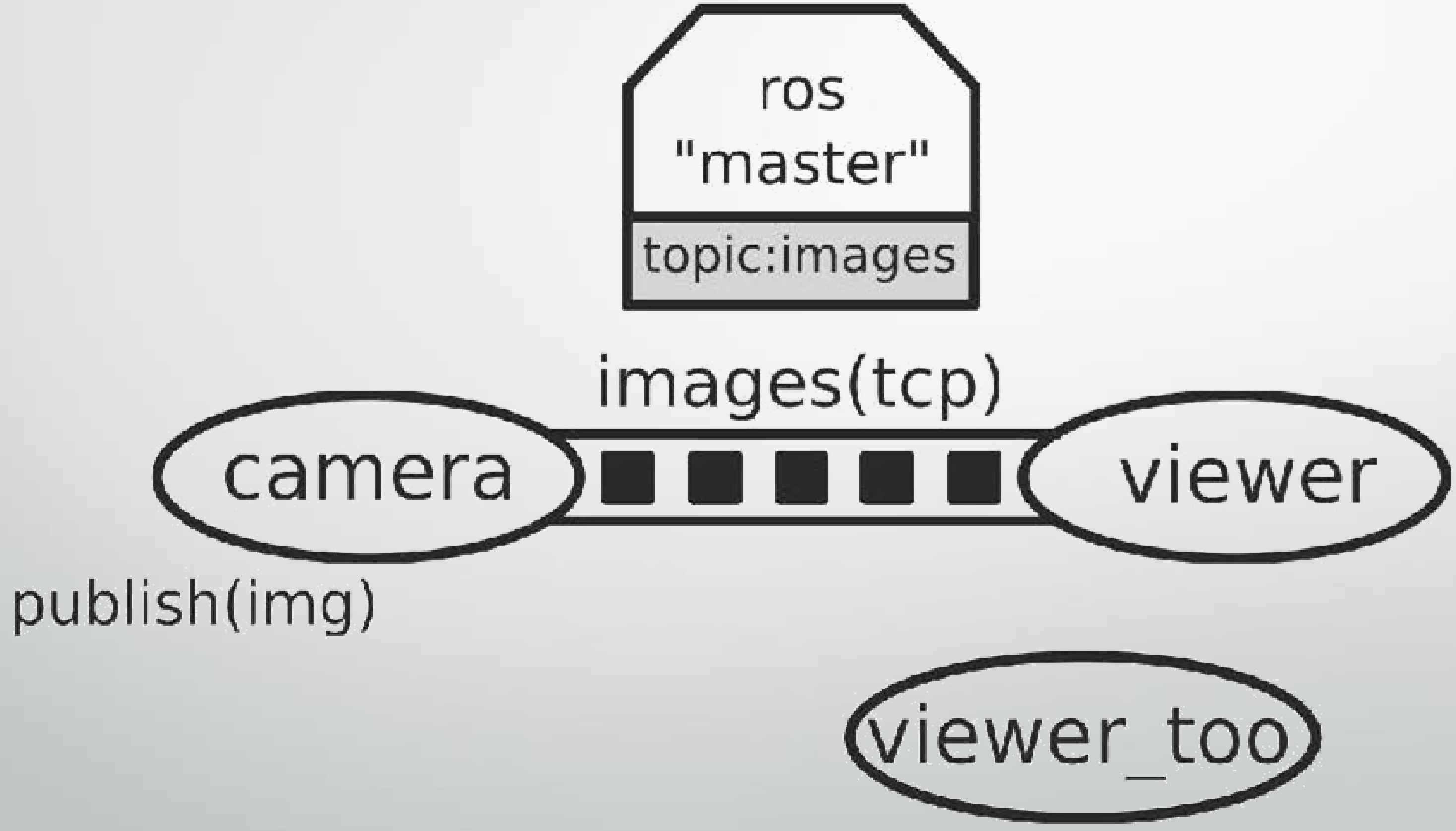
UNDERSTANDING ROS MASTER



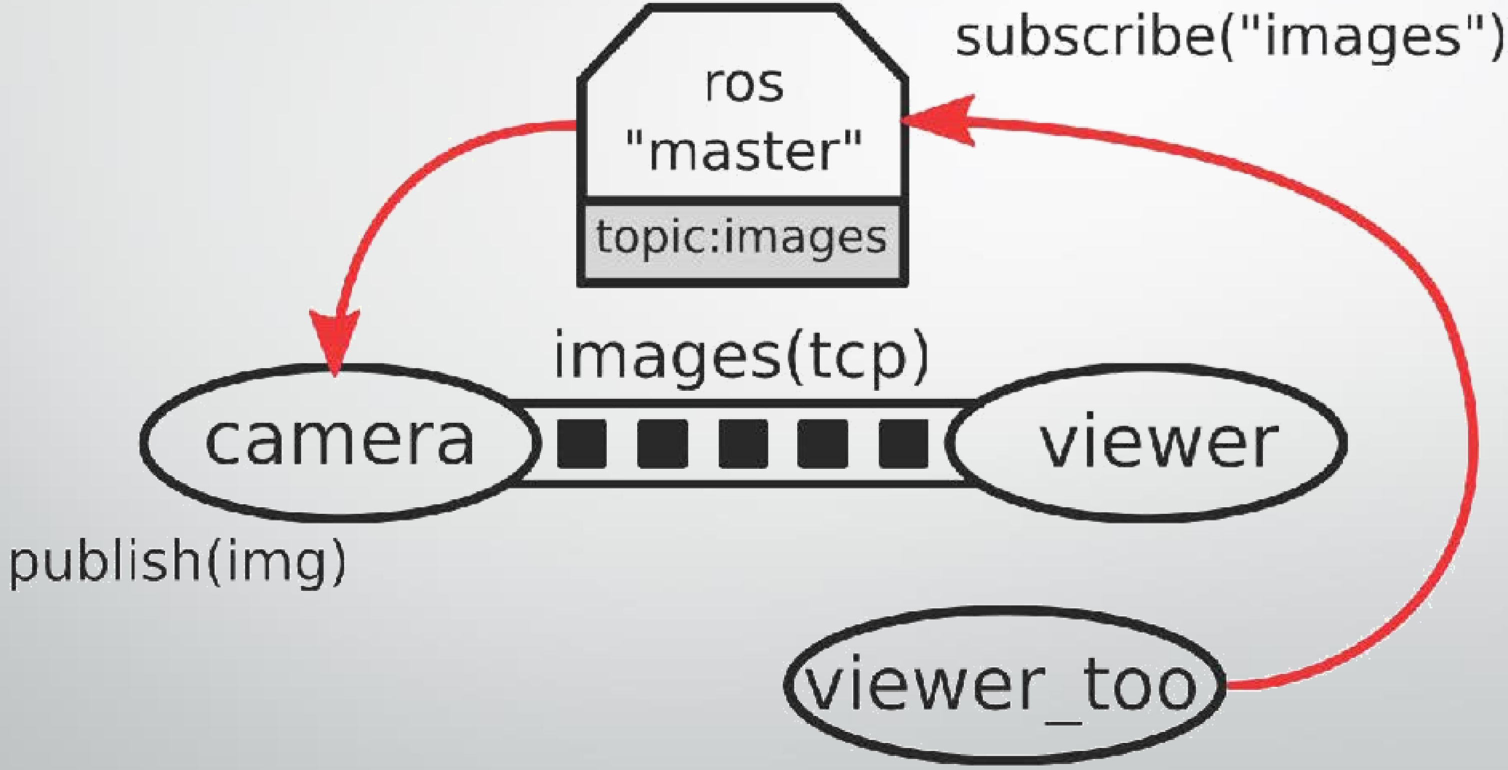
UNDERSTANDING ROS MASTER



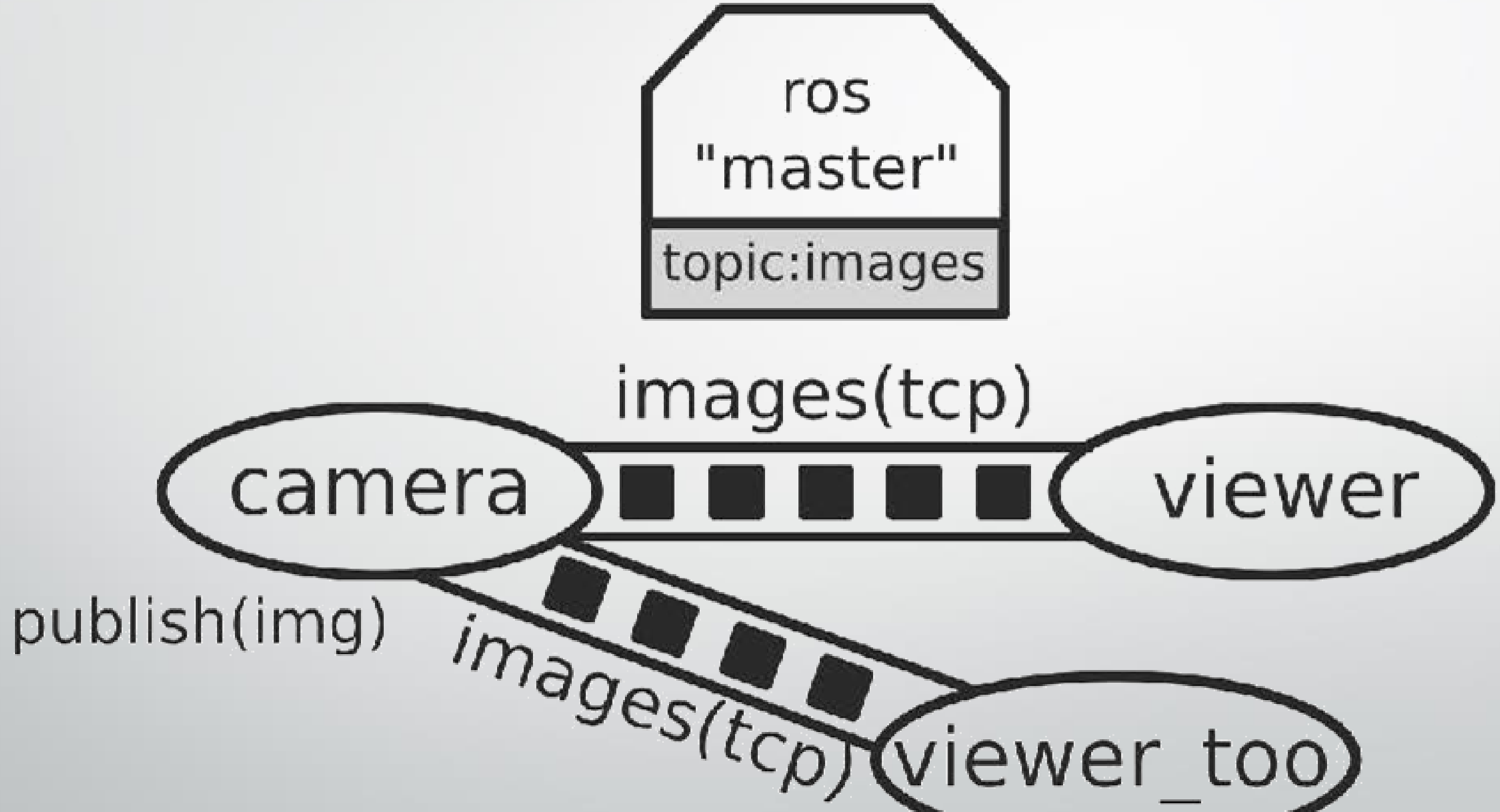
UNDERSTANDING ROS MASTER



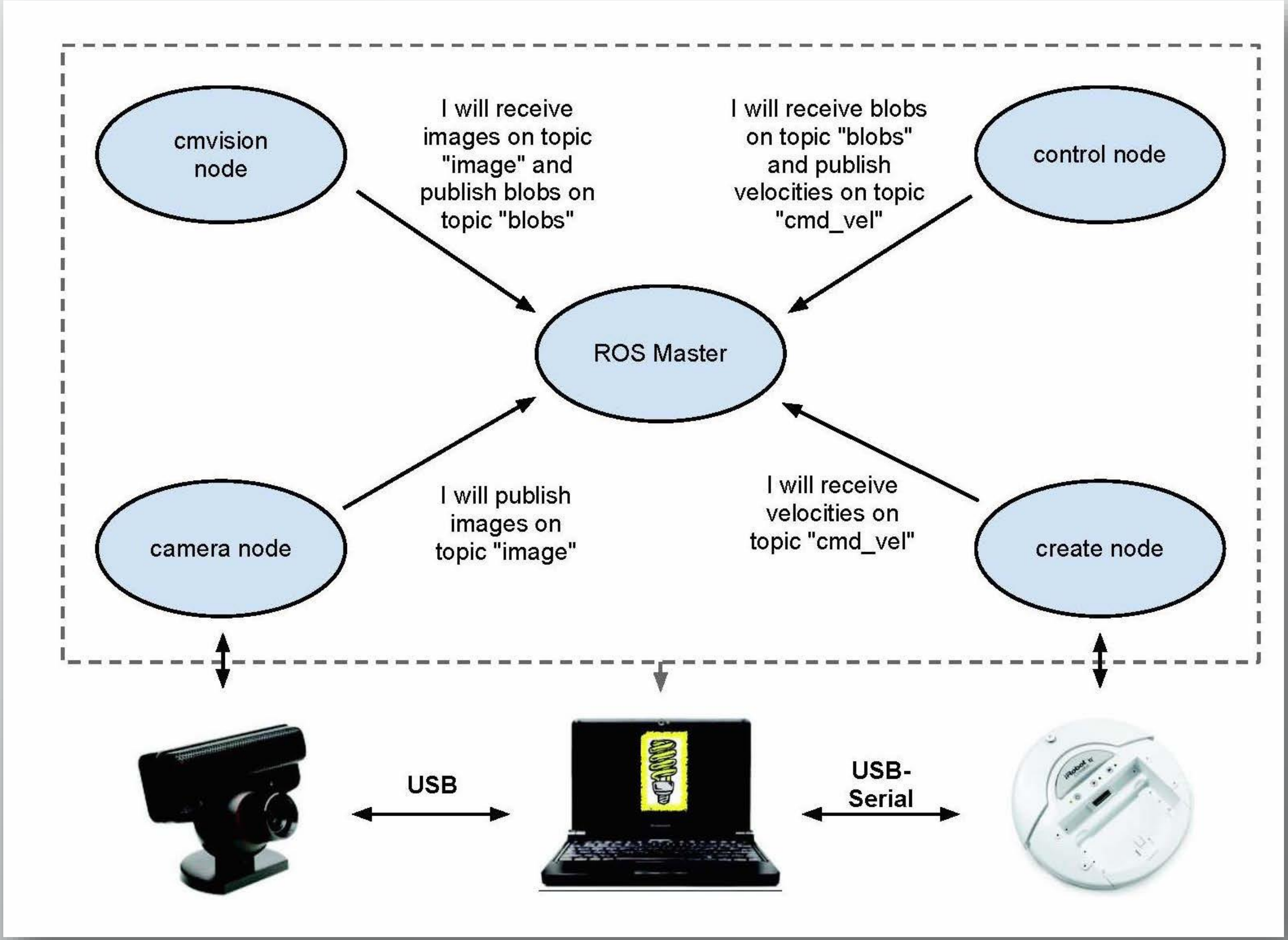
UNDERSTANDING ROS MASTER



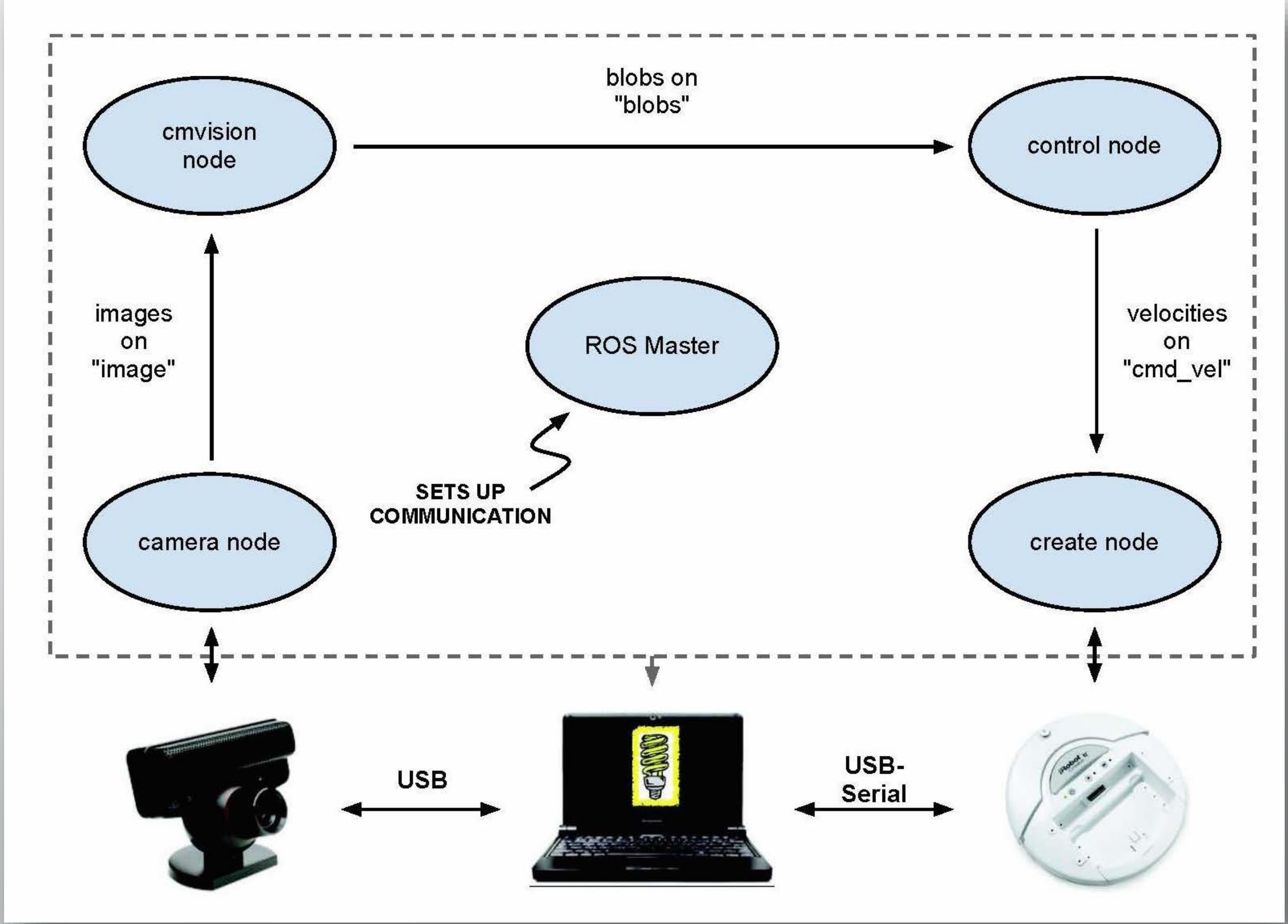
UNDERSTANDING ROS MASTER



UNDERSTANDING ROS MASTER


cmvision node
I will receive images on topic "image" and publish blobs on topic "blobs"
I will receive blobs on topic "blobs" and publish velocities on topic "cmd_vel"
control node
ROS Master
camera node
I will publish images on topic "image"
I will receive velocities on topic "cmd_vel"
create node
USB
USB-Serial

UNDERSTANDING ROS MASTER



INTRODUCTION TO ROS

ROS-PART 2

Amirreza Kabiri & Fatemeh Pahlevan @ Autumn 2017

USING THE ROS PARAMETER

- ❑ A high number of parameters
  - ➢ store it as files

- ❑ share between two or more programs too.
  - ➢ a parameter server

- ❑ The parameter server supports the following XMLRPC datatypes:
  - ➢ 32-bit integers
  - ➢ Booleans
  - ➢ strings
  - ➢ doubles
  - ➢ iso8601 dates
  - ➢ lists
  - ➢ base64-encoded binary data

USING THE ROS PARAMETER

❑ YAML file

```
/camera/name : 'nikon' #string type
/camera/fps : 30 #integer
/camera/exposure : 1.2 #float
/camera/active : true #boolean
```

❑ rosparam

```
$ rosparam set [parameter_name] [value]
$ rosparam get [parameter_name]
$ rosparam load [YAML file]
$ rosparam dump [YAML file]
$ rosparam delete [parameter_name]
$ rosparam list:
```

❑ dyamic_reconfigure http://wiki.ros.org/dynamic_reconfigure

RUNNING ROS MASTER AND ROS PARAMETER SERVER

- ❑ start ROS Master and the ROS parameter Server
  - ○ `Roscore`
- ❑ a prerequisite before running any ROS node
  - ✓ ROS Master
  - ✓ ROS parameter server
  - ✓ rosout logging nodes

- ❑ Rosout node and topic
  - ○ `/rosout_agg`
    - ➢ aggregate stream of log messages

RUNNING ROS MASTER AND ROS PARAMETER SERVER

- ○ `$ roscore`

❑ A log file is creating inside the ~/.ros/log used for debugging purposes

❑ A ROS launch file called `roscore.xml`

- ➢ Automatically starts the `rosmaster` and ROS parameter server.

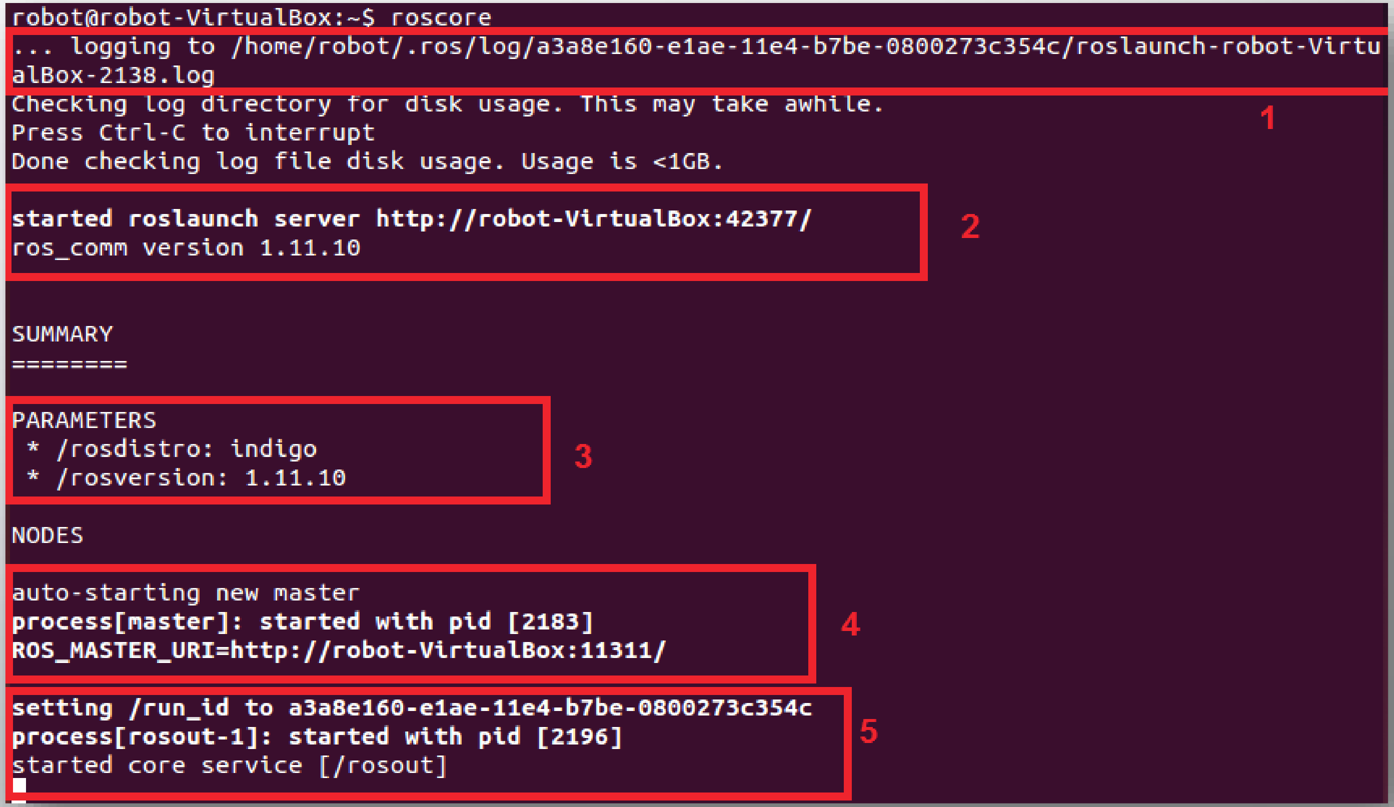
❑ Parameters :

- ➢ `rosdistro`
- ➢ `rosversion`

❑ the rosmaster node is started using

- ○ `ROS_MASTER_URI`

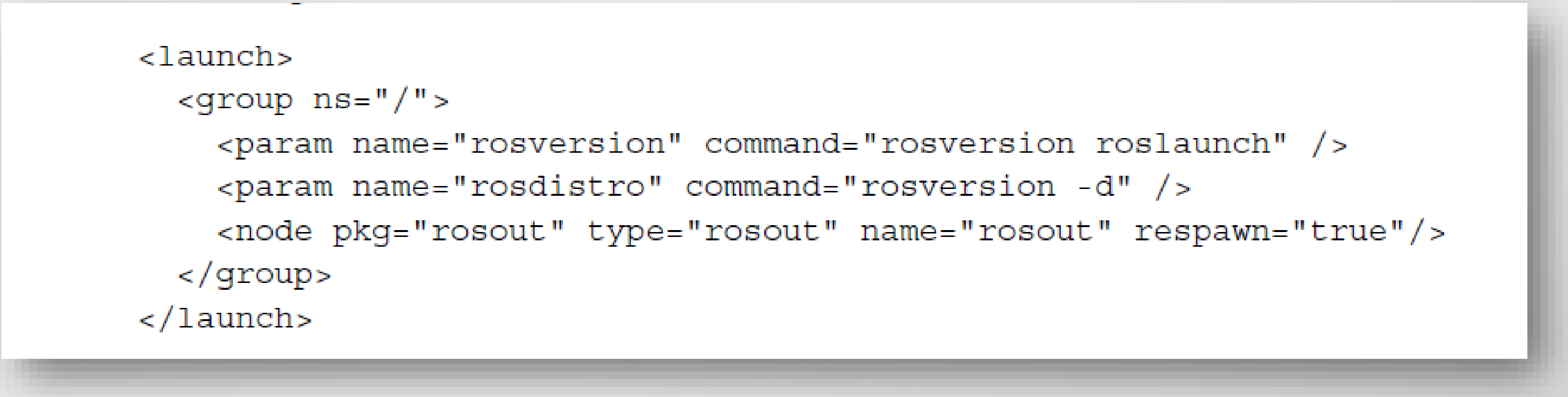
❑ The rosout node is started



```
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]
```

RUNNING ROS MASTER AND ROS PARAMETER SERVER

- ❑ roscore.xml



```
<launch>
  <group ns="/">
    <param name="rosversion" command="rosversion roslaunch" />
    <param name="rosdistro" command="rosversion -d" />
    <node pkg="rosout" type="rosout" name="rosout" respawn="true"/>
  </group>
</launch>
```

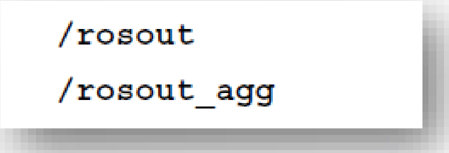
- ➢ rosversion roslaunch and rosversion -d commands

RUNNING ROS MASTER AND ROS PARAMETER SERVER

❑ CHECKING THE ROSCORE COMMAND OUTPUT

```
$ rostopic list
```

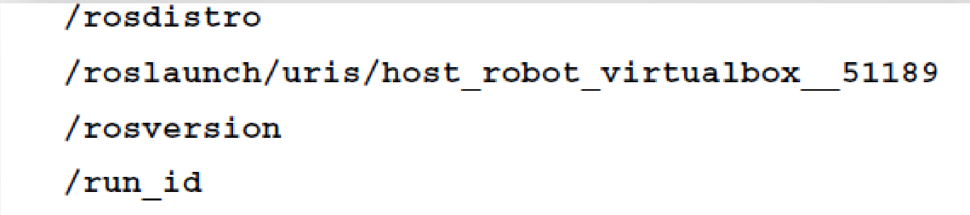
✓ lists the active topics



```
/rosout
/rosout_agg
```

```
$ rosparam list
```

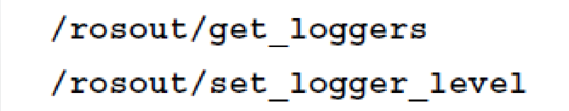
✓ lists the available parameters



```
/rosdistro
/roslaunch/uris/host_robot_virtualbox__51189
/rosversion
/run_id
```

```
$ rosservice list
```

✓ lists the running services



```
/rosout/get_loggers
/rosout/set_logger_level
```

What makes up a catkin package?

- ❑ For a package to be considered a catkin package it must meet a few requirements:
  - ❑ The package must contain a catkin compliant package.xml file.
    - ➢ That package.xml file provides meta information about the package.
  - ❑ The package must contain a CMakeLists.txt which uses catkin.
    - ➢ If it is a catkin metapackage it must have the relevant boilerplate CMakeLists.txt file.
  - ❑ Each package must have its own folder
    - ➢ This means no nested packages nor multiple packages sharing the same directory.
  - ❑ The simplest possible package might have a structure which looks like this:
    - ➢ my_package/ CMakeLists.txt package.xml

Packages in a catkin Workspace

- The recommended method of working with catkin packages is using a catkin workspace, but you can also build catkin packages standalone. A trivial workspace might look like this:

```
workspace_folder/ -- WORKSPACE
src/ -- SOURCE SPACE
CMakeLists.txt -- 'Toplevel' CMake file, provided by catkin
package_1/
        CMakeLists.txt -- CMakeLists.txt file for package_1
        package.xml -- Package manifest for package_1
        ...
package_n/
        CMakeLists.txt -- CMakeLists.txt file for package_n
        package.xml -- Package manifest for package_n
```

CREATING A ROS PACKAGE

- ❑ The basic unit of the ROS system

- ❑ Using the catkin build system which is based on **CMake** (**Cross Platform Make**) to build ROS packages
  - ➢ responsible for generating `'targets'`(**executable/libraries**) from a raw source code
    - ✓ porting the package into other operating system
- ➢ rosbuild In older distributions

❖ CREATE A ROS CATKIN WORKSPACE

- ❑ The procedure to build a catkin workspace

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

(Initialize a new catkin workspace, build the workspace even if there are no packages)

```
$ cd ~/catkin_ws
$ catkin_make
```

(command will build the workspace)

CREATING A ROS PACKAGE

- After building the empty workspace

- Overlaying the workspace (set the environment of the current workspace to be visible by the ROS system.)

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

  - Source a bash script called `setup.bash`

- `catkin_create_pkg` is used to create a ROS package.

  - `catkin_create_pkg [package_name] [dependency1] [dependency2]`

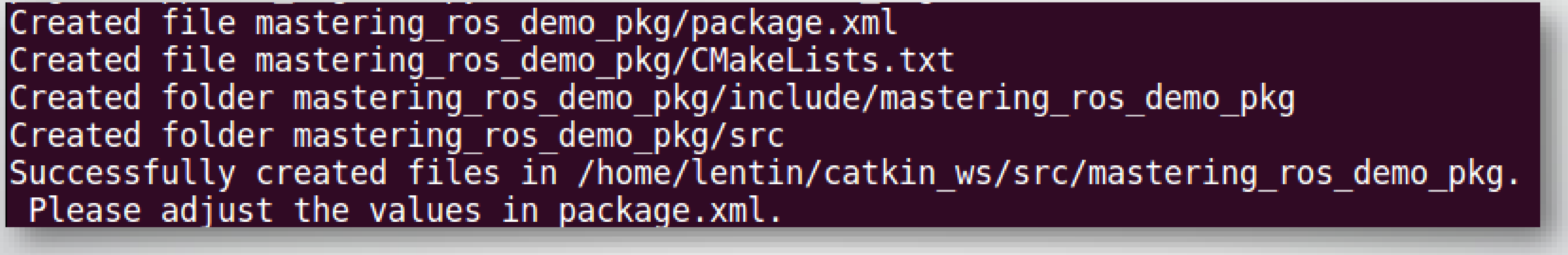
```
$ catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs
```

- Dependencies

```
roscpp
std_msgs
actionlib
actionlib_msgs
```

CREATING A ROS PACKAGE

```
$ catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs
```



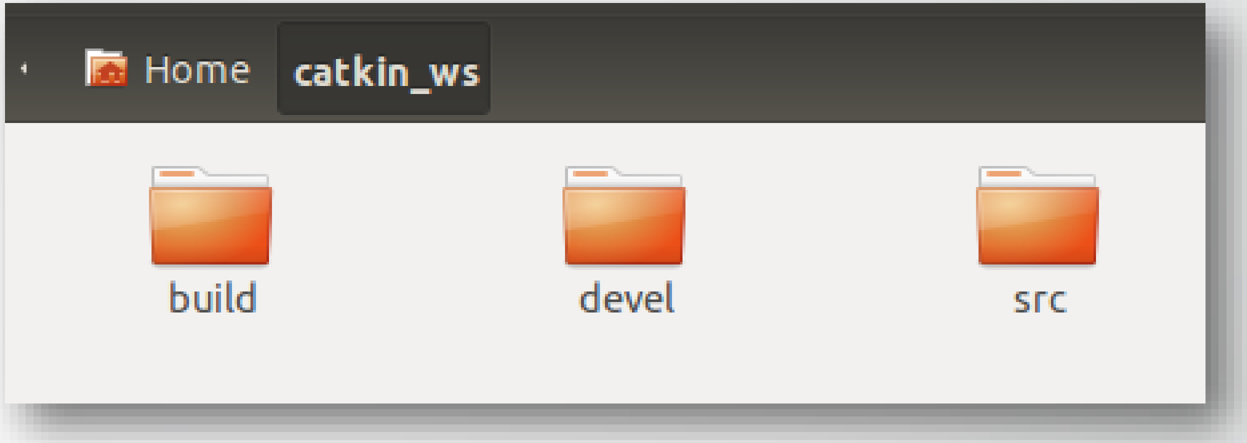
```
Created file mastering_ros_demo_pkg/package.xml
Created file mastering_ros_demo_pkg/CMakeLists.txt
Created folder mastering_ros_demo_pkg/include/mastering_ros_demo_pkg
Created folder mastering_ros_demo_pkg/src
Successfully created files in /home/lentin/catkin_ws/src/mastering_ros_demo_pkg.
 Please adjust the values in package.xml.
```

- Terminal messages while creating a ROS package

□ build the package by the `catkin_make` command must be executed from the catkin workspace path

➢ start adding nodes to the `src` folder

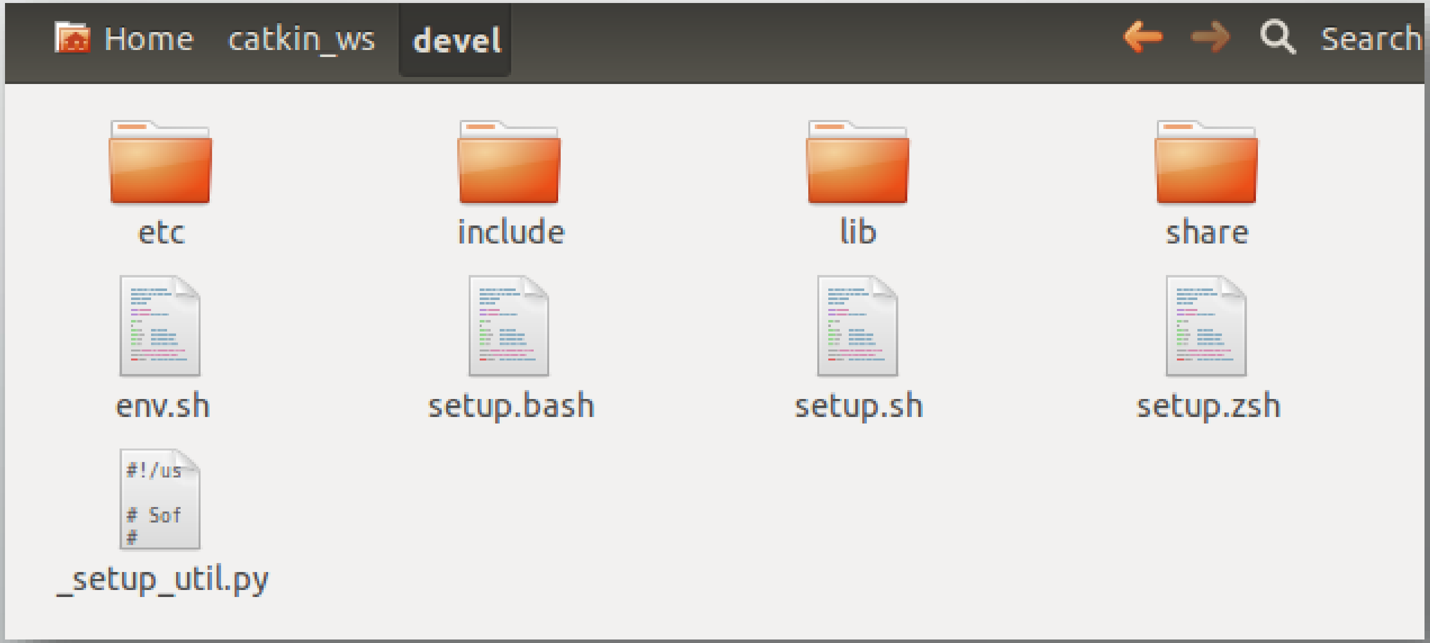
CREATING A ROS PACKAGE



Home catkin_ws

build devel src

- ❑ The `build` folder
  - ➢ executables of the nodes
- ❑ The `devel` folder
  - ➢ bash script, header files, and other executables



Home catkin_ws devel

etc include lib share

env.sh setup.bash setup.sh setup.zsh

_setup_util.py

ROS META-FILESYSTEM

- ❑ Increasing codebase flexibility

- ❑ The minimal representation of a ros package is a directory in the `$ROS_PACKAGE_PATH` which contains

  - ❑ `manifest.xml`
    - ✓ Contains package metadata (author, license, url, etc)
    - ✓ Specifies system and package dependencies
    - ✓ Specifies language-specific export flags

- ❑ `CMakeLists.txt`
  - ✓ Contains ROS build rules (executables, libraries, costum build flags, etc)

  - ❑ `Makefile`
    - ✓ Just a proxy to build this package

CMAKELISTS.TXT

http://wiki.ros.org/catkin/CMakeLists.txt

- ❑ The `CMakeLists.txt` file in the package to compile and build the source code
  - ❑ Required CMake Version (cmake_minimum_required)
    - ➢ `cmake_minimum_required(VERSION 2.8.3)`
  - ❑ Package Name (project())
    - ➢ `project(mastering_ros_demo_pkg)`
  - ❑ Find other CMake/Catkin packages needed for build (find_package())
    - ➢ find_package(catkin REQUIRED COMPONENTS …)

      ```
      find_package(catkin REQUIRED COMPONENTS
            roscpp
            rospy
            std_msgs
            actionlib
            actionlib_msgs
            message_generation
          )
      ```
    - ➢ `find_package(Boost REQUIRED COMPONENTS system)`

CMAKELISTS.TXT

- ❑ Enable Python module support before the call to generate_messages() and catkin_package()
  - ➢ (catkin_python_setup())

- ❑ Specify package build info export before declaring any targets with add_library() or add_executable()
  - ➢ (catkin_package())

    `INCLUDE_DIRS` - The exported include paths (i.e. cflags) for the package

    `LIBRARIES` - The exported libraries from the project

    `CATKIN_DEPENDS` - Other catkin projects that this project depends on

    `DEPENDS` - Non-catkin CMake projects that this project depends on.

    `CFG_EXTRAS` - Additional configuration options

➢
```
catkin_package(CATKIN_DEPENDS roscpp rospy std_msgs actionlib actionlib_msgs
                message_runtime)
```

CMAKELISTS.TXT

❑ Message/Service/Action Generators

- ➢ (add_message_files(), add_service_files(), add_action_files())
  - ✓ Generates programming language-specific files so that one can utilize messages, services, and actions
  - ✓ These macros must come BEFORE the `catkin_package()` macro in order for generation to work correctly.
  - ✓ Your `catkin_package()` macro must have a CATKIN_DEPENDS dependency on message_runtime.
  - ✓ You must use `find_package()` for the package message_generation, either alone or as a component of catkin.

```
add_message_files(
   FILES
   demo_msg.msg
 )
 add_service_files(
   FILES
   demo_srv.srv
 )
  add_action_files(
   FILES
   Demo_action.action
 )
```

CMAKELISTS.TXT

- ❑ Specifying Build Targets with unique names
  - ➢ Executable Target - programs we can run
  - ➢ Library Target - libraries that can be used by executable targets at build and/or runtime
- ❑ Specify where resources can be found for said targets
  - ➢ Include Paths - Where can header files be found for the code (most common in C/C++) being built
  - ➢ Library Paths - Where are libraries located that executable target build against?
  - ➢ include_directories(<dir1>, <dir2>, ..., <dirN>)
  - ➢ link_directories(<dir1>, <dir2>, ..., <dirN>)

```
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${Boost_INCLUDE_DIRS}
)
```

CMAKELISTS.TXT

- ❑ Invoke message/service/action generation (generate_messages())
  - ✓ Actually generate the language-specific message and service files.

```
generate_messages(
   DEPENDENCIES
   std_msgs
   actionlib_msgs
 )
```

- ❑ Libraries/Executables to build (add_library()/add_executable()/target_link_libraries())
  - ✓ Used to specify libraries to build

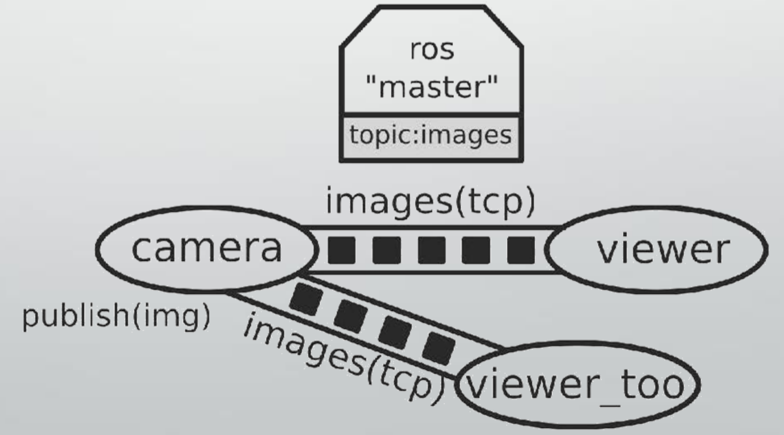
```
add_executable(demo_msg_publisher src/demo_msg_publisher.cpp)

add_dependencies(demo_msg_publisher mastering_ros_demo_pkg_generate_messages_cpp)

target_link_libraries(demo_msg_publisher ${catkin_LIBRARIES})
```

Working with ROS topics

- ❑ Buses in which ROS nodes exchange messages

- ❑ The basic way of communicating between two nodes

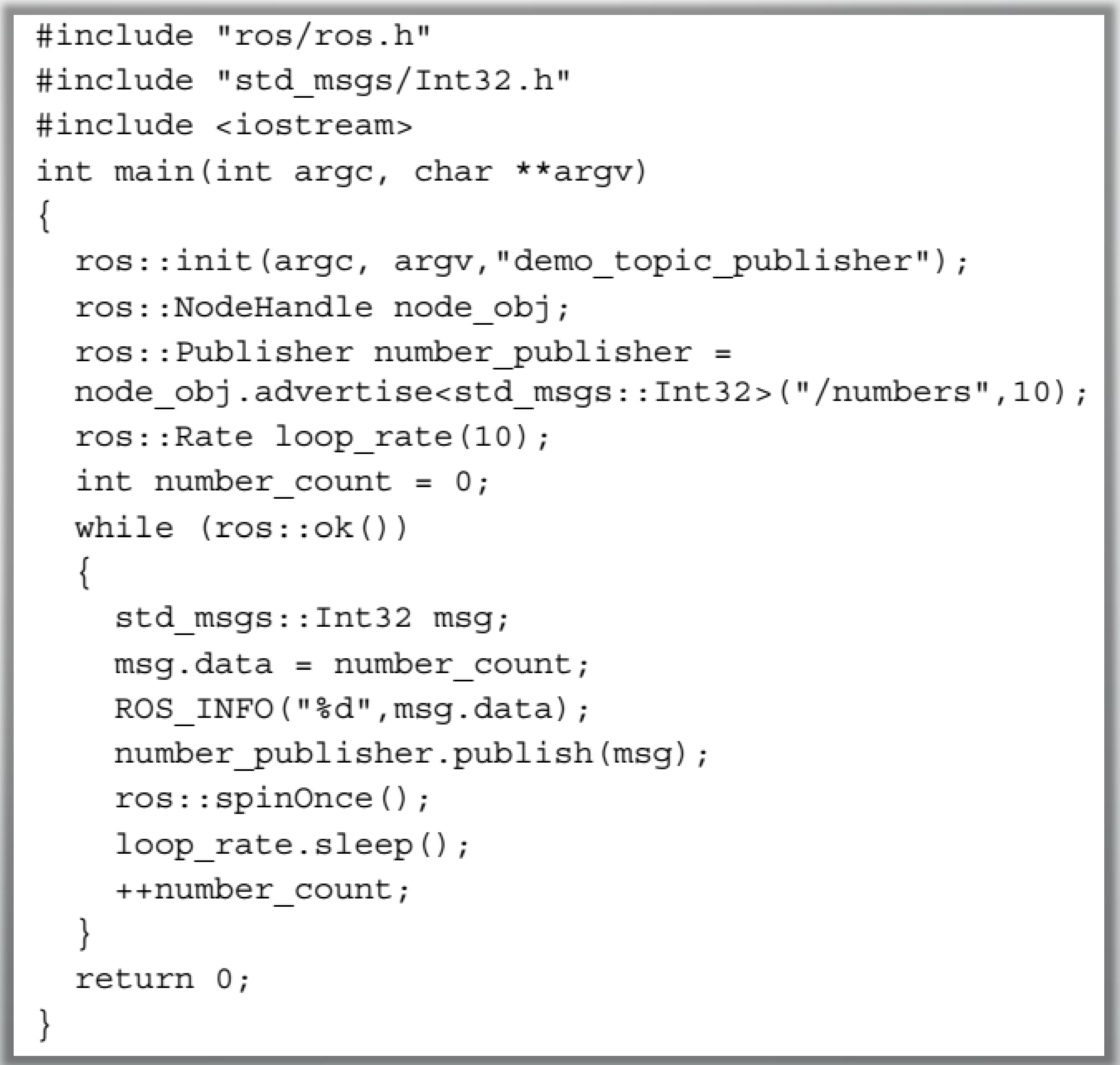
- ❑ Creating two ROS nodes for publishing a topic and subscribing the same



CREATING ROS NODES

❑ demo_topic_publisher.cpp

✓ publishes an integer value on a topic called /numbers

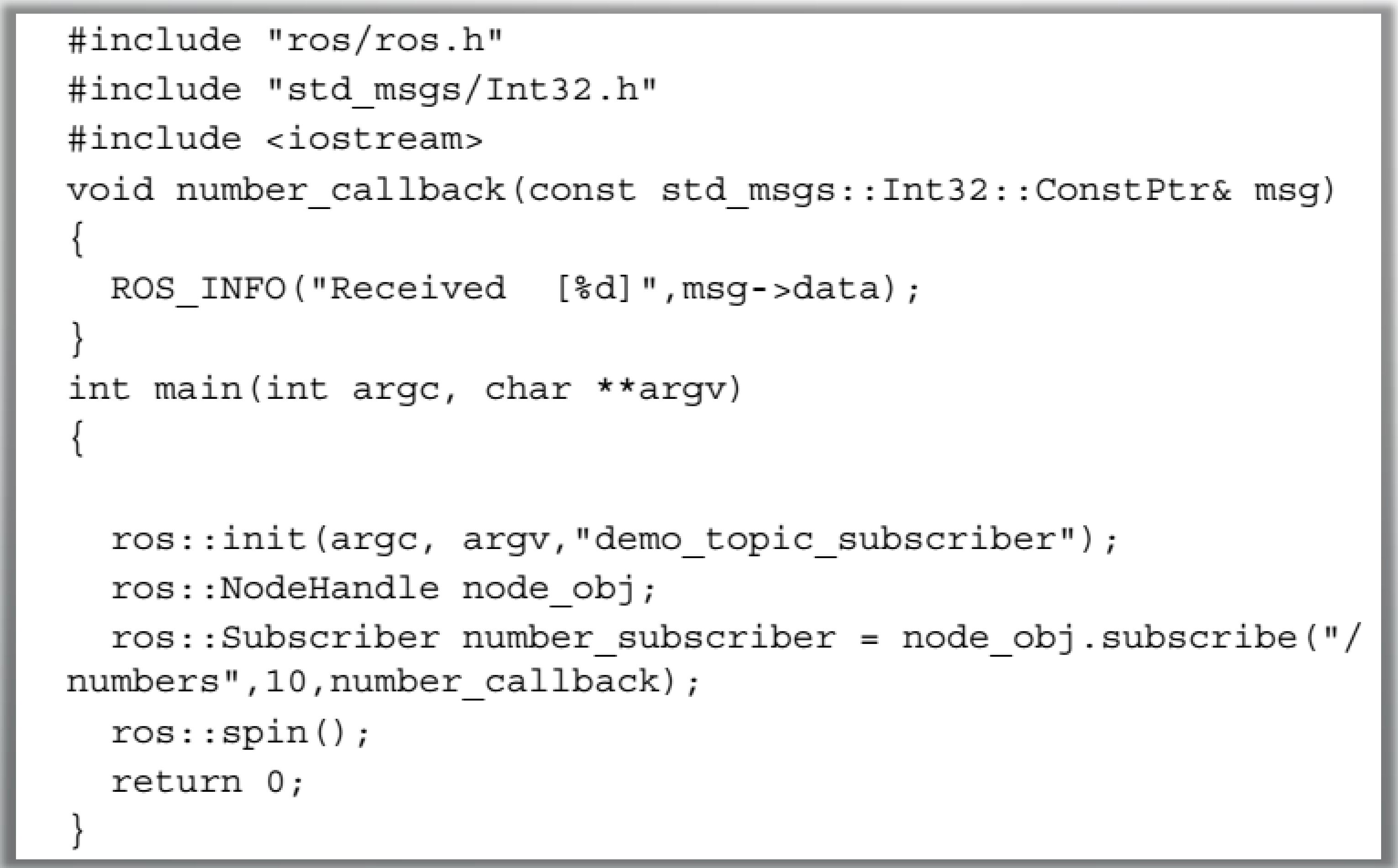


```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
int main(int argc, char **argv)
{
  ros::init(argc, argv,"demo_topic_publisher");
  ros::NodeHandle node_obj;
  ros::Publisher number_publisher =
  node_obj.advertise<std_msgs::Int32>("/numbers",10);
  ros::Rate loop_rate(10);
  int number_count = 0;
  while (ros::ok())
  {
    std_msgs::Int32 msg;
    msg.data = number_count;
    ROS_INFO("%d",msg.data);
    number_publisher.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++number_count;
  }
  return 0;
}
```

CREATING ROS NODES

- ❑ The subscriber node

  `demo_topic_subscriber.cpp`



```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <iostream>
void number_callback(const std_msgs::Int32::ConstPtr& msg)
{
  ROS_INFO("Received  [%d]",msg->data);
}
int main(int argc, char **argv)
{

  ros::init(argc, argv,"demo_topic_subscriber");
  ros::NodeHandle node_obj;
  ros::Subscriber number_subscriber = node_obj.subscribe("/
numbers",10,number_callback);
  ros::spin();
  return 0;
}
```

BUILDING THE NODES

- ❑ Following codes are used in `CMakeLists.txt` to build the nodes.

```
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${Boost_INCLUDE_DIRS}
)

#This will create executables of the nodes
add_executable(demo_topic_publisher     src/demo_topic_publisher.cpp)
add_executable(demo_topic_subscriber    src/demo_topic_subscriber.cpp)

#This will generate message header file before building the target
add_dependencies(demo_topic_publisher  mastering_ros_demo_pkg_generate_messages_cpp)
add_dependencies(demo_topic_subscriber mastering_ros_demo_pkg_generate_messages_cpp)

#This will link executables to the appropriate libraries
target_link_libraries(demo_topic_publisher     ${catkin_LIBRARIES})
target_link_libraries(demo_topic_subscriber    ${catkin_LIBRARIES})
```

BUILDING THE NODES

- switch to workspace

```
$ cd ~/catkin_ws
```

- Build mastering_ros_demo_package as follows:

```
$ catkin_make mastering_ros_demo_package
```

- create executables in `~/catkin_ws/devel/lib/<package name>.`

□ execute the nodes

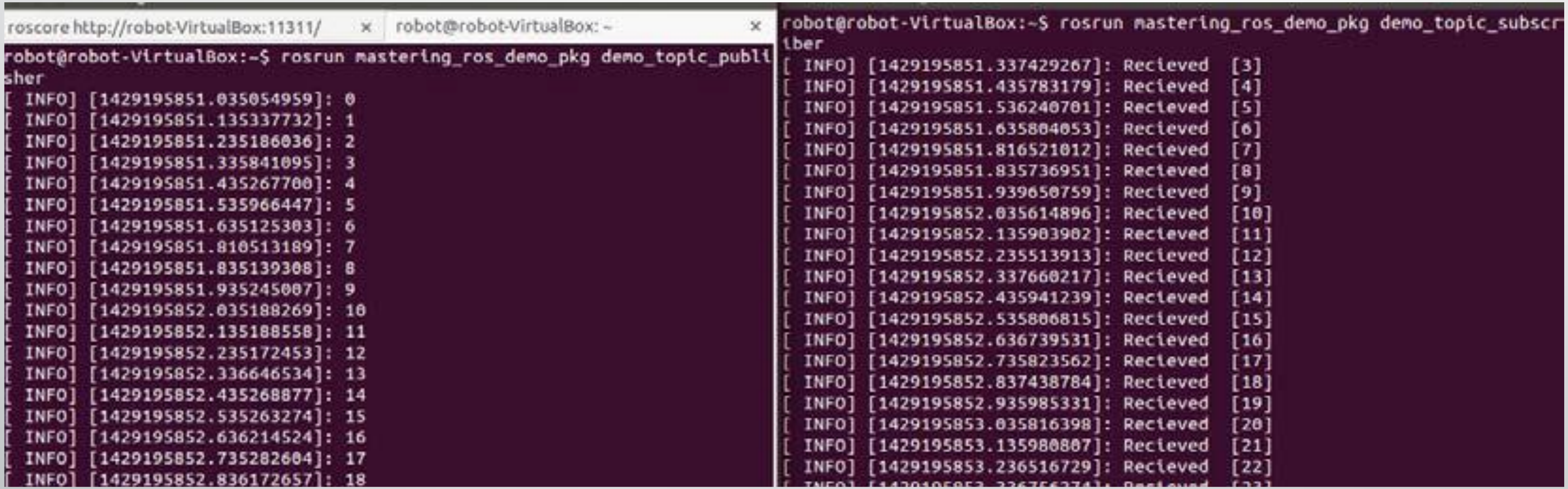
- start roscore:

```
$ roscore
```

- run both commands in two shells

```
$ rosrun mastering_ros_demo_package demo_topic_publisher
$ rosrun mastering_ros_demo_package demo_topic_subscriber
```

BUILDING THE NODES


roscore http://robot-VirtualBox:11311/
robot@robot-VirtualBox: ~

```
robot@robot-VirtualBox:~$ rosrun mastering_ros_demo_pkg demo_topic_publi
sher
[ INFO] [1429195851.035054959]: 0
[ INFO] [1429195851.135337732]: 1
[ INFO] [1429195851.235186036]: 2
[ INFO] [1429195851.335841095]: 3
[ INFO] [1429195851.435267700]: 4
[ INFO] [1429195851.535966447]: 5
[ INFO] [1429195851.635125303]: 6
[ INFO] [1429195851.810513189]: 7
[ INFO] [1429195851.835139308]: 8
[ INFO] [1429195851.935245007]: 9
[ INFO] [1429195852.035188269]: 10
[ INFO] [1429195852.135188558]: 11
[ INFO] [1429195852.235172453]: 12
[ INFO] [1429195852.336646534]: 13
[ INFO] [1429195852.435268877]: 14
[ INFO] [1429195852.535263274]: 15
[ INFO] [1429195852.636214524]: 16
[ INFO] [1429195852.735282604]: 17
[ INFO] [1429195852.836172657]: 18
```

```
robot@robot-VirtualBox:~$ rosrun mastering_ros_demo_pkg demo_topic_subscr
iber
[ INFO] [1429195851.337429267]: Recieved  [3]
[ INFO] [1429195851.435783179]: Recieved  [4]
[ INFO] [1429195851.536240701]: Recieved  [5]
[ INFO] [1429195851.635804053]: Recieved  [6]
[ INFO] [1429195851.816521012]: Recieved  [7]
[ INFO] [1429195851.835736951]: Recieved  [8]
[ INFO] [1429195851.939650759]: Recieved  [9]
[ INFO] [1429195852.035614896]: Recieved  [10]
[ INFO] [1429195852.135903902]: Recieved  [11]
[ INFO] [1429195852.235513913]: Recieved  [12]
[ INFO] [1429195852.337660217]: Recieved  [13]
[ INFO] [1429195852.435941239]: Recieved  [14]
[ INFO] [1429195852.535806815]: Recieved  [15]
[ INFO] [1429195852.636739531]: Recieved  [16]
[ INFO] [1429195852.735823562]: Recieved  [17]
[ INFO] [1429195852.837438784]: Recieved  [18]
[ INFO] [1429195852.935985331]: Recieved  [19]
[ INFO] [1429195853.035816398]: Recieved  [20]
[ INFO] [1429195853.135980807]: Recieved  [21]
[ INFO] [1429195853.236516729]: Recieved  [22]
```

BUILDING THE NODES

❖ NODE DEBUGGING TOOLS

- `$ rosnode list`
  - ✓ This will list the active nodes
- `$ rosnode info demo_topic_publisher`
  - ✓ This will get the info of the publisher node
- `$ rostopic echo /numbers`
  - ✓ This will display the value sending through the `/numbers` topic
- `$ rostopic type /numbers`
  - ✓ This will print the message type of the `/numbers` topic

Adding custom msg and srv files

- ❑ Custom messages and services definitions

- ❑ These definitions inform ROS about the type of data and name of data to be transmitted from a ROS node

  - ➢ message definitions in a .msg file
  - ➢ service definition in a .srv file

- msg: msg files are simple text files that describe the fields of a ROS message. They are used to generate source code for messages in different languages.
- srv: an srv file describes a service. It is composed of two parts: a request and a response.

Adding Custom MSG File

❑ Create a message file called demo_msg.msg

```
string greeting
int32 number
```

❑ Corresponding lines in Package.xml file and CMakeLists.txt

➢ Package.xml

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

➢ CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
       message_generation
)
add_message_files(
   FILES
   demo_msg.msg
)
## Generate added messages and services with any
dependencies listed here
generate_messages(
   DEPENDENCIES
   std_msgs
   actionlib_msgs
)
```

ADDING CUSTOM MSG FILE

- Compile and build the package:

```
$ cd ~/catkin_ws/
$ catkin_make
```

- To check whether the message is built properly, we can use the rosmsg command:

```
$ rosmsg   show   mastering_ros_demo_pkg/demo_msg
```

- Now we can build a publisher and subscriber using the custom message type

```
  mastering_ros_demo_pkg::demo_msg msg;
  std::stringstream ss;
  ss << "hello world ";
  msg.greeting = ss.str();
  msg.number = number_count;

#include "mastering_ros_demo_pkg/demo_msg.h"
#include <sstream>
```

Adding Custom msg file

- Run roscore:

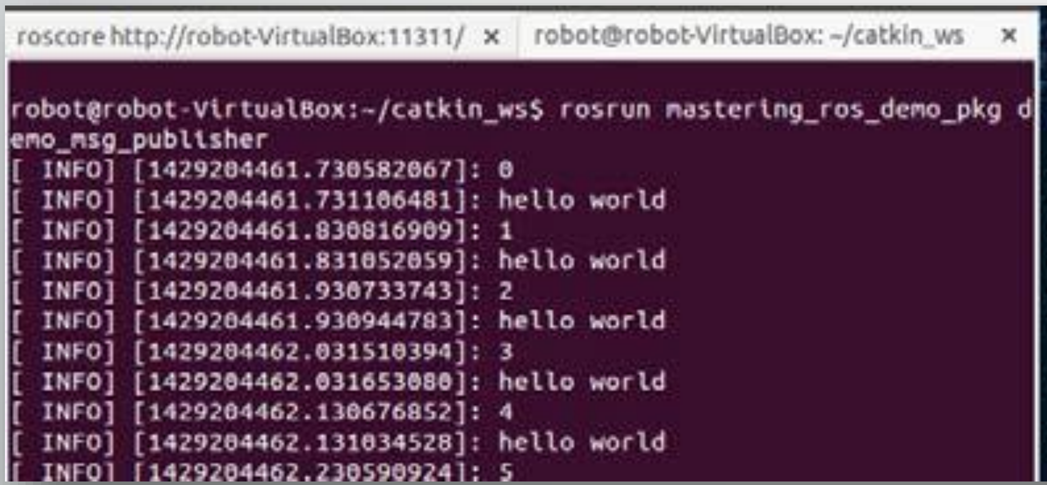
```
$ roscore
```

- Start the custom message publisher node:

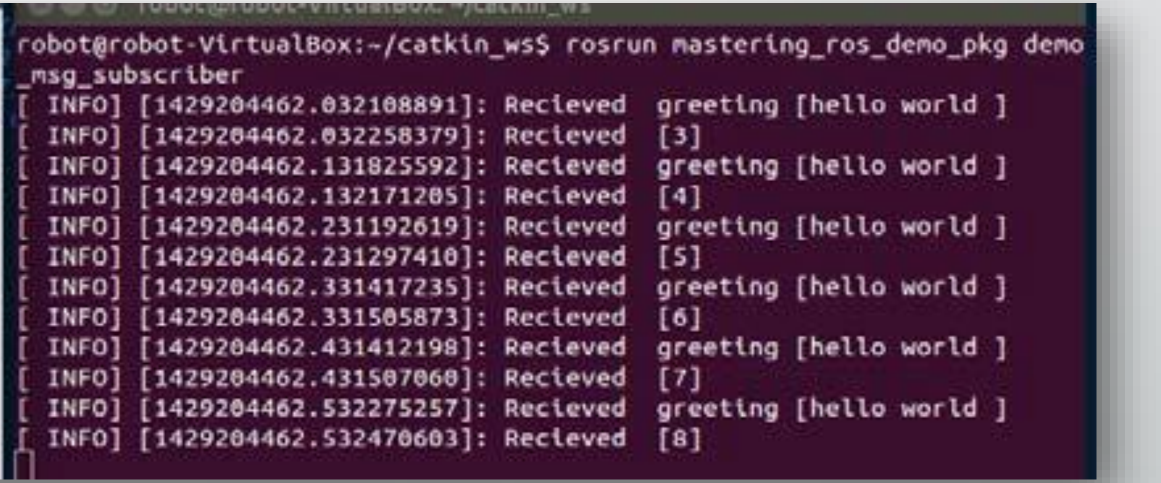
```
$ rosrun mastering_ros_demo_pkg demo_msg_publisher
```

- Start the custom message subscriber node:

```
$ rosrun mastering_ros_demo_pkg demo_msg_subscriber
```


roscore http://robot-VirtualBox:11311/ ×
robot@robot-VirtualBox: ~/catkin_ws ×

```
robot@robot-VirtualBox:~/catkin_ws$ rosrun mastering_ros_demo_pkg d
emo_msg_publisher
[ INFO] [1429204461.730582067]: 0
[ INFO] [1429204461.731106481]: hello world
[ INFO] [1429204461.830816909]: 1
[ INFO] [1429204461.831052059]: hello world
[ INFO] [1429204461.930733743]: 2
[ INFO] [1429204461.930944783]: hello world
[ INFO] [1429204462.031510394]: 3
[ INFO] [1429204462.031653080]: hello world
[ INFO] [1429204462.130676852]: 4
[ INFO] [1429204462.131034528]: hello world
[ INFO] [1429204462.230590924]: 5
```



```
robot@robot-VirtualBox:~/catkin_ws$ rosrun mastering_ros_demo_pkg demo
_msg_subscriber
[ INFO] [1429204462.032108891]: Recieved  greeting [hello world ]
[ INFO] [1429204462.032258379]: Recieved  [3]
[ INFO] [1429204462.131825592]: Recieved  greeting [hello world ]
[ INFO] [1429204462.132171205]: Recieved  [4]
[ INFO] [1429204462.231192619]: Recieved  greeting [hello world ]
[ INFO] [1429204462.231297410]: Recieved  [5]
[ INFO] [1429204462.331417235]: Recieved  greeting [hello world ]
[ INFO] [1429204462.331505873]: Recieved  [6]
[ INFO] [1429204462.431412198]: Recieved  greeting [hello world ]
[ INFO] [1429204462.431507060]: Recieved  [7]
[ INFO] [1429204462.532275257]: Recieved  greeting [hello world ]
[ INFO] [1429204462.532470603]: Recieved  [8]
```

Adding custom srv file

❑ Create a new folder called `srv` in the current package folder
add a `srv` file called `demo_srv.srv`

```
string in
---
string out
```

❑ Corresponding lines in Package.xml file and CMakeLists.txt

- ➢ **Package.xml**

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

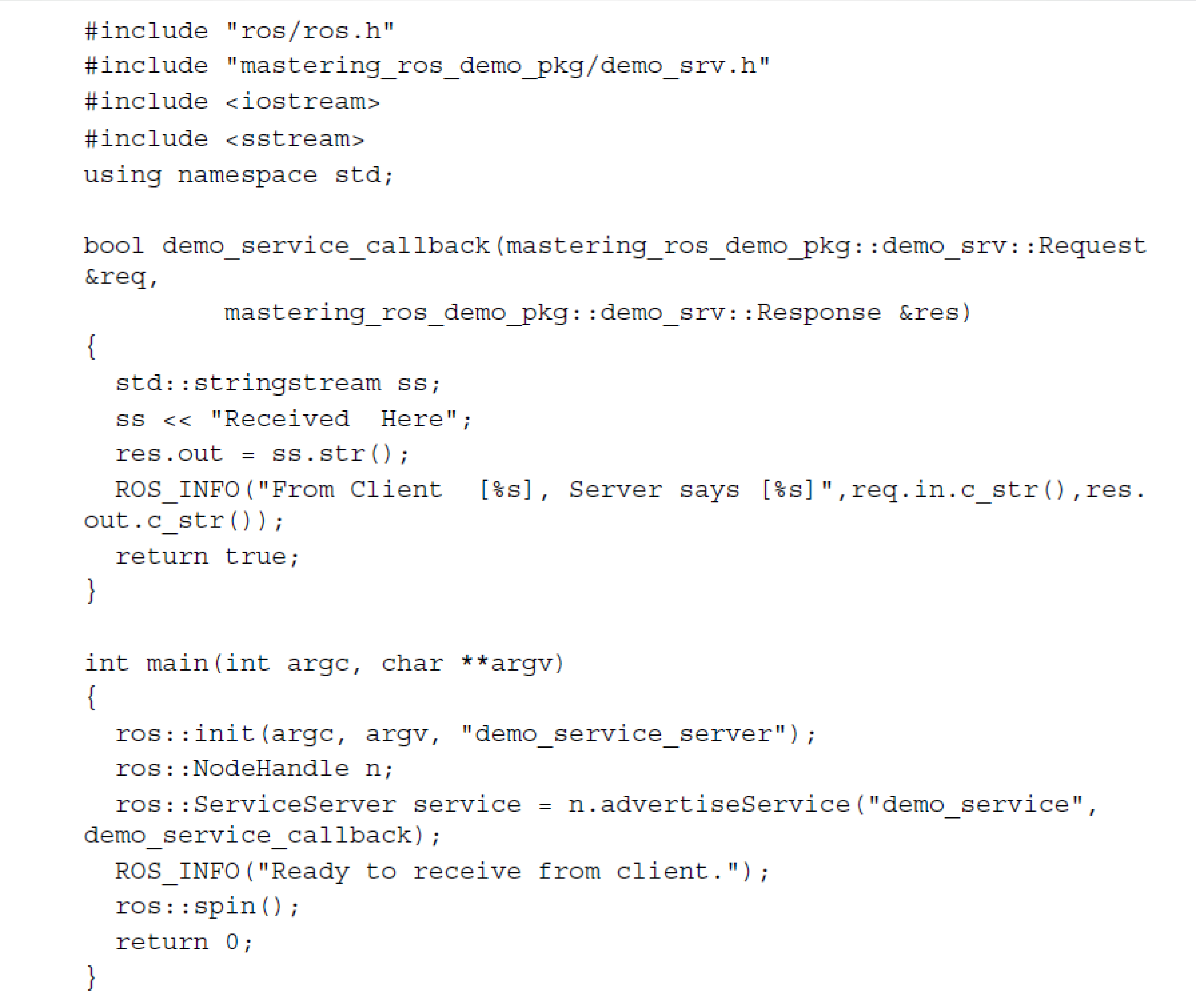
- ➢ **CMakeLists.txt**

```
catkin_package( …
               message_runtime
               )
## Generate services in the 'srv' folder
   add_service_files(
   FILES
   demo_srv.srv
   )
```

WORKING WITH ROS SERVICES

- ❑ Create ROS nodes, which can use the services definition

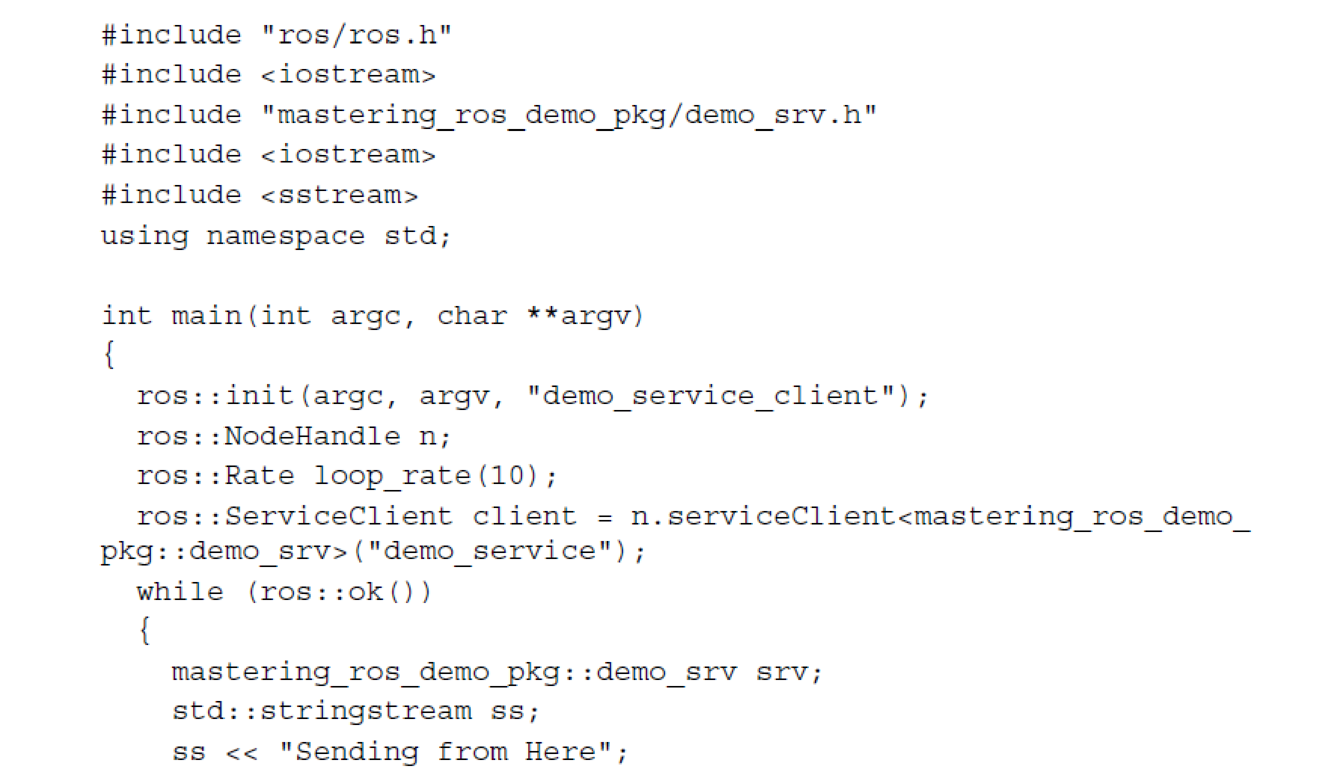
  - ➢ demo_service_server.cpp



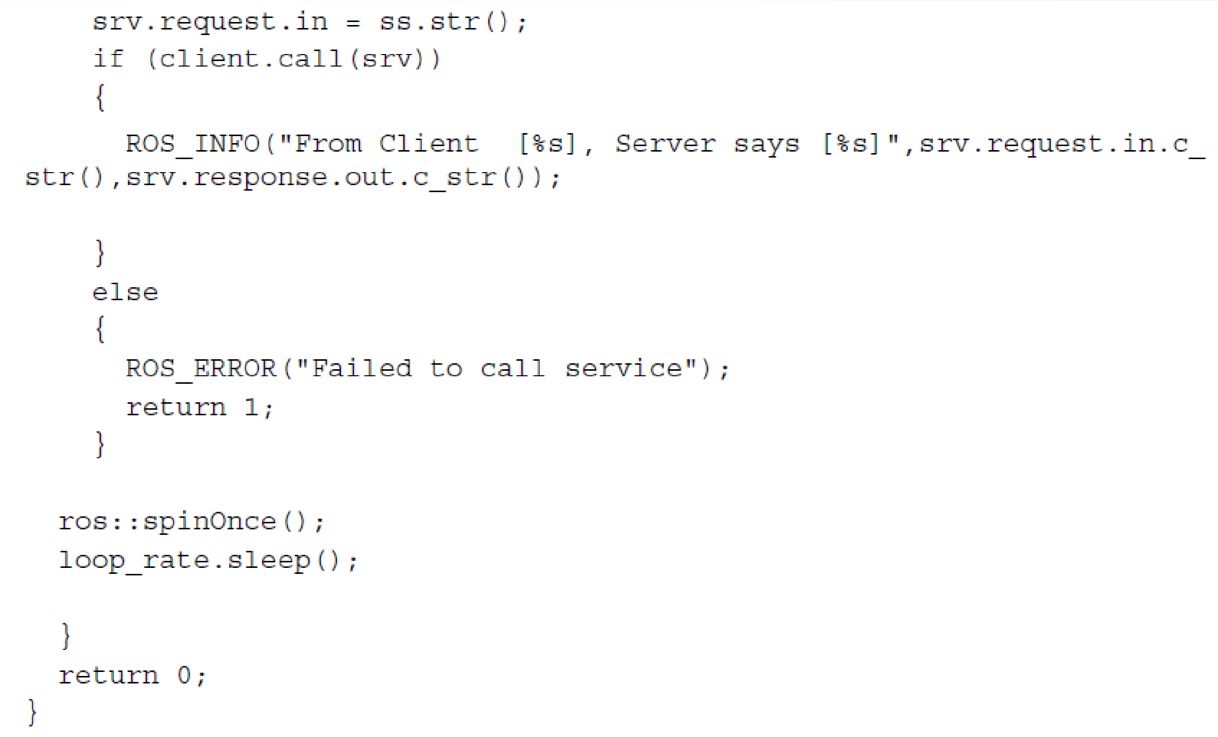
```
#include "ros/ros.h"
#include "mastering_ros_demo_pkg/demo_srv.h"
#include <iostream>
#include <sstream>
using namespace std;

bool demo_service_callback(mastering_ros_demo_pkg::demo_srv::Request
&req,
         mastering_ros_demo_pkg::demo_srv::Response &res)
{
  std::stringstream ss;
  ss << "Received  Here";
  res.out = ss.str();
  ROS_INFO("From Client  [%s], Server says [%s]",req.in.c_str(),res.
out.c_str());
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "demo_service_server");
  ros::NodeHandle n;
  ros::ServiceServer service = n.advertiseService("demo_service",
demo_service_callback);
  ROS_INFO("Ready to receive from client.");
  ros::spin();
  return 0;
}
```

WORKING WITH ROS SERVICES

- demo_service_client.cpp



```
#include "ros/ros.h"
#include <iostream>
#include "mastering_ros_demo_pkg/demo_srv.h"
#include <iostream>
#include <sstream>
using namespace std;

int main(int argc, char **argv)
{
  ros::init(argc, argv, "demo_service_client");
  ros::NodeHandle n;
  ros::Rate loop_rate(10);
  ros::ServiceClient client = n.serviceClient<mastering_ros_demo_
pkg::demo_srv>("demo_service");
  while (ros::ok())
  {
    mastering_ros_demo_pkg::demo_srv srv;
    std::stringstream ss;
    ss << "Sending from Here";
```



```
    srv.request.in = ss.str();
    if (client.call(srv))
    {
      ROS_INFO("From Client  [%s], Server says [%s]",srv.request.in.c_
str(),srv.response.out.c_str());

    }
    else
    {
      ROS_ERROR("Failed to call service");
      return 1;
    }

  ros::spinOnce();
  loop_rate.sleep();

  }
  return 0;
}
```

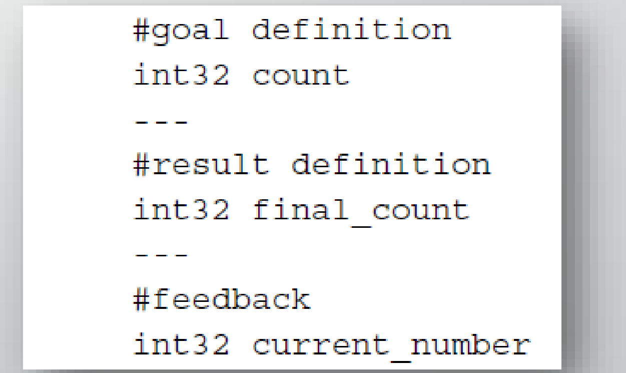
ROSSERVICE COMMANDS

- `$ rosservice list:`
  - ✓ This will list the current ROS services
- `$ rosservice type /demo_service`:
  - ✓ This will print the message type of /demo_service
- `$ rosservice info /demo_service:`
  - ✓ This will print the information of /demo_service

WORKING WITH ROS ACTIONLIB

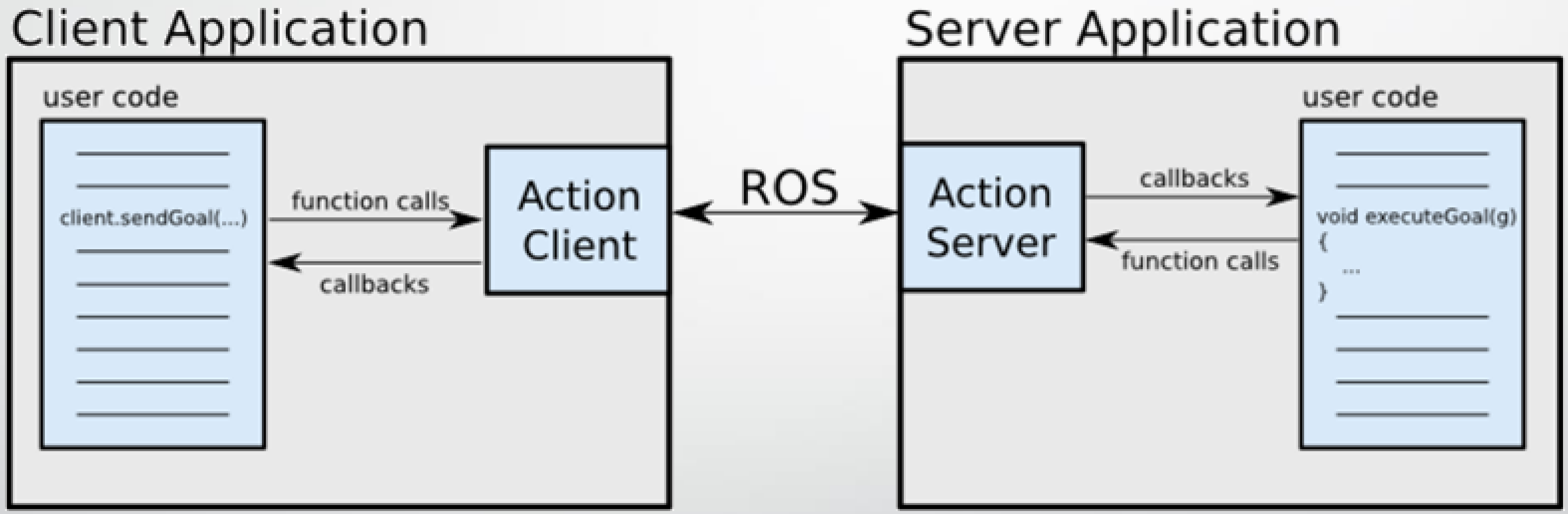
❑ When to use actionlib

- ❑ action specification
  - ➢ .action file with the following parts
    - ❑ **Goal**
      - ✓ To be executed by the action server
    - ❑ **Feedback**
      - ✓ The current operation inside the callback function
    - ❑ **Result**
      - ✓ A final result of completion



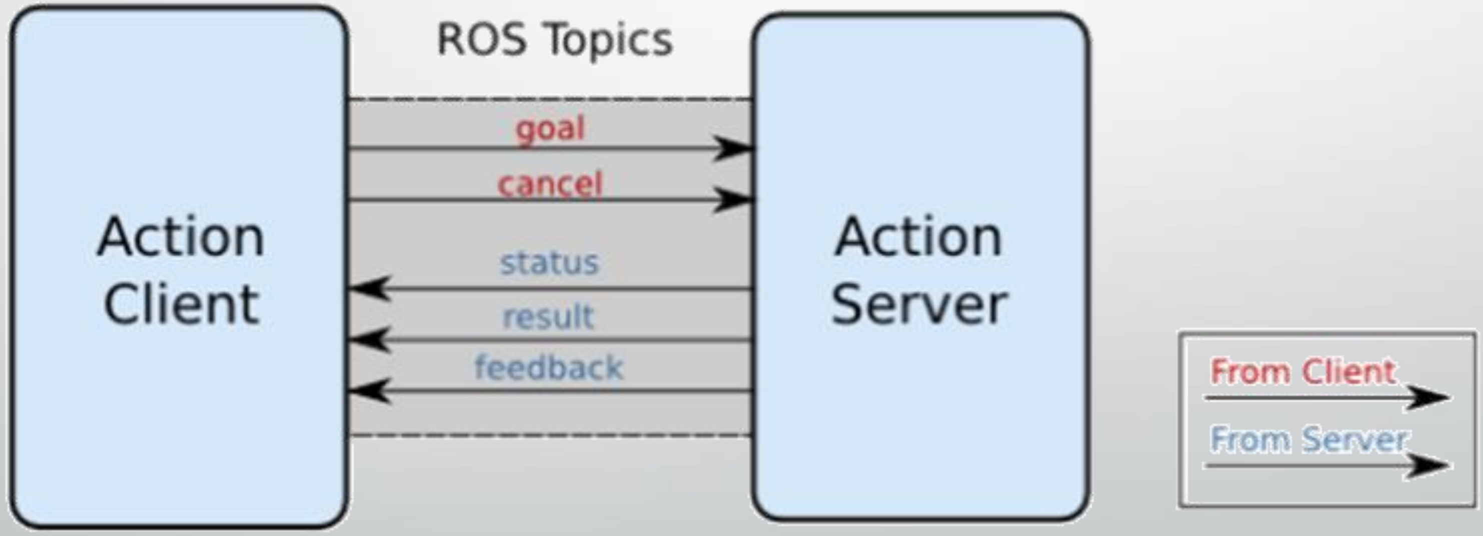
```
#goal definition
int32 count
---
#result definition
int32 final_count
---
#feedback
int32 current_number
```

Working with ROS actionlib

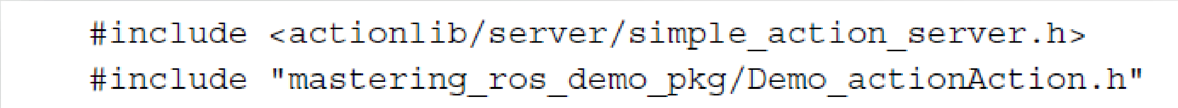


WORKING WITH ROS ACTIONLIB

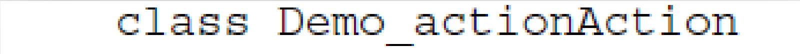
- ❑ Action protocol relies on ROS topics to transport messages
- ❑ **Action Interface**



CREATING THE ROS ACTION SERVER

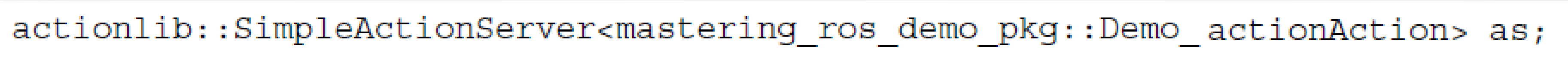


```
#include <actionlib/server/simple_action_server.h>
#include "mastering_ros_demo_pkg/Demo_actionAction.h"
```



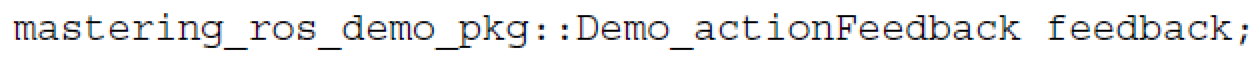
```
class Demo_actionAction
```

- ✓ Containing the action class definition



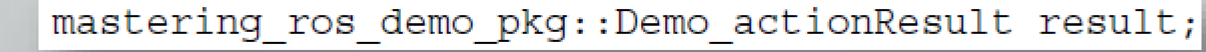
```
actionlib::SimpleActionServer<mastering_ros_demo_pkg::Demo_ actionAction> as;
```

- ✓ Creating an action server instance



```
mastering_ros_demo_pkg::Demo_actionFeedback feedback;
```

- ✓ Creating a feedback instance

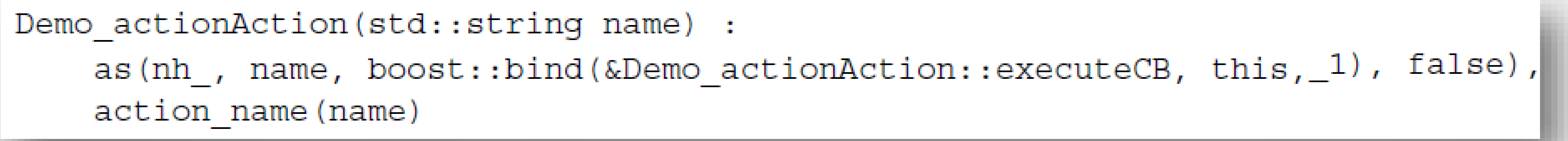


```
mastering_ros_demo_pkg::Demo_actionResult result;
```

- ✓ And finally creating a result instance

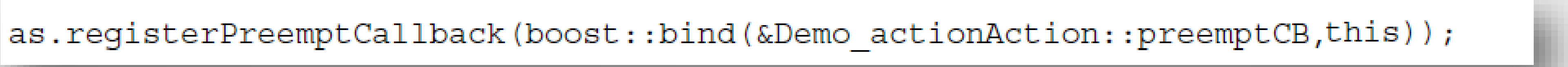
CREATING THE ROS ACTION SERVER

- The action constructor



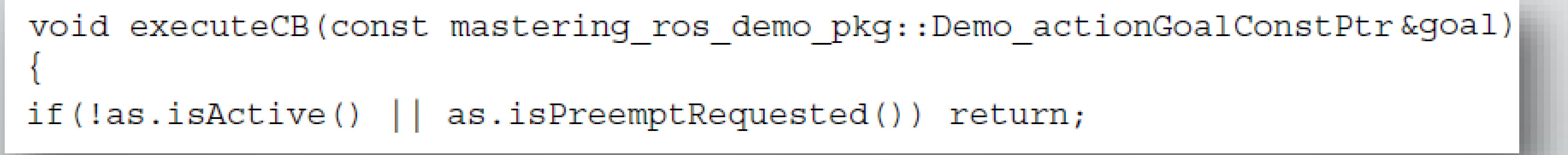
```
Demo_actionAction(std::string name) :
    as(nh_, name, boost::bind(&Demo_actionAction::executeCB, this,_1), false),
    action_name(name)
```

- Registering a callback when the action is preempted



```
as.registerPreemptCallback(boost::bind(&Demo_actionAction::preemptCB,this));
```

- The callback definition



```
void executeCB(const mastering_ros_demo_pkg::Demo_actionGoalConstPtr &goal)
{
if(!as.isActive() || as.isPreemptRequested()) return;
```

✓ Other actionlib commands could be found in: Here

APPLICATIONS OF TOPICS, SERVICES, AND ACTIONLIB

- ➢ **topics**
  - ✓ a unidirectional communication method,
- ➢ **services**
  - ✓ a bidirectional request/reply communication
- ➢ **actionlib**
  - ✓ a modified form of ROS services

- **Topics**: Robot teleoperation, publishing odometry, sending robot transform (TF), and sending robot joint states
- **Services**: This saves camera calibration parameters to a file, saves a map of the robot after SLAM, and loads a parameter file
- **Actionlib**: This is used in motion planners and ROS navigation stacks

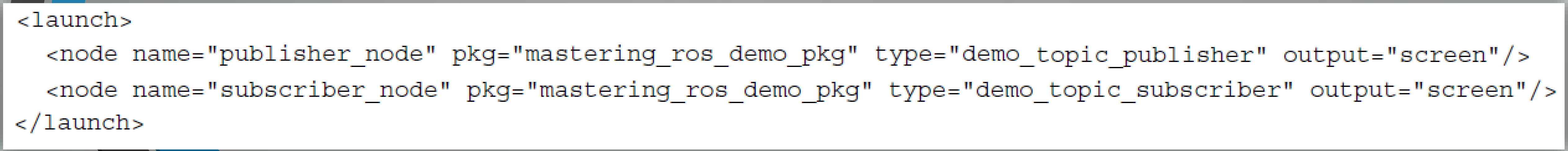
INTRODUCTION TO ROS

ROS-PART 3

CREATING LAUNCH FILES

- ❑ launching more than one node
  - ➢ Previously the codes should be each in a terminal one by one
  - ✓ It is possible to write all nodes inside a XML based file called launch files and using a command called `roslaunch`
  - ✓ automatically starts ROS Master and the parameter server

- ➢ Create a `.launch` file in launch folder of the package with the following content:



```
<launch>
  <node name="publisher_node" pkg="mastering_ros_demo_pkg" type="demo_topic_publisher" output="screen"/>
  <node name="subscriber_node" pkg="mastering_ros_demo_pkg" type="demo_topic_subscriber" output="screen"/>
</launch>
```

CREATING LAUNCH FILES

- Using the following command the launch file could be run

```
$ roslaunch package_name luanchfile_name.launch
```

- The list of nodes and the logs could be reached by the following commands:

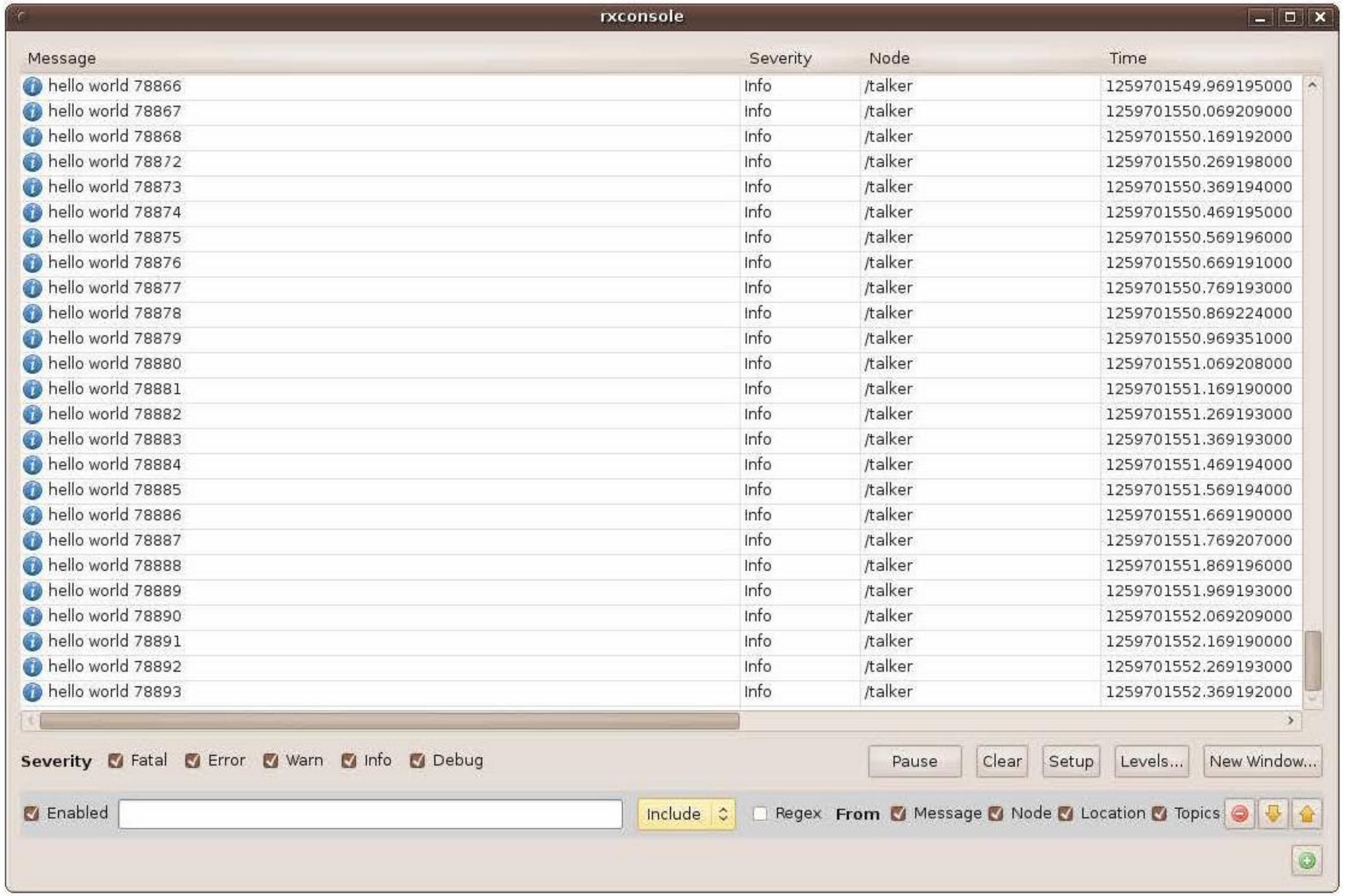
```
$ rosnode list
$ rqt_console
```

DEBUGGING
ROSOUT

- ROS provides mechanisms in all languages for specifying dfferent levels of human readable log messages
  - The five levels are:
    - Fatal
    - Error
    - Info
    - Debug
  - Coressponding logging commands in C++:
    - ROS_FATAL(…)
    - ROS_WARN(…)
    - ROS_INFO(…)
    - ROS_DEBUG(…)

DEBUGGING

RXCONSOL



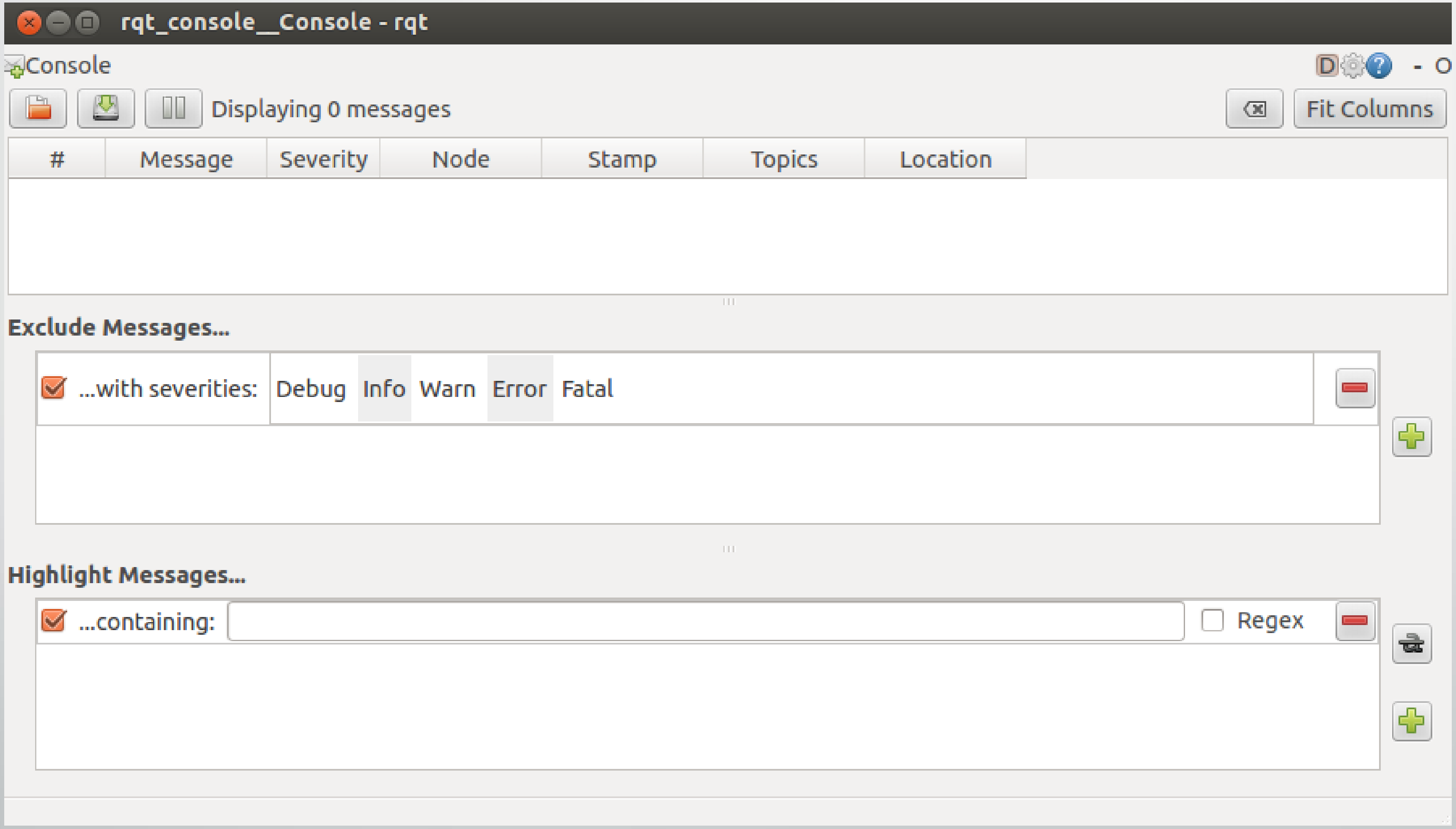
Message	Severity	Node	Time
hello world 78866	Info	/talker	1259701549.969195000
hello world 78867	Info	/talker	1259701550.069209000
hello world 78868	Info	/talker	1259701550.169192000
hello world 78872	Info	/talker	1259701550.269198000
hello world 78873	Info	/talker	1259701550.369194000
hello world 78874	Info	/talker	1259701550.469195000
hello world 78875	Info	/talker	1259701550.569196000
hello world 78876	Info	/talker	1259701550.669191000
hello world 78877	Info	/talker	1259701550.769193000
hello world 78878	Info	/talker	1259701550.869224000
hello world 78879	Info	/talker	1259701550.969351000
hello world 78880	Info	/talker	1259701551.069208000
hello world 78881	Info	/talker	1259701551.169190000
hello world 78882	Info	/talker	1259701551.269193000
hello world 78883	Info	/talker	1259701551.369193000
hello world 78884	Info	/talker	1259701551.469194000
hello world 78885	Info	/talker	1259701551.569194000
hello world 78886	Info	/talker	1259701551.669190000
hello world 78887	Info	/talker	1259701551.769207000
hello world 78888	Info	/talker	1259701551.869196000
hello world 78889	Info	/talker	1259701551.969193000
hello world 78890	Info	/talker	1259701552.069209000
hello world 78891	Info	/talker	1259701552.169190000
hello world 78892	Info	/talker	1259701552.269193000
hello world 78893	Info	/talker	1259701552.369192000

Severity ☑ Fatal ☑ Error ☑ Warn ☑ Info ☑ Debug

☑ Enabled ______ Include □ Regex **From** ☑ Message ☑ Node ☑ Location ☑ Topics

DEBUGGING

RXCOSOL



Displaying 0 messages

#	Message	Severity	Node	Stamp	Topics	Location

Exclude Messages...

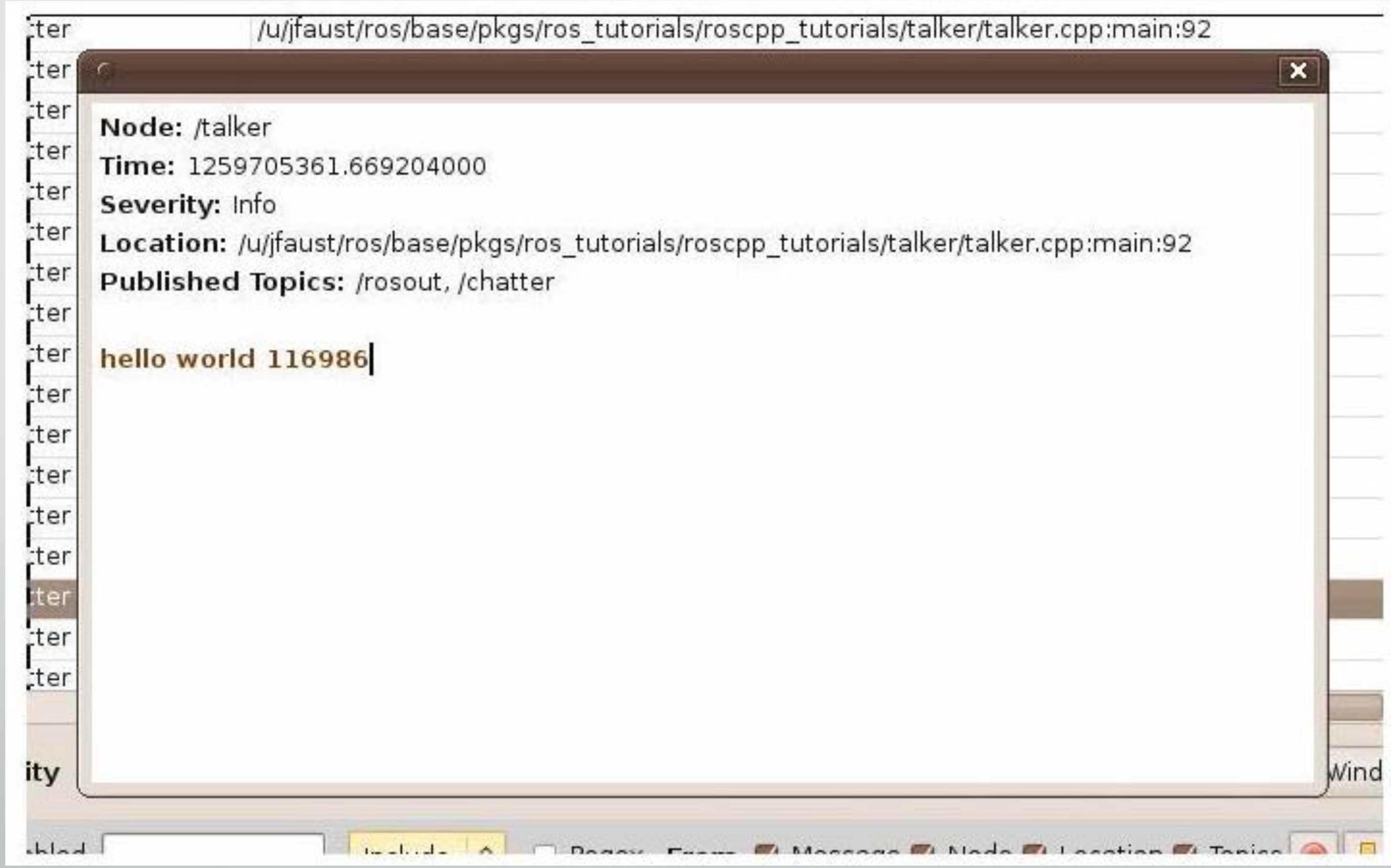
☑ ...with severities: Debug Info Warn Error Fatal

Highlight Messages...

☑ ...containing: □ Regex

DEBUGGING

RXCOSOL

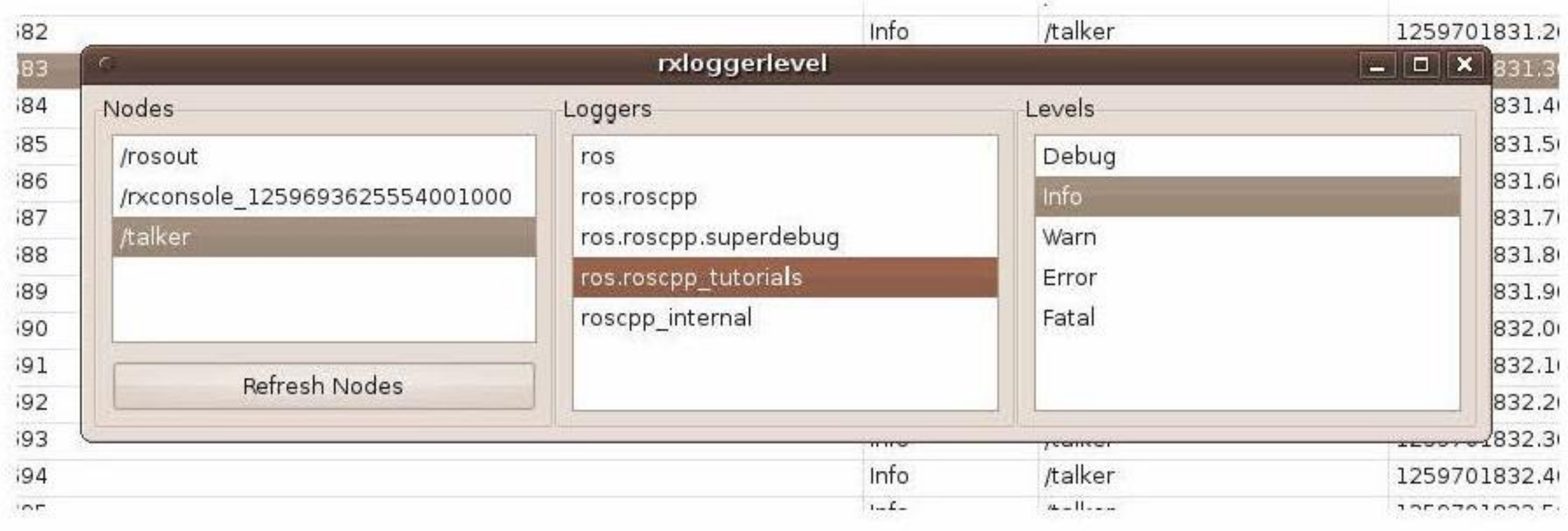


```
Node: /talker
Time: 1259705361.669204000
Severity: Info
Location: /u/jfaust/ros/base/pkgs/ros_tutorials/roscpp_tutorials/talker/talker.cpp:main:92
Published Topics: /rosout, /chatter

hello world 116986
```

/u/jfaust/ros/base/pkgs/ros_tutorials/roscpp_tutorials/talker/talker.cpp:main:92
ter
ity
Wind
Include
Regex
From
Message
Node
Location
Topics

DEBUGGING

RXCOSOL



rxloggerlevel

Nodes

- /rosout
- /rxconsole_1259693625554001000
- /talker

Refresh Nodes

Loggers

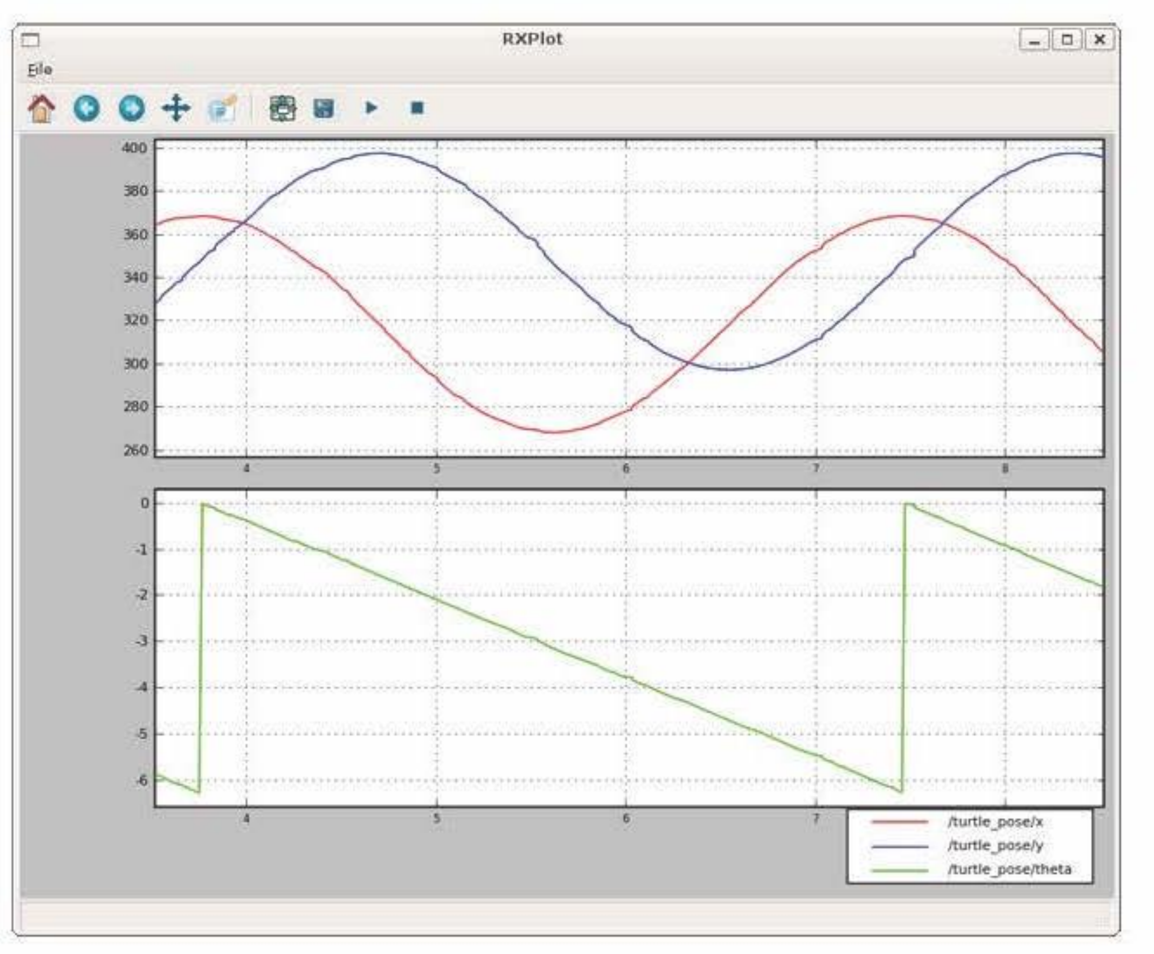
- ros
- ros.roscpp
- ros.roscpp.superdebug
- ros.roscpp_tutorials
- roscpp_internal

Levels

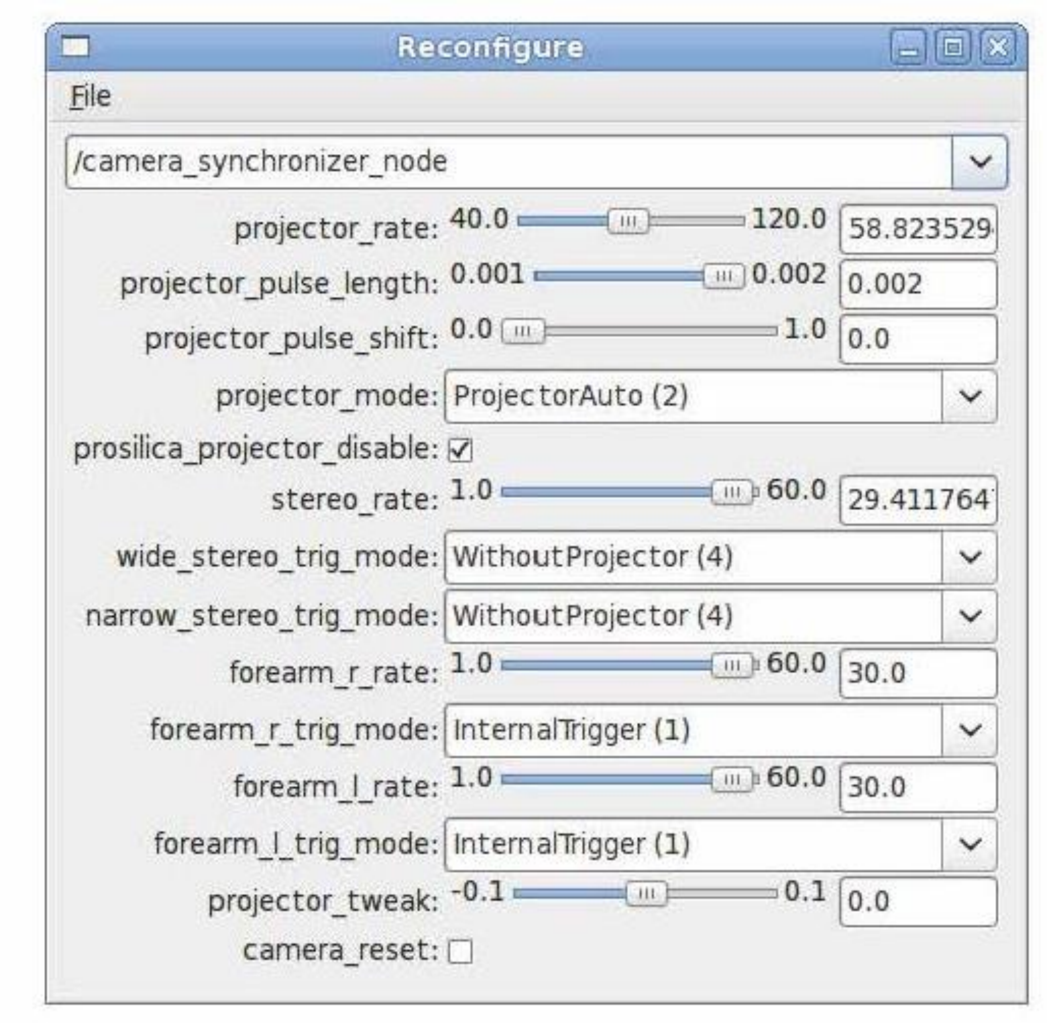
- Debug
- Info
- Warn
- Error
- Fatal

```
582   Info   /talker   1259701831.2
594   Info   /talker   1259701832.4
```

ROS GUI TOOLS
RXPLOT



ROS GUI TOOLS



/camera_synchronizer_node

projector_rate: 40.0 – 120.0 58.823529

projector_pulse_length: 0.001 – 0.002 0.002

projector_pulse_shift: 0.0 – 1.0 0.0

projector_mode: ProjectorAuto (2)

prosilica_projector_disable: ☑

stereo_rate: 1.0 – 60.0 29.411764

wide_stereo_trig_mode: WithoutProjector (4)

narrow_stereo_trig_mode: WithoutProjector (4)

forearm_r_rate: 1.0 – 60.0 30.0

forearm_r_trig_mode: InternalTrigger (1)

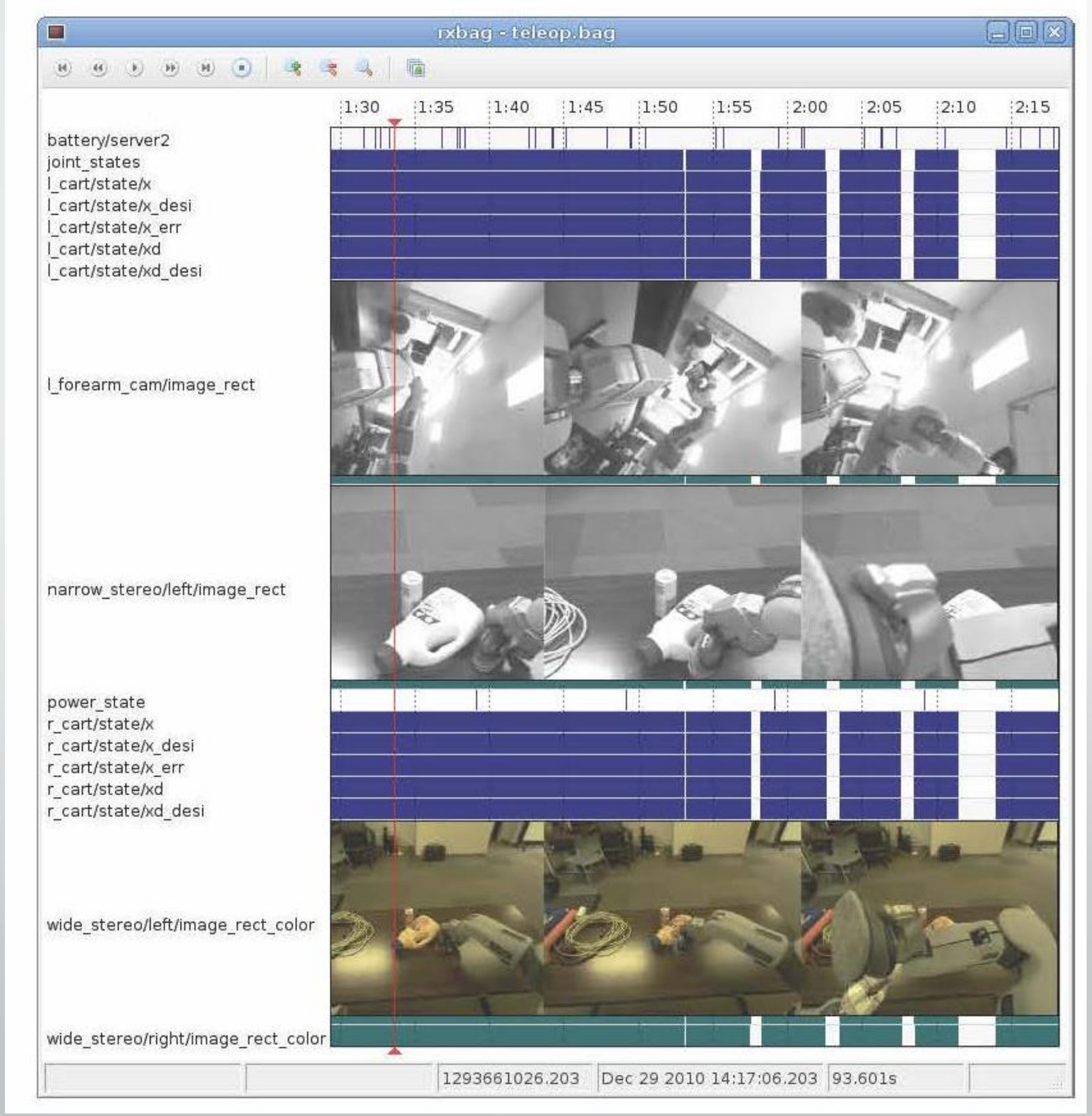
forearm_l_rate: 1.0 – 60.0 30.0

forearm_l_trig_mode: InternalTrigger (1)

projector_tweak: -0.1 – 0.1 0.0

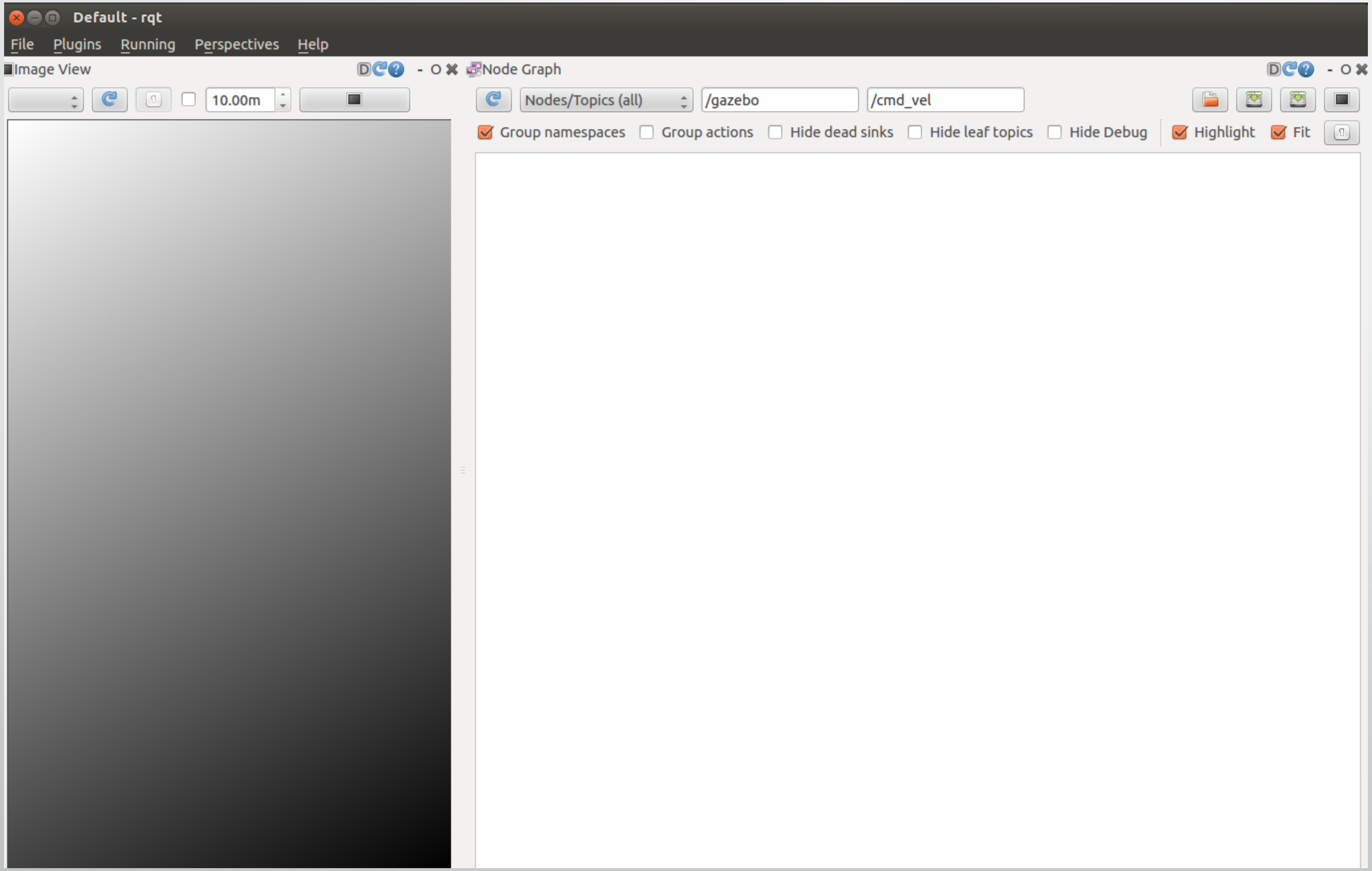
camera_reset: □

ROS GUI TOOLS

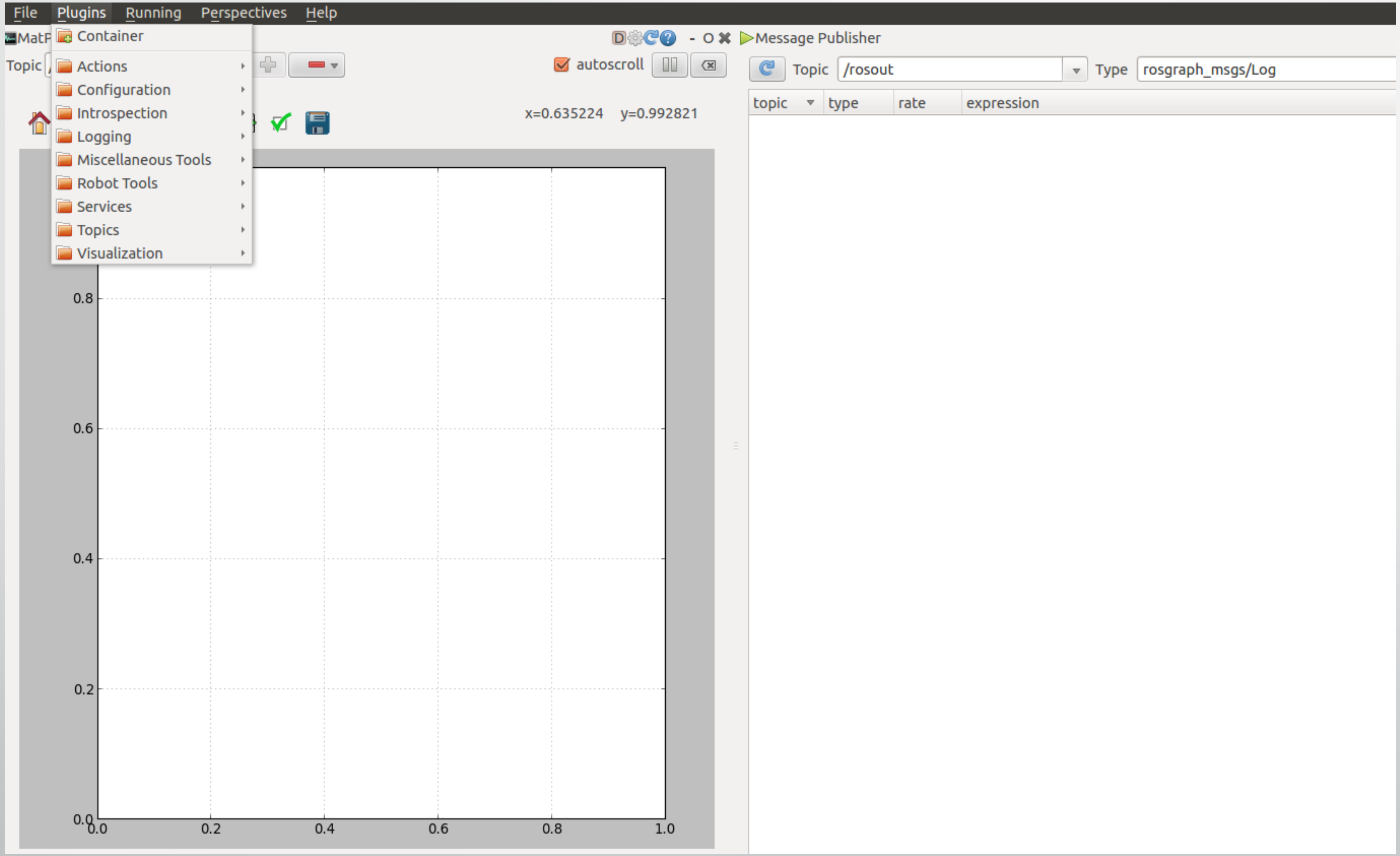


ROS GUI TOOLS

RQT

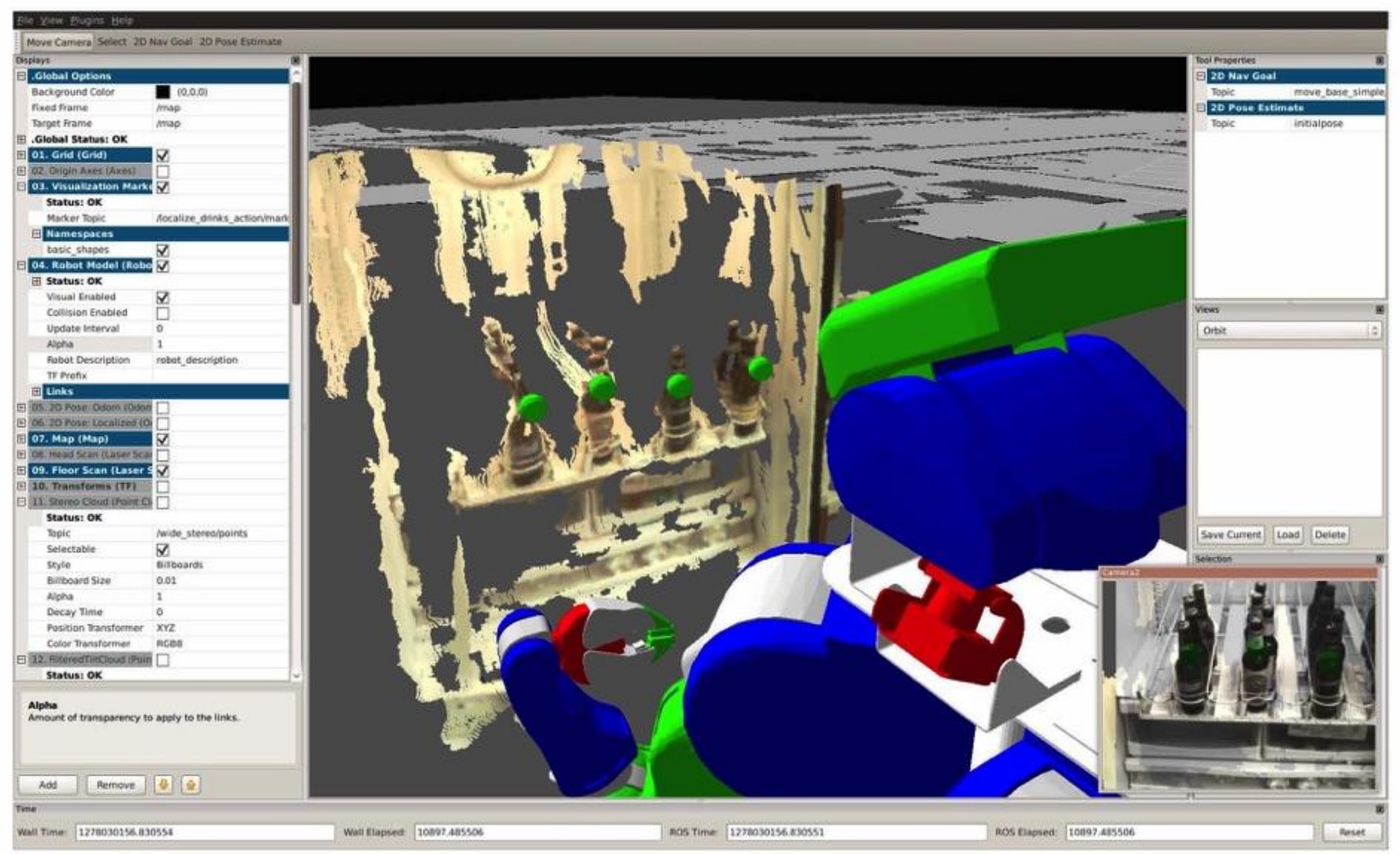


ROS GUI TOOLS



ROS GUI TOOLS

RVIZ - 3D VISUALIZATION



.Global Options	
Background Color	■ (0,0,0)
Fixed Frame	/map
Target Frame	/map
.Global Status: OK	
01. Grid (Grid)	☑
02. Origin Axes (Axes)	□
03. Visualization Marke	☑
Status: OK	
Marker Topic	/localize_drinks_action/mark
Namespaces	
basic_shapes	☑
04. Robot Model (Robo	☑
Status: OK	
Visual Enabled	☑
Collision Enabled	□
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
05. 2D Pose: Odom (Odon	□
06. 2D Pose: Localized (O	□
07. Map (Map)	☑
08. Head Scan (Laser Scar	□
09. Floor Scan (Laser S	☑
10. Transforms (TF)	□
11. Stereo Cloud (Point Cl	□
Status: OK	
Topic	/wide_stereo/points
Selectable	☑
Style	Billboards
Billboard Size	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
12. FilteredTiltCloud (Poin	□
Status: OK	

Alpha
Amount of transparency to apply to the links.

2D Nav Goal	
Topic	move_base_simple
2D Pose Estimate	
Topic	initialpose

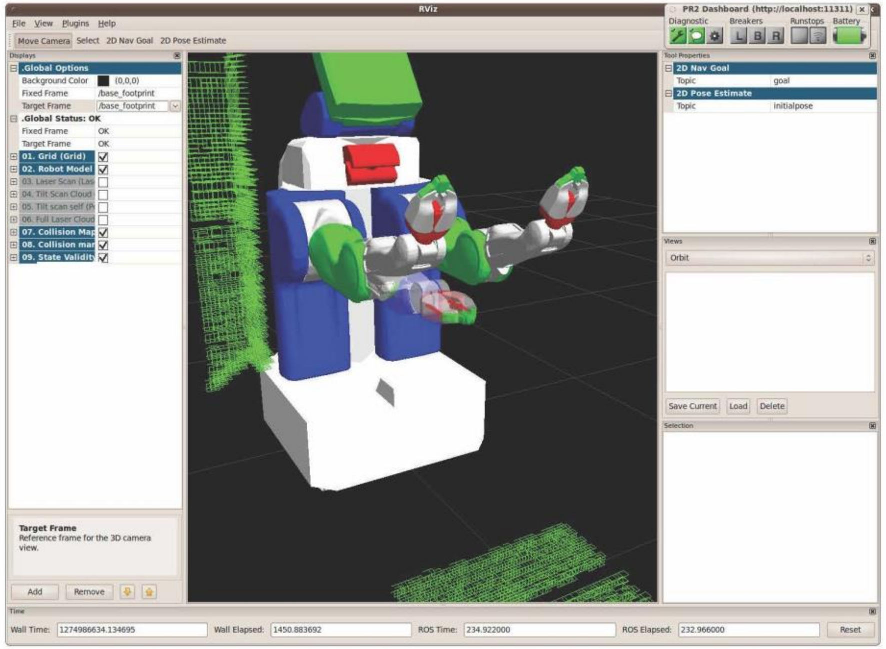
Views: Orbit

Wall Time: 1278030156.830554 Wall Elapsed: 10897.485506 ROS Time: 1278030156.830551 ROS Elapsed: 10897.485506

ROS GUI TOOLS
RVIZ - 3D VISUALIZATION

RViz

- Full simulation environment
- Environment Visualization
- Many, many plugins available
- Can make virtual objects for real system to interact with



.Global Options
Background Color (0,0,0)
Fixed Frame /base_footprint
Target Frame /base_footprint
.Global Status: OK
Fixed Frame OK
Target Frame OK
01. Grid (Grid) ☑
02. Robot Model ☑
03. Laser Scan (Las □
04. Tilt Scan Cloud □
05. Tilt scan self (P □
06. Full Laser Cloud □
07. Collision Map ☑
08. Collision mar ☑
09. State Validity ☑

Target Frame
Reference frame for the 3D camera view.

2D Nav Goal
Topic goal
2D Pose Estimate
Topic initialpose

Orbit

Wall Time: 1274986634.134695 Wall Elapsed: 1450.883692 ROS Time: 234.922000 ROS Elapsed: 232.966000

ROS GUI TOOLS
RVIZ - 3D VISUALIZATION

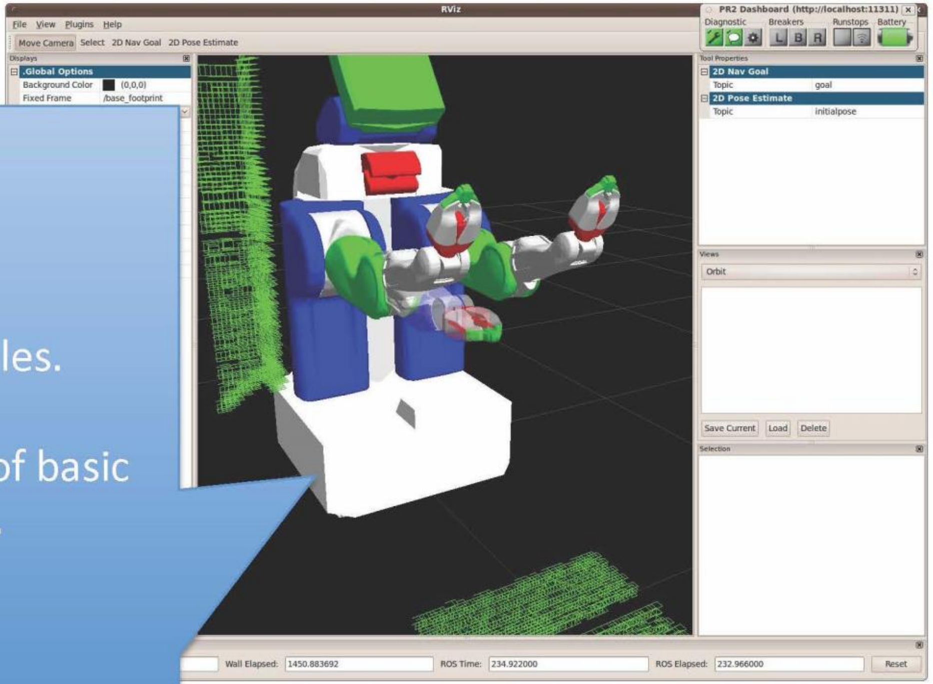
RViz

– Full simulation

3D models are defined by URDF files.

URDF files contain XML descriptions of basic shapes and their relationships.

system to interact with



Global Options
Background Color (0,0,0)
Fixed Frame /base_footprint

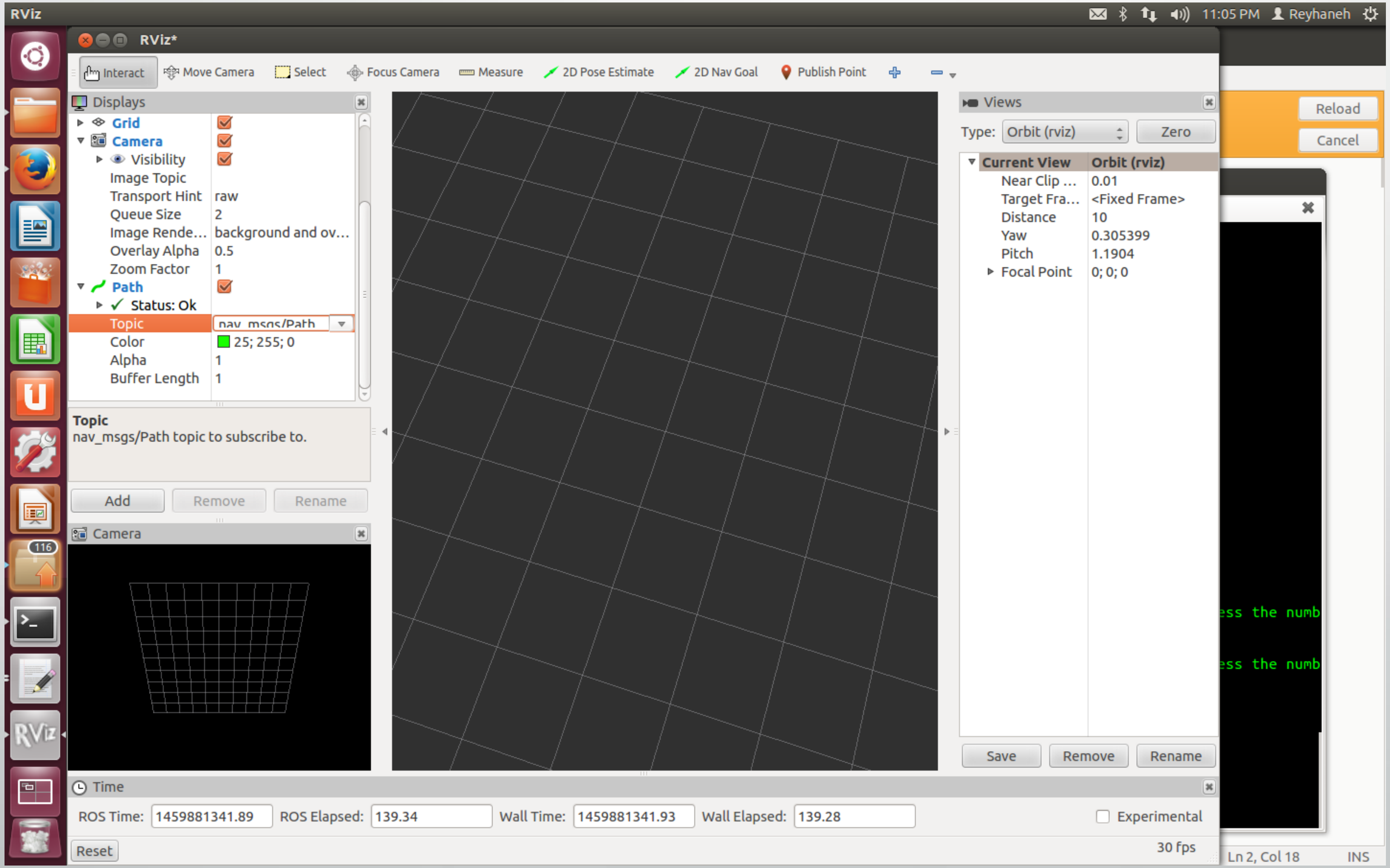
2D Nav Goal
Topic goal

2D Pose Estimate
Topic initialpose

Orbit

Wall Elapsed: 1450.883692 ROS Time: 234.922000 ROS Elapsed: 232.966000

ROS GUI TOOLS
RVIZ - 3D VISUALIZATION



Displays

Grid	☑
Camera	☑
Visibility	☑
Image Topic	
Transport Hint	raw
Queue Size	2
Image Rende…	background and ov…
Overlay Alpha	0.5
Zoom Factor	1
Path	☑
✓ Status: Ok	
Topic	nav_msgs/Path
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1

Topic
nav_msgs/Path topic to subscribe to.

Camera

Views

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip …	0.01
Target Fra…	<Fixed Frame>
Distance	10
Yaw	0.305399
Pitch	1.1904
Focal Point	0; 0; 0

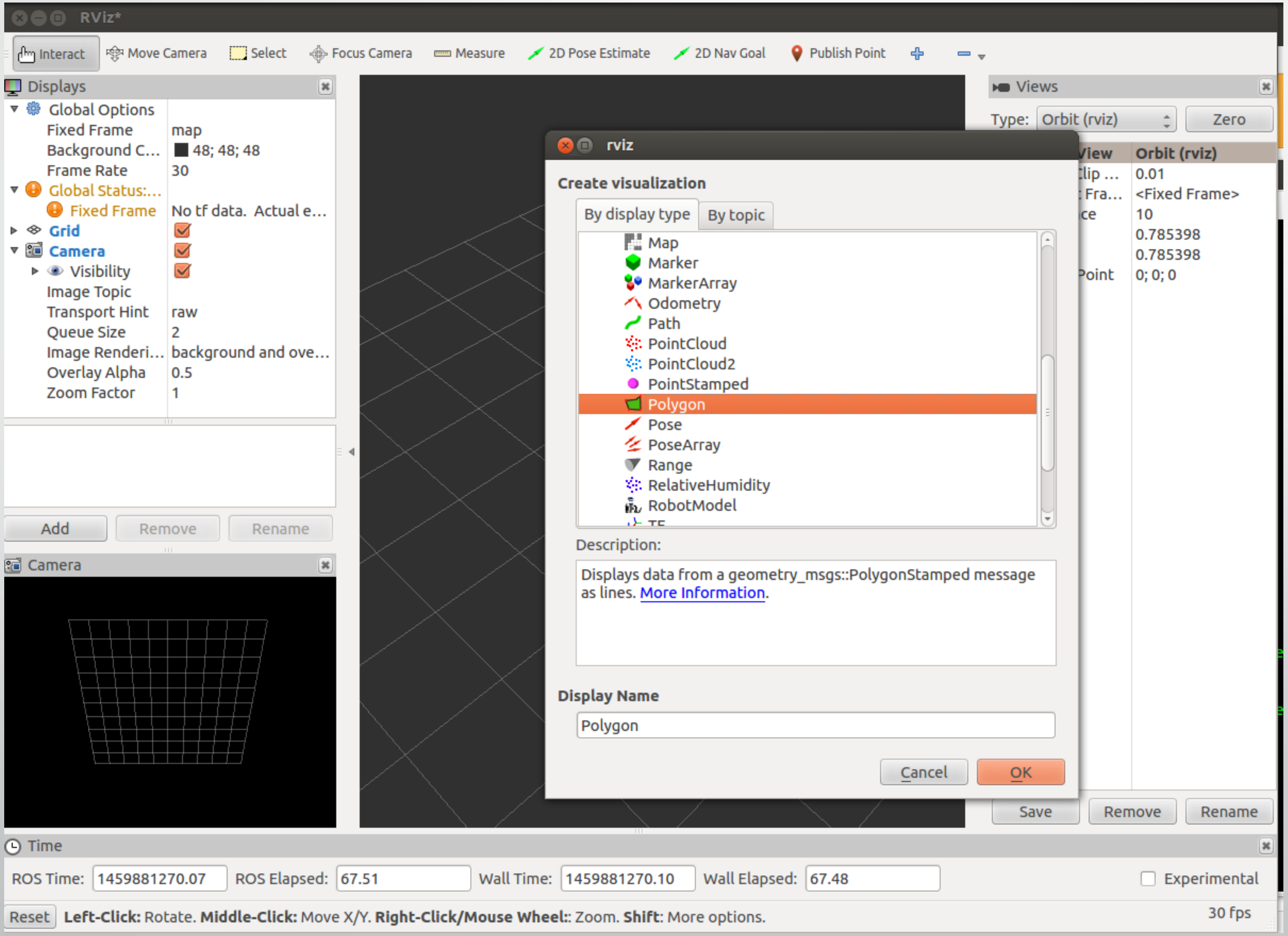
Time

ROS Time: 1459881341.89 ROS Elapsed: 139.34 Wall Time: 1459881341.93 Wall Elapsed: 139.28 □ Experimental

30 fps

ROS GUI TOOLS

RVIZ - 3D VISUALIZATION



Global Options
Fixed Frame: map
Background C…: 48; 48; 48
Frame Rate: 30
Global Status:…
Fixed Frame: No tf data. Actual e…
Grid ☑
Camera ☑
Visibility ☑
Image Topic
Transport Hint: raw
Queue Size: 2
Image Renderi…: background and ove…
Overlay Alpha: 0.5
Zoom Factor: 1

Type: Orbit (rviz)

View	Orbit (rviz)
…lip …	0.01
…Fra…	<Fixed Frame>
…ce	10
	0.785398
	0.785398
…oint	0; 0; 0

Create visualization

Description:

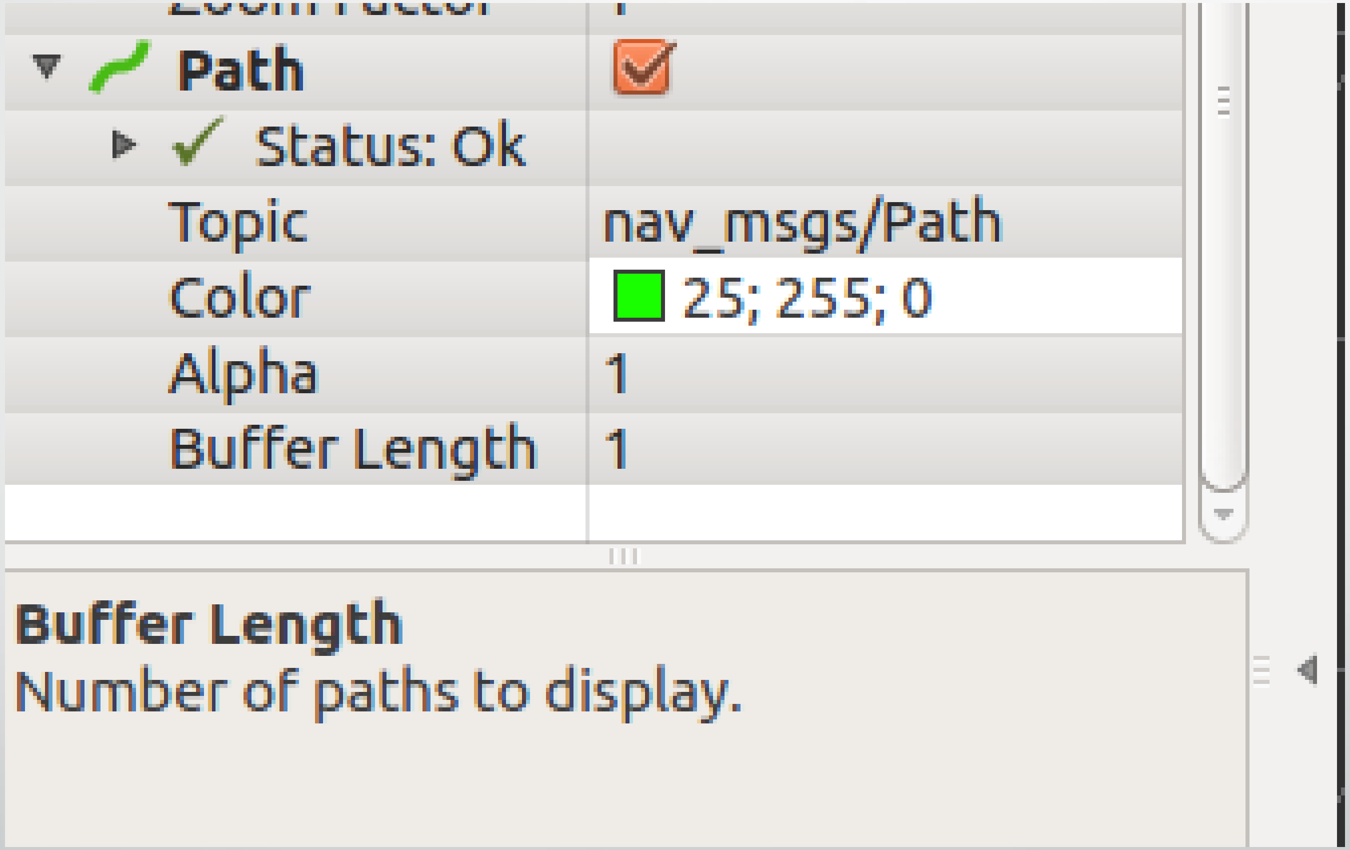
Displays data from a geometry_msgs::PolygonStamped message as lines. More Information.

Display Name: Polygon

ROS Time: 1459881270.07 ROS Elapsed: 67.51 Wall Time: 1459881270.10 Wall Elapsed: 67.48 □ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 30 fps

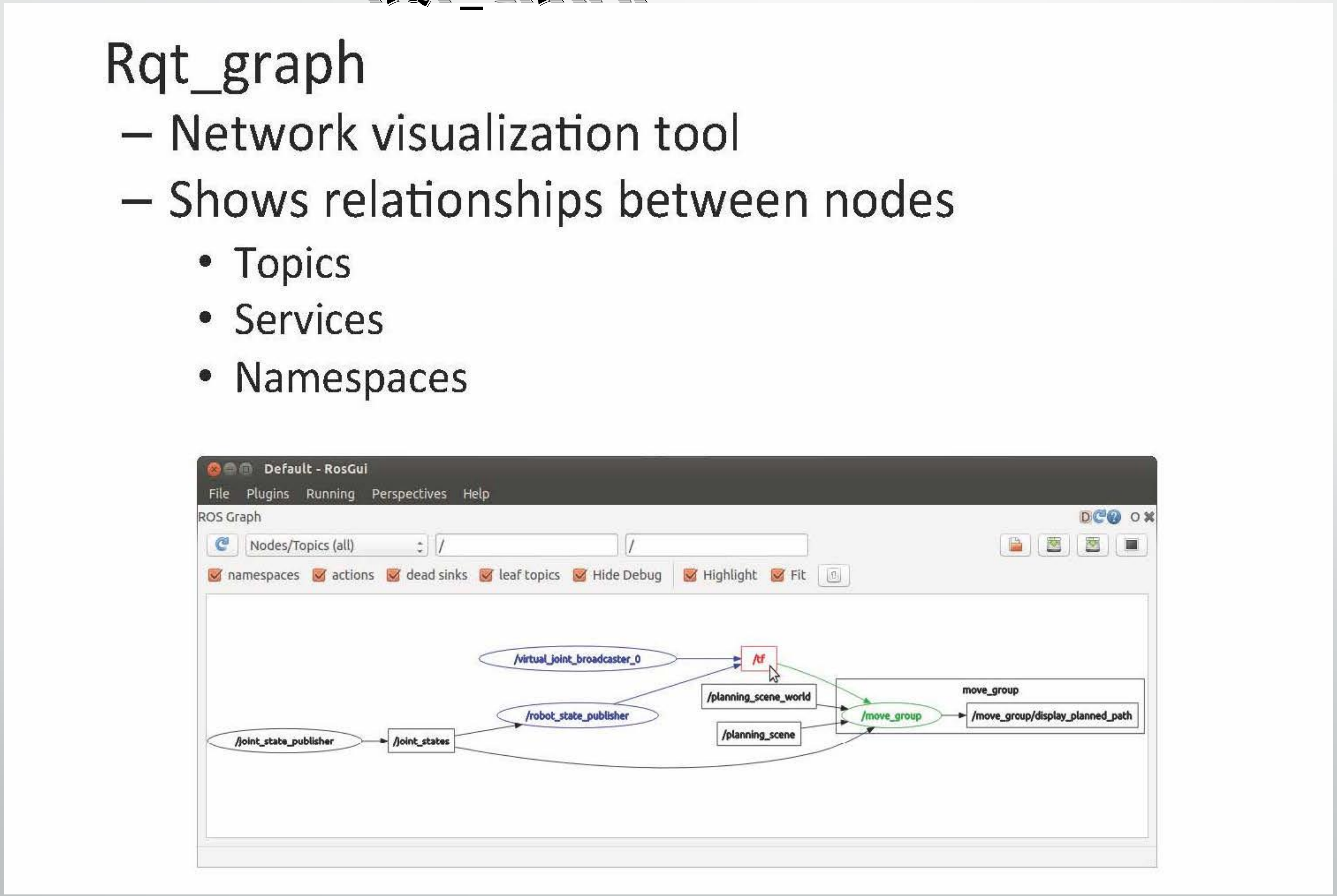
ROS GUI TOOLS
RVIZ - 3D VISUALIZATION



Path	☑
✓ Status: Ok	
Topic	nav_msgs/Path
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1

Buffer Length
Number of paths to display.

ROS GUI TOOLS
RQT_GRAPH



Rqt_graph

- Network visualization tool
- Shows relationships between nodes
  - Topics
  - Services
  - Namespaces

Default - RosGui

ROS Graph

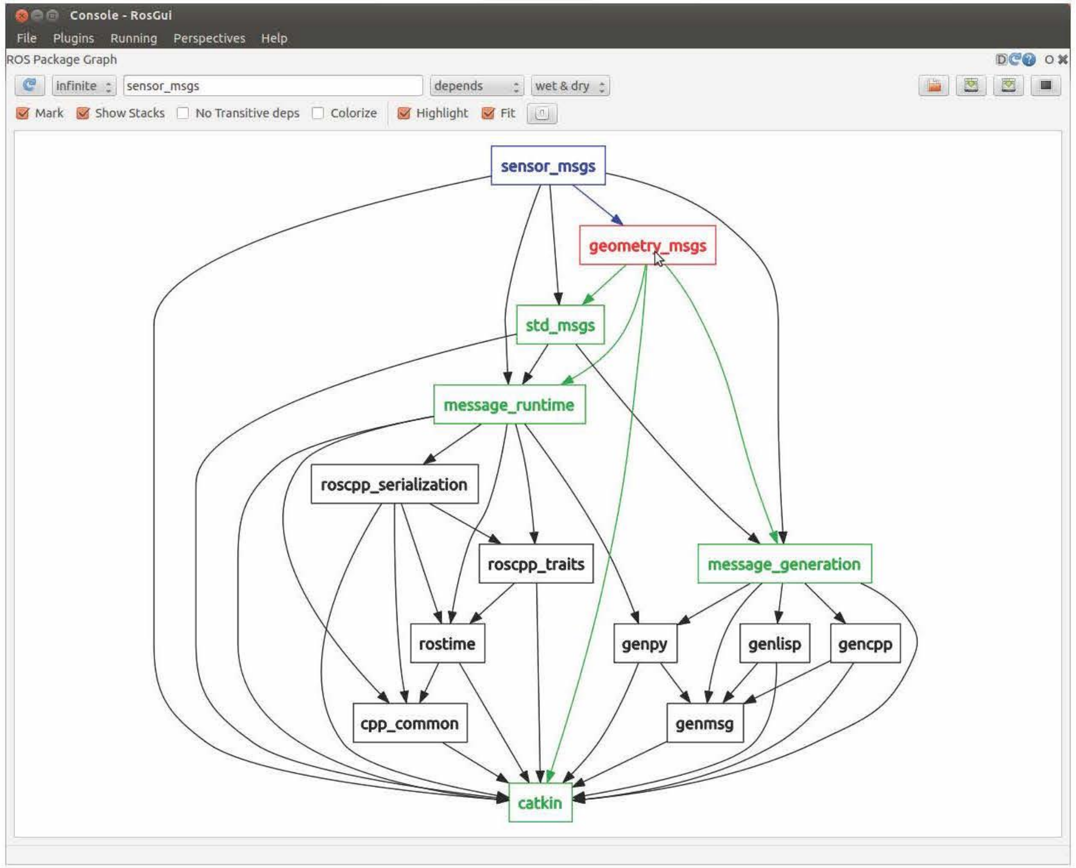
Nodes/Topics (all)

☑ namespaces ☑ actions ☑ dead sinks ☑ leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

ROS GUI TOOLS
RQT_DEP

Rqt_dep

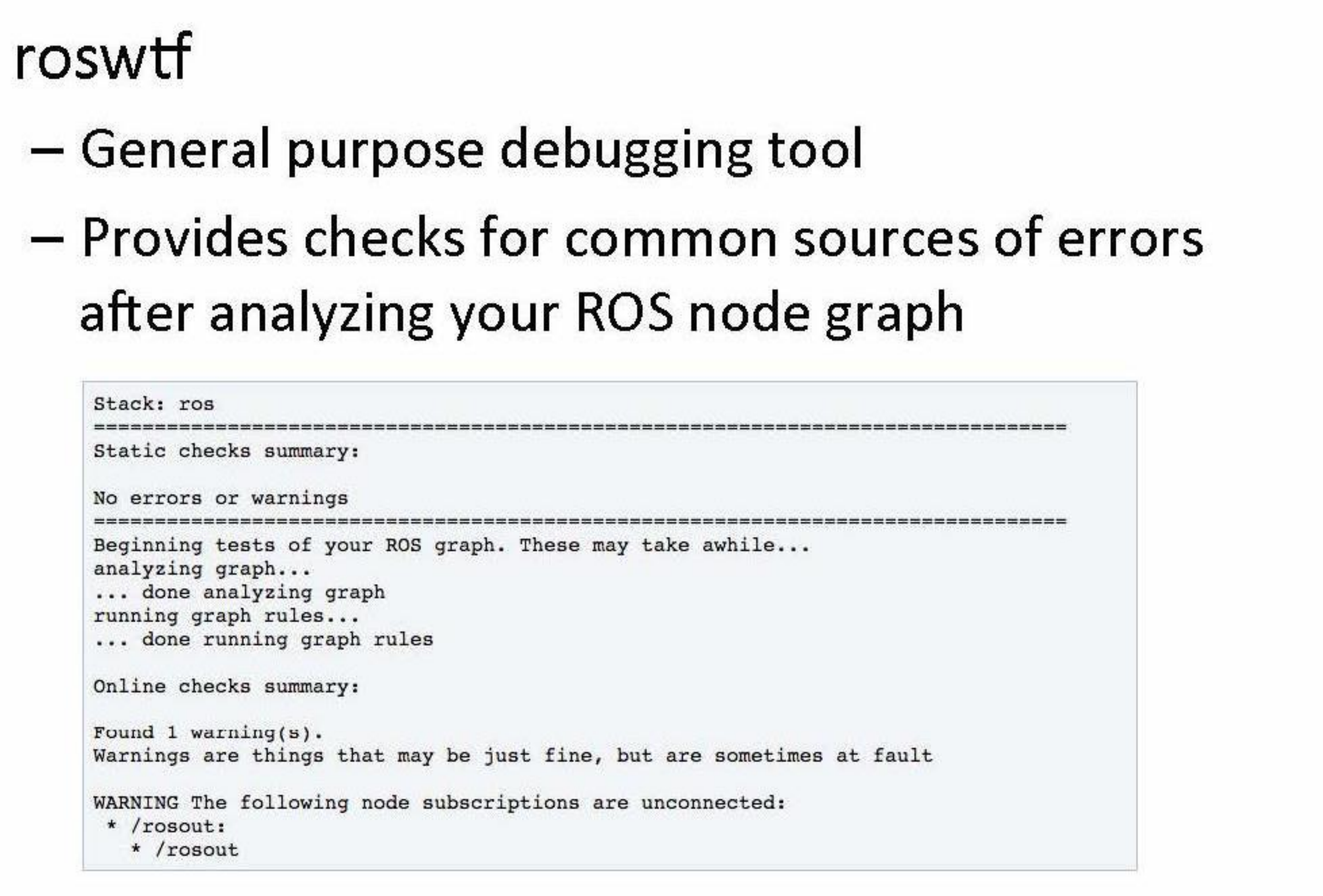
- Package dependency graph visualization tool



infinite | sensor_msgs | depends | wet & dry

☑ Mark ☑ Show Stacks □ No Transitive deps □ Colorize ☑ Highlight ☑ Fit

ROS GUI TOOLS
ROSWTF

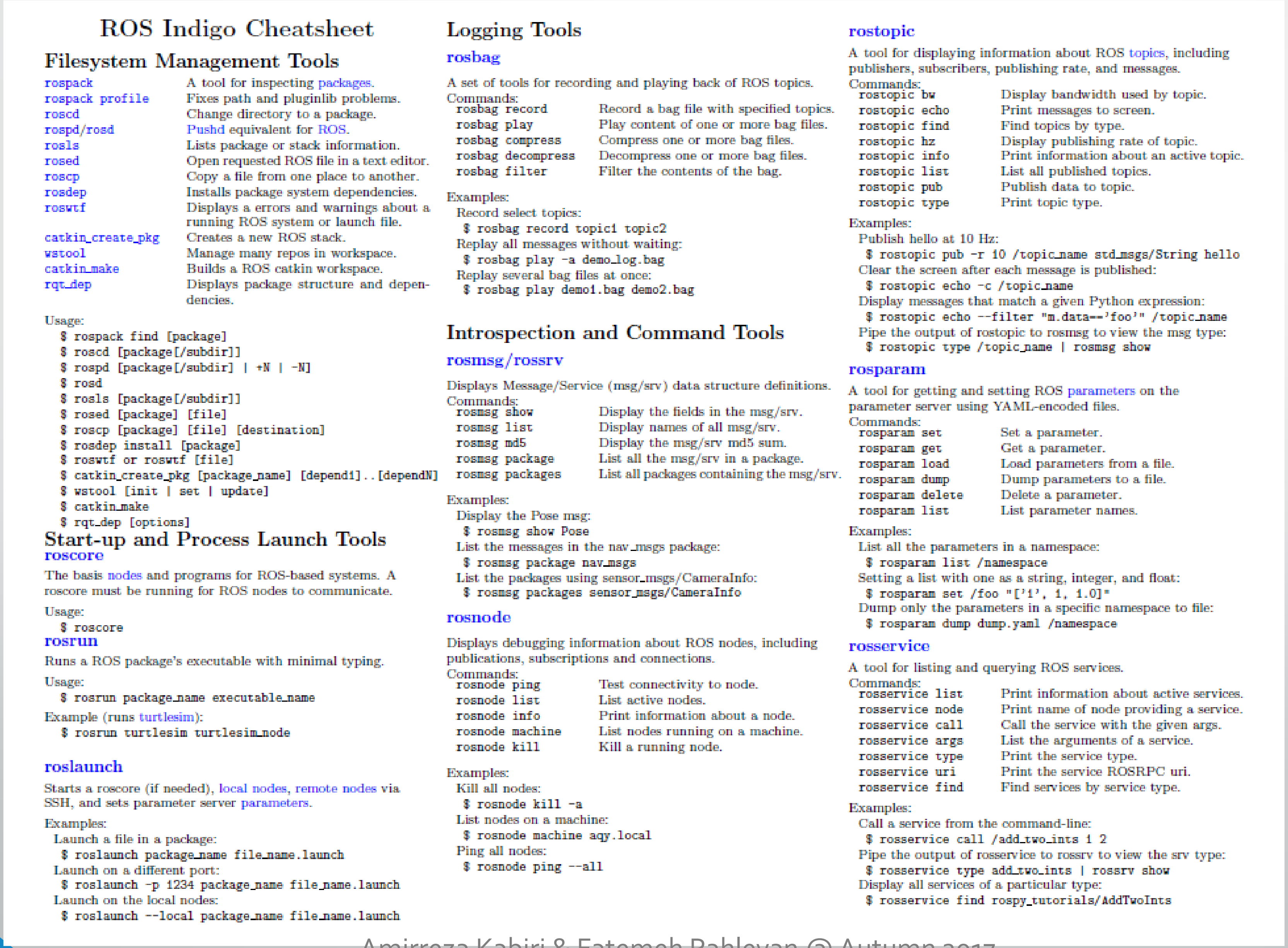


roswtf

- General purpose debugging tool
- Provides checks for common sources of errors after analyzing your ROS node graph

```
Stack: ros
================================================================================
Static checks summary:

No errors or warnings
================================================================================
Beginning tests of your ROS graph. These may take awhile...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules

Online checks summary:

Found 1 warning(s).
Warnings are things that may be just fine, but are sometimes at fault

WARNING The following node subscriptions are unconnected:
 * /rosout:
   * /rosout
```

ROS CHEATSHEET



ROS Indigo Cheatsheet

Filesystem Management Tools

rospack	A tool for inspecting packages.
rospack profile	Fixes path and pluginlib problems.
roscd	Change directory to a package.
rospd/rosd	Pushd equivalent for ROS.
rosls	Lists package or stack information.
rosed	Open requested ROS file in a text editor.
roscp	Copy a file from one place to another.
rosdep	Installs package system dependencies.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
catkin_create_pkg	Creates a new ROS stack.
wstool	Manage many repos in workspace.
catkin_make	Builds a ROS catkin workspace.
rqt_dep	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rospd [package[/subdir] | +N | -N]
$ rosd
$ rosls [package[/subdir]]
$ rosed [package] [file]
$ roscp [package] [file] [destination]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ catkin_create_pkg [package_name] [depend1]..[dependN]
$ wstool [init | set | update]
$ catkin_make
$ rqt_dep [options]
```

Start-up and Process Launch Tools

roscore

The basis nodes and programs for ROS-based systems. A roscore must be running for ROS nodes to communicate.

Usage:

```
$ roscore
```

rosrun

Runs a ROS package's executable with minimal typing.

Usage:

```
$ rosrun package_name executable_name
```

Example (runs turtlesim):

```
$ rosrun turtlesim turtlesim_node
```

roslaunch

Starts a roscore (if needed), local nodes, remote nodes via SSH, and sets parameter server parameters.

Examples:

Launch a file in a package:

```
$ roslaunch package_name file_name.launch
```

Launch on a different port:

```
$ roslaunch -p 1234 package_name file_name.launch
```

Launch on the local nodes:

```
$ roslaunch --local package_name file_name.launch
```

Logging Tools

rosbag

A set of tools for recording and playing back of ROS topics.

Commands:

rosbag record	Record a bag file with specified topics.
rosbag play	Play content of one or more bag files.
rosbag compress	Compress one or more bag files.
rosbag decompress	Decompress one or more bag files.
rosbag filter	Filter the contents of the bag.

Examples:

Record select topics:

```
$ rosbag record topic1 topic2
```

Replay all messages without waiting:

```
$ rosbag play -a demo_log.bag
```

Replay several bag files at once:

```
$ rosbag play demo1.bag demo2.bag
```

Introspection and Command Tools

rosmsg/rossrv

Displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg/srv.
rosmsg list	Display names of all msg/srv.
rosmsg md5	Display the msg/srv md5 sum.
rosmsg package	List all the msg/srv in a package.
rosmsg packages	List all packages containing the msg/srv.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in the nav_msgs package:

```
$ rosmsg package nav_msgs
```

List the packages using sensor_msgs/CameraInfo:

```
$ rosmsg packages sensor_msgs/CameraInfo
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a machine.
rosnode kill	Kill a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

rostopic

A tool for displaying information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic find	Find topics by type.
rostopic hz	Display publishing rate of topic.
rostopic info	Print information about an active topic.
rostopic list	List all published topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print name of node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

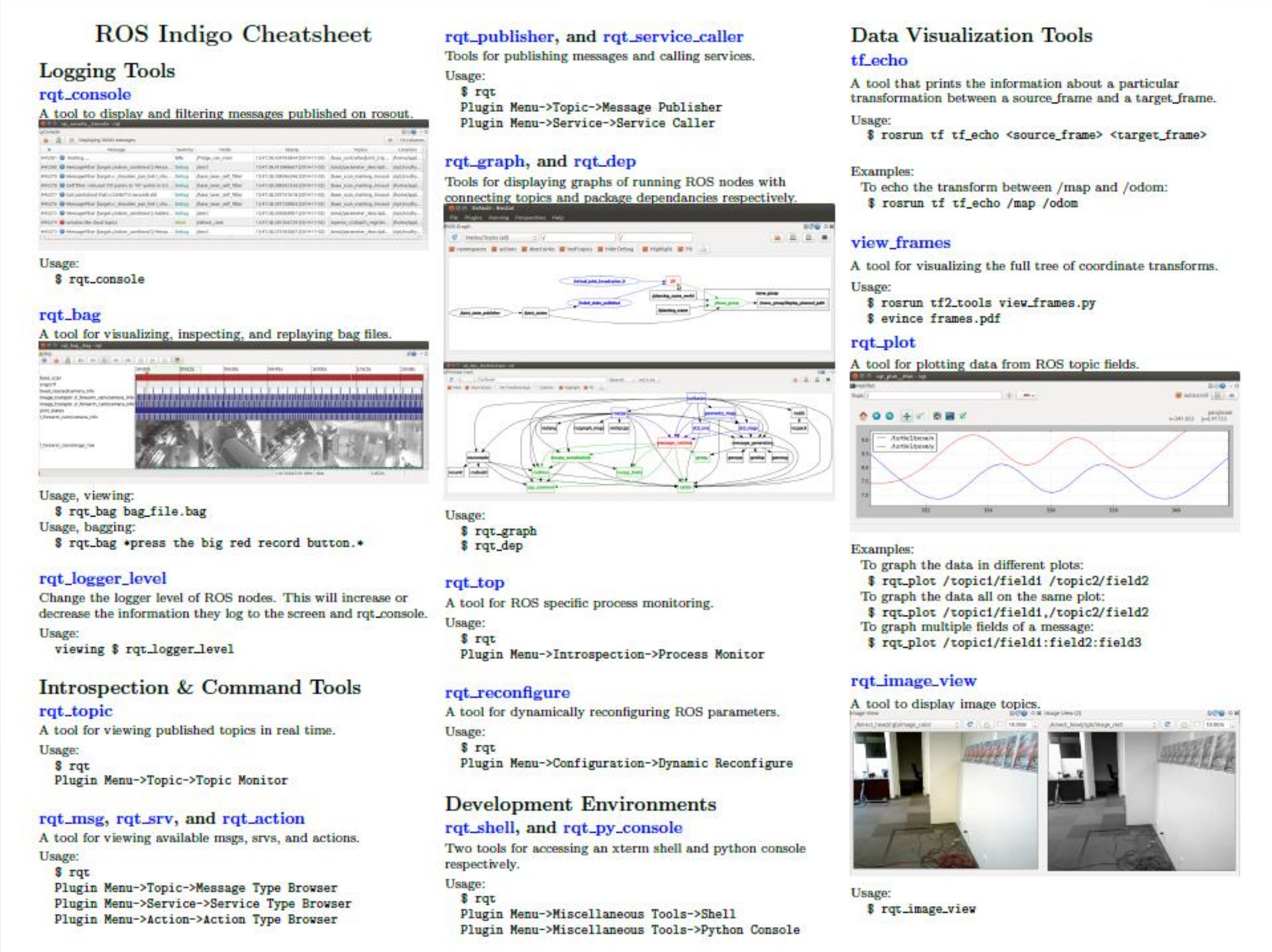
Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

ROS CHEATSHEET



ROS Indigo Cheatsheet

Logging Tools

rqt_console

A tool to display and filtering messages published on rosout.

Usage:

```
  $ rqt_console
```

rqt_bag

A tool for visualizing, inspecting, and replaying bag files.

Usage, viewing:

```
  $ rqt_bag bag_file.bag
```

Usage, bagging:

```
  $ rqt_bag *press the big red record button.*
```

rqt_logger_level

Change the logger level of ROS nodes. This will increase or decrease the information they log to the screen and rqt_console.

Usage:

```
  viewing $ rqt_logger_level
```

Introspection & Command Tools

rqt_topic

A tool for viewing published topics in real time.

Usage:

```
  $ rqt
  Plugin Menu->Topic->Topic Monitor
```

rqt_msg, rqt_srv, and rqt_action

A tool for viewing available msgs, srvs, and actions.

Usage:

```
  $ rqt
  Plugin Menu->Topic->Message Type Browser
  Plugin Menu->Service->Service Type Browser
  Plugin Menu->Action->Action Type Browser
```

rqt_publisher, and rqt_service_caller

Tools for publishing messages and calling services.

Usage:

```
  $ rqt
  Plugin Menu->Topic->Message Publisher
  Plugin Menu->Service->Service Caller
```

rqt_graph, and rqt_dep

Tools for displaying graphs of running ROS nodes with connecting topics and package dependancies respectively.

Usage:

```
  $ rqt_graph
  $ rqt_dep
```

rqt_top

A tool for ROS specific process monitoring.

Usage:

```
  $ rqt
  Plugin Menu->Introspection->Process Monitor
```

rqt_reconfigure

A tool for dynamically reconfiguring ROS parameters.

Usage:

```
  $ rqt
  Plugin Menu->Configuration->Dynamic Reconfigure
```

Development Environments

rqt_shell, and rqt_py_console

Two tools for accessing an xterm shell and python console respectively.

Usage:

```
  $ rqt
  Plugin Menu->Miscellaneous Tools->Shell
  Plugin Menu->Miscellaneous Tools->Python Console
```

Data Visualization Tools

tf_echo

A tool that prints the information about a particular transformation between a source_frame and a target_frame.

Usage:

```
  $ rosrun tf tf_echo <source_frame> <target_frame>
```

Examples:

To echo the transform between /map and /odom:

```
  $ rosrun tf tf_echo /map /odom
```

view_frames

A tool for visualizing the full tree of coordinate transforms.

Usage:

```
  $ rosrun tf2_tools view_frames.py
  $ evince frames.pdf
```

rqt_plot

A tool for plotting data from ROS topic fields.

Examples:

To graph the data in different plots:

```
  $ rqt_plot /topic1/field1 /topic2/field2
```

To graph the data all on the same plot:

```
  $ rqt_plot /topic1/field1,/topic2/field2
```

To graph multiple fields of a message:

```
  $ rqt_plot /topic1/field1:field2:field3
```

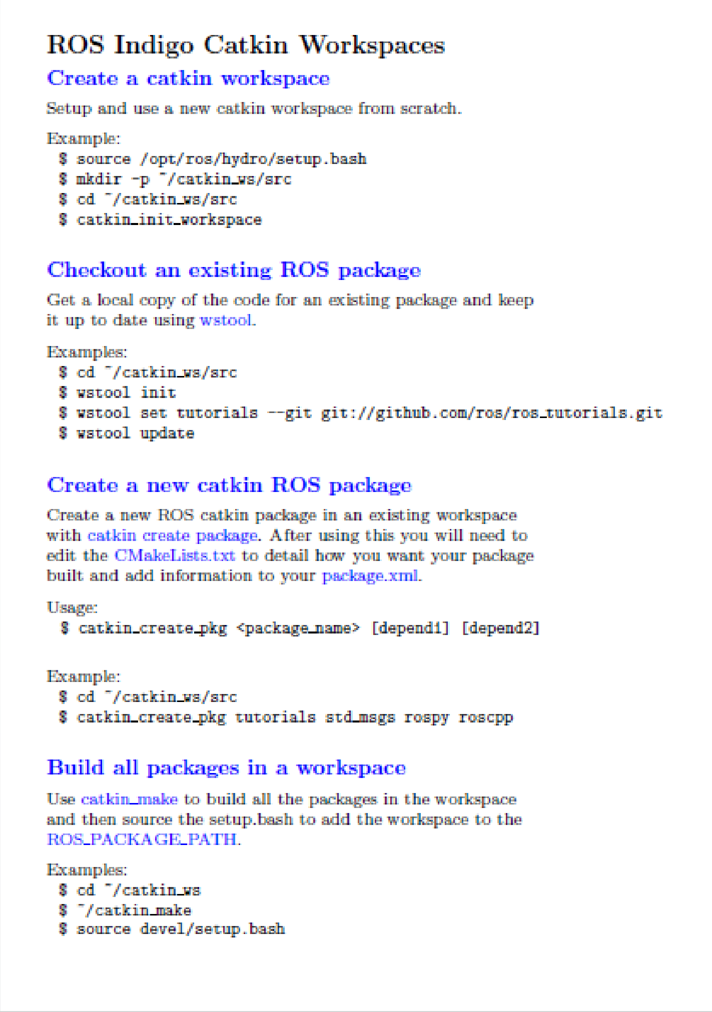
rqt_image_view

A tool to display image topics.

Usage:

```
  $ rqt_image_view
```

ROS CHEATSHEET



ROS Indigo Catkin Workspaces

Create a catkin workspace

Setup and use a new catkin workspace from scratch.

Example:

```
$ source /opt/ros/hydro/setup.bash
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Checkout an existing ROS package

Get a local copy of the code for an existing package and keep it up to date using wstool.

Examples:

```
$ cd ~/catkin_ws/src
$ wstool init
$ wstool set tutorials --git git://github.com/ros/ros_tutorials.git
$ wstool update
```

Create a new catkin ROS package

Create a new ROS catkin package in an existing workspace with catkin create package. After using this you will need to edit the CMakeLists.txt to detail how you want your package built and add information to your package.xml.

Usage:

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

Example:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorials std_msgs rospy roscpp
```

Build all packages in a workspace

Use catkin_make to build all the packages in the workspace and then source the setup.bash to add the workspace to the ROS_PACKAGE_PATH.

Examples:

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