

ROS Crash-Course, Part I

Introduction to ROS distribution, build system and infrastructure

Jonathan Bohren

*With some information and figures adapted from <http://www.ros.org>
and the COTESYS-ROS School 2010 presentation given by Radu Rusu*



LABORATORY FOR

**Computational
Sensing + Robotics**

THE JOHNS HOPKINS UNIVERSITY

Outline

- ① Introduction
 - High-Level
 - The ROS Ecosystem
 - ROS Community
- ② ROS as a Communication Platform
 - The ROS Network Graph
 - Running and Connecting Nodes
 - Analyzing the System at Runtime
- ③ ROS as a Build Platform
 - Distribution & Package Management System
 - Build System

Outline (revisited)

① Introduction

High-Level

The ROS Ecosystem

ROS Community

② ROS as a Communication Platform

The ROS Network Graph

Running and Connecting Nodes

Analyzing the System at Runtime

③ ROS as a Build Platform

Distribution & Package Management System

Build System

What is ROS?

More than just middleware

- A “meta” operating system for robots
- A collection of packaging, software building tools
- An architecture for distributed* inter-process/inter-machine communication and configuration
- Development tools for system runtime and data analysis
- Open-source under permissive BSD licenses (*ros core libraries*)
- A language-independent architecture (c++, python, lisp, java, *and more*)
- A *scalable* platform (ARM CPUS to Xeon Clusters)

With the intent to enable researchers to rapidly develop new robotic systems without having to “reinvent the wheel” through use of standard tools and interfaces.

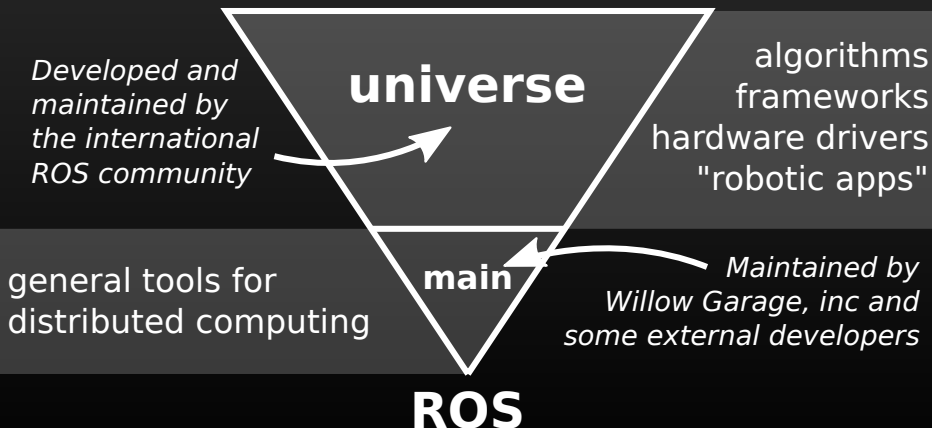
What is ROS *not*?

No confusion

- An *actual* operating system
- A programming language
- A programming environment / IDE
- A hard real-time architecture*

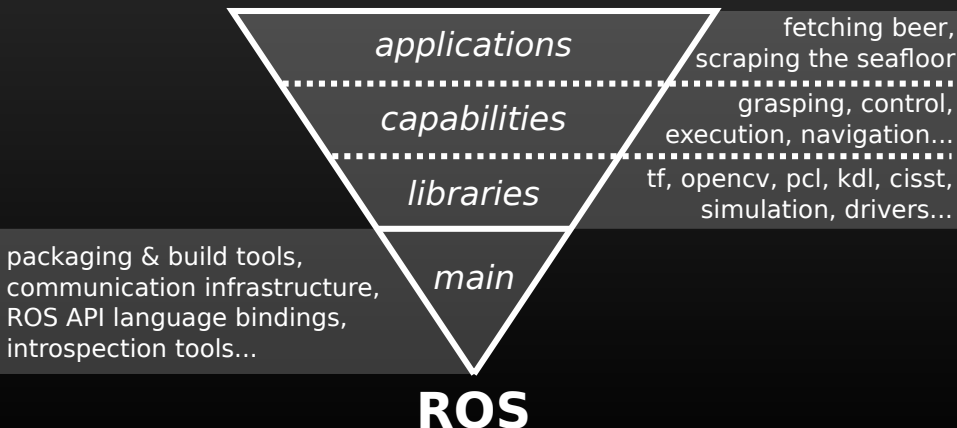
What does ROS get you?

All levels of development



What does ROS get you?

All levels of development



Outline (revisited)

- 1 Introduction
 - High-Level
 - The ROS Ecosystem
 - ROS Community
- 2 ROS as a Communication Platform
 - The ROS Network Graph
 - Running and Connecting Nodes
 - Analyzing the System at Runtime
- 3 ROS as a Build Platform
 - Distribution & Package Management System
 - Build System



ROS Core

Where it all comes together

The ROS core is a set of the only three programs that are necessary for the ROS runtime. They include:

- **ROS Master**

- A centralized XML-RPC server
- Negotiates communication connections
- Registers and looks up names for ROS graph resources

- **Parameter Server**

Stores persistent configuration parameters and other arbitrary data

- **roscout**

Essentially a network-based `stdout` for human-readable messages



ROS “Graph” Abstraction

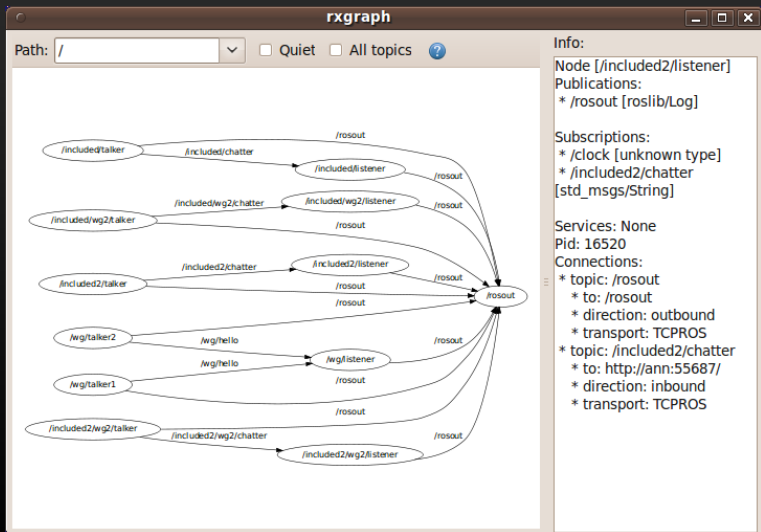
Named network resources

The ROS runtime designates several classes of named ROS *graph resources*. These resources can exist in namespaces to reduce naming collisions, and fall into the following categories:

- **nodes**
Represent processes distributed across the ROS network. A ROS node is a source and sink for data that is sent over the ROS network.
- **parameters**
Persistent (while the core is running) data such as configuration & initialization settings, stored on the parameter server.
- **topics**
Asynchronous many-to-many communication streams.
- **services**
Synchronous one-to-many network-based functions.

ROS “Graph”

rxgraph: communication network visualization





Creating and Running ROS Nodes

Distributing computation with ROS

ROS provides a mechanism for simultaneously configuring and *launching* collections of ROS nodes. This is done with lightweight xml files and the `roslaunch` program.

Launch files enable users to:

- Associate a set of parameters and nodes with a single file
- Hierarchically compose collections of other launch files
- Automatically re-spawn nodes if they crash
- Change node names, namespaces, topics, and other resource names *without* recompiling
- Easily distribute nodes across multiple machines
- Attach `gdb` to a series of nodes



ROS Communication Protocols

Connecting nodes over the network

ROS supports a growing number of communication capabilities that enable distributing computation in a robotic system. These capabilities are currently built entirely on two high-level communication APIs:

- **ROS Topics**

- Asynchronous “stream-like” communication
- TCP/IP or UDP Transport
- Strongly-typed (ROS .msg spec)
- Can have one or more *publishers*
- Can have one or more *subscribers*

ROS Communication Protocols

Connecting nodes over the network

ROS supports a growing number of communication capabilities that enable distributing computation in a robotic system. These capabilities are currently built entirely on two high-level communication APIs:

■ ROS Topics

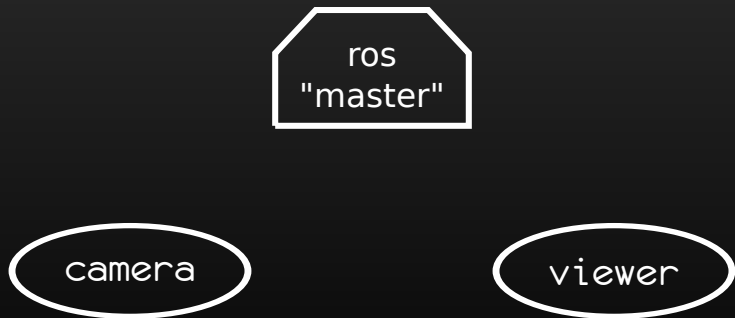
- Asynchronous “stream-like” communication
- TCP/IP or UDP Transport
- Strongly-typed (ROS .msg spec)
- Can have one or more *publishers*
- Can have one or more *subscribers*

■ ROS Services

- Synchronous “function-call-like” communication
- TCP/IP or UDP Transport
- Strongly-typed (ROS .srv spec)
- Can have only one *server*
- Can have one or more *clients*

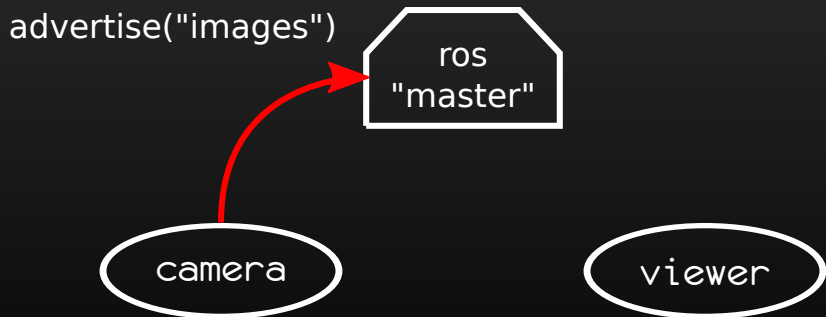
Asynchronous Distributed* Communication

ROS TCP Topics



Asynchronous Distributed* Communication

ROS TCP Topics



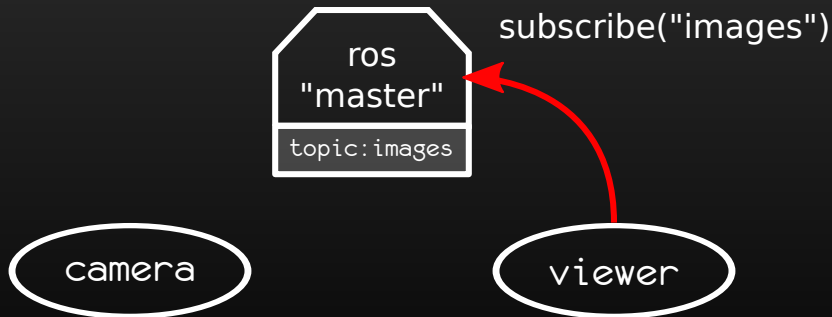
Asynchronous Distributed* Communication

ROS TCP Topics



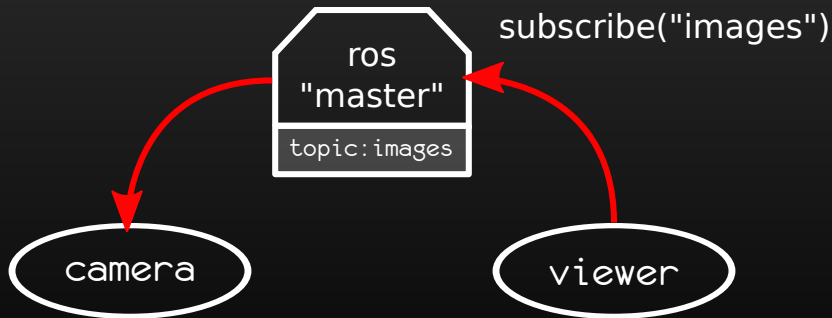
Asynchronous Distributed* Communication

ROS TCP Topics



Asynchronous Distributed* Communication

ROS TCP Topics



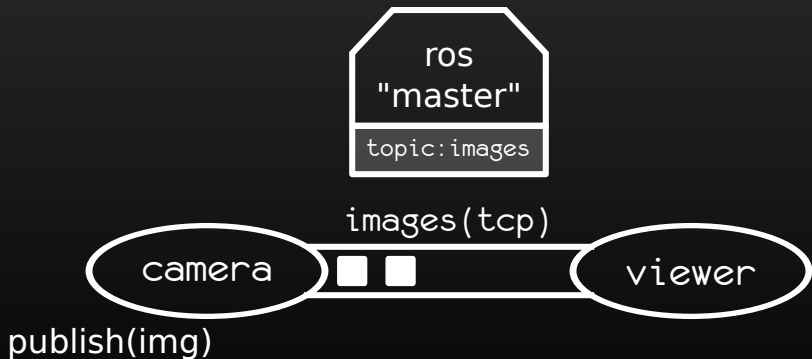
Asynchronous Distributed* Communication

ROS TCP Topics



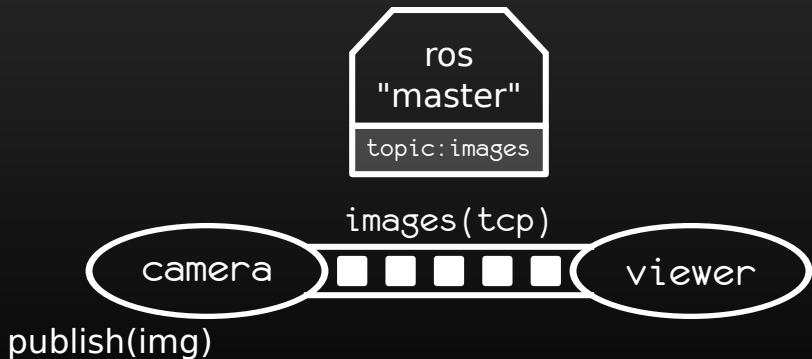
Asynchronous Distributed* Communication

ROS TCP Topics



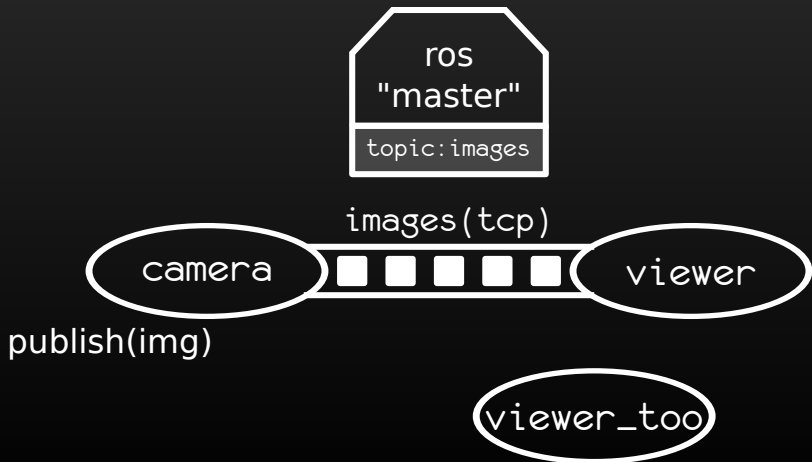
Asynchronous Distributed* Communication

ROS TCP Topics



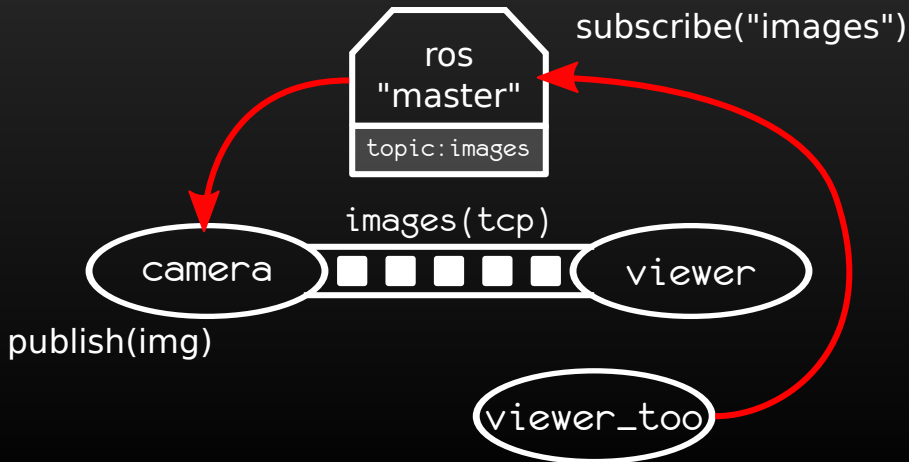
Asynchronous Distributed* Communication

ROS TCP Topics



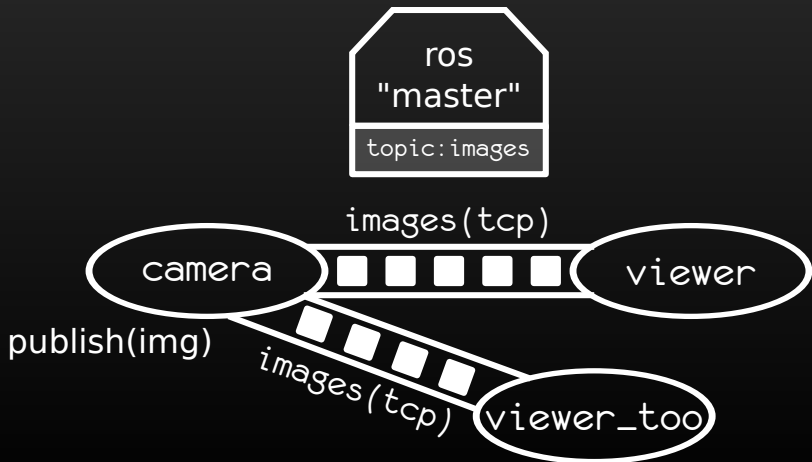
Asynchronous Distributed* Communication

ROS TCP Topics



Asynchronous Distributed* Communication

ROS TCP Topics



rosout Messaging

stdout on steroids

ROS provides mechanisms in all languages for specifying different *levels* of human-readable log messages. The five default levels are:

- fatal
- error
- warn
- info
- debug

These enable a user to “add printf’s” to their code for debugging, and selectively enable and disable them *at runtime* without a large performance hit. For example, useful debug messages that might be necessary to diagnose a problem could be left in the code and re-enabled at a critical time.

ROS Graph Introspection

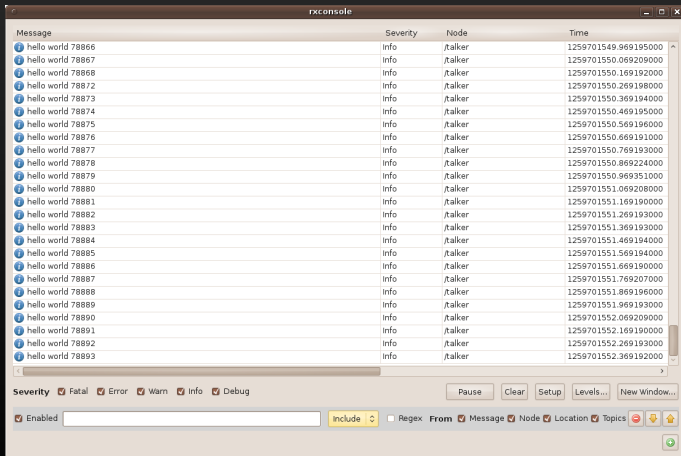
No more wireshark

ROS provides several tools for analyzing the data flowing over ROS communication resources:

- **rostopic**
Gives a user information about a node: publications, subscriptions, etc
- **rostopic**
Gives datarate, actual data, publishers, subscribes
- **rosservice**
Enables a user to call a ROS Service from the command line
- **roswtf** (wire trouble finder)
Diagnoses problems with a ROS network

ROS GUI Tools

There are lots. . .



The screenshot shows the rxconsole window with a table of messages. The table has four columns: Message, Severity, Node, and Time. The messages are all 'hello world' with varying IDs, all at 'info' severity, and all from the 'talker' node. The time values range from 1259701549.969195000 to 1259701552.369192000.

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

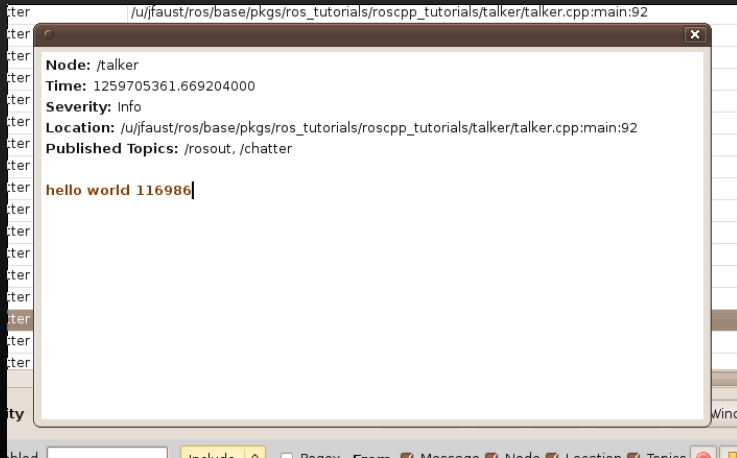
Severity: Fatal Error Warn Info Debug

Buttons: Pause, Clear, Setup, Levels..., New Window...

Filter: Enabled [] Include [] Regexp From Message Node Location Topics

ROS GUI Tools

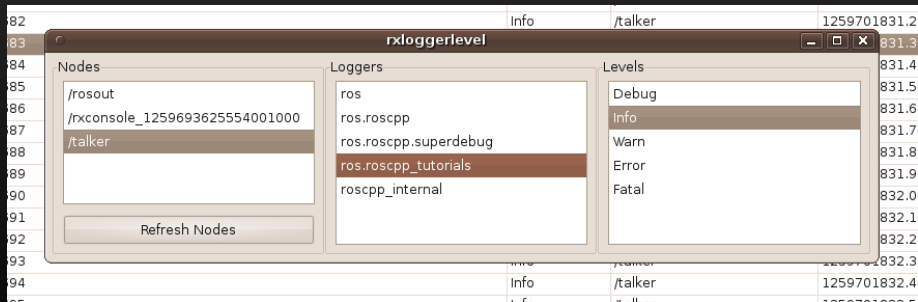
There are lots. . .

A screenshot of a ROS GUI tool window. The window title is '/u/jfaust/ros/base/pkgs/ros_tutorials/roscpp_tutorials/talker/talker.cpp:main:92'. The window content shows the following information:
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

Below this information, the message content is displayed: **hello world 116986**. The window has a standard OS-style title bar with a close button (X) in the top right corner. The background shows a blurred terminal window with the text 'ter' repeated vertically.

ROS GUI Tools

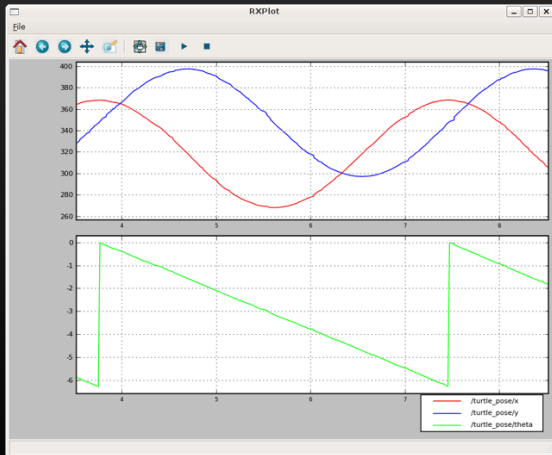
There are lots. . .



The screenshot shows the `rloggerlevel` GUI tool. It has three main sections: Nodes, Loggers, and Levels. The Nodes section lists `/rosout`, `/rxconsole_1259693625554001000`, and `/talker`. The Loggers section lists `ros`, `ros.roscpp`, `ros.roscpp.superdebug`, `ros.roscpp_tutorials`, and `roscpp_internal`. The Levels section lists `Debug`, `Info`, `Warn`, `Error`, and `Fatal`. A `Refresh Nodes` button is located at the bottom of the Nodes section. The background shows a terminal window with ROS log output.

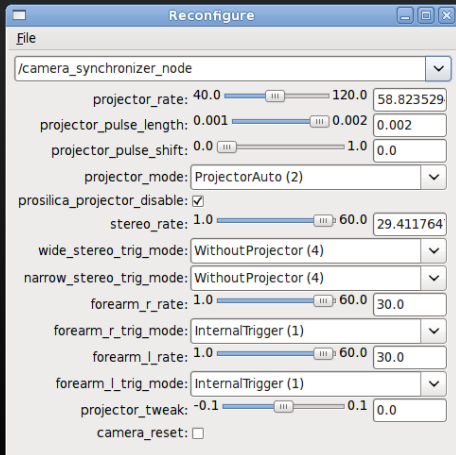
ROS GUI Tools

There are lots. . .



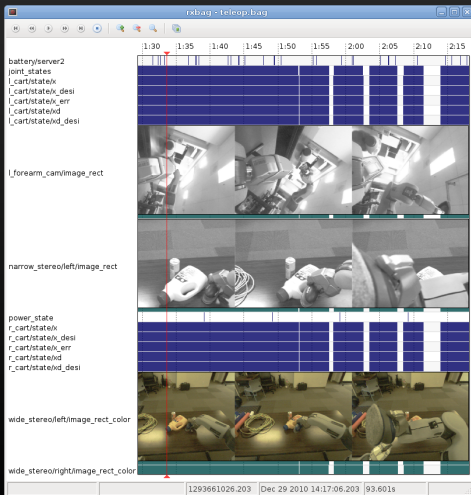
ROS GUI Tools

There are lots. . .



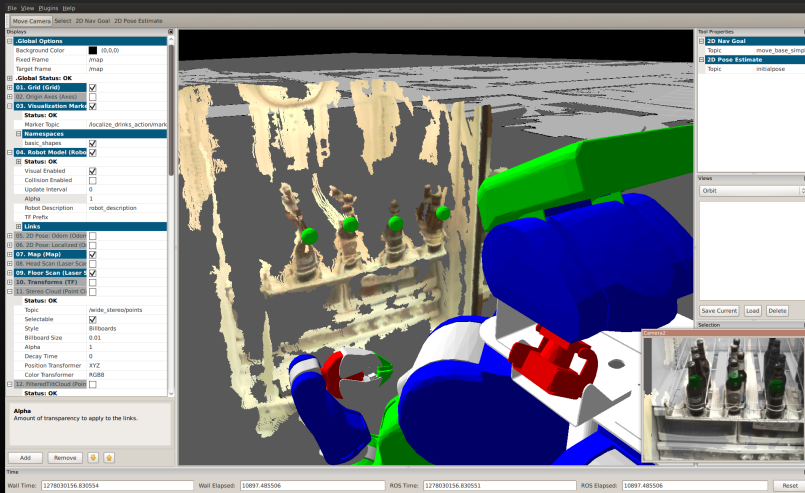
ROS GUI Tools

There are lots. . .



rviz - 3D Visualization

Modular state and sensor visualization



The screenshot displays the rviz 3D visualization interface. The main window shows a 3D scene with a robot model (blue and red) and a 2D map (yellow and black). The interface is divided into several panels:

- Global Options:** Background Color (0.0, 0.0, 0.0), Fixed Frame (/map), Target Frame (/map).
- Global Status: OK**
- 01. Grid (Grid):** (Status: OK)
- 02. Visualization Markers:** (Status: OK)
- Namespaces:**
 - base_controller (Status: OK)
 - 04. Robot Model (Robot): (Status: OK)
- 05. 2D Pose Estimate (Pose):** (Status: OK)
- 06. 2D Nav Goal (Goal):** (Status: OK)
- 07. Map (Map):** (Status: OK)
- 08. Head Scan (Laser Scan):** (Status: OK)
- 09. Floor Scan (Laser):** (Status: OK)
- 10. Transforms (TF):** (Status: OK)
- 11. Stereo Display (Pose):** (Status: OK)
- 12. Filtered Cloud (Point Cloud):** (Status: OK)

The **Views** panel shows the current view is **Orbit**. The **Topic Priorities** panel shows the following topics and their priorities:

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

The **Time** panel at the bottom shows the following information:

- Wall Time: 1278030156.830554
- Wall Elapsed: 10897.485506
- ROS Time: 1278030156.830553
- ROS Elapsed: 10897.485506

Outline (revisited)

- 1 Introduction
 - High-Level
 - The ROS Ecosystem
 - ROS Community
- 2 ROS as a Communication Platform
 - The ROS Network Graph
 - Running and Connecting Nodes
 - Analyzing the System at Runtime
- 3 ROS as a Build Platform
 - Distribution & Package Management System
 - Build System



ROS Stacks & Packages

How to organize code in a ROS ecosystem

ROS code is grouped at two different levels:

- **Packages**

A named collection of software that is built and treated as an atomic dependency in the ROS build system.

- **Stacks**

A named collection of packages for distribution.



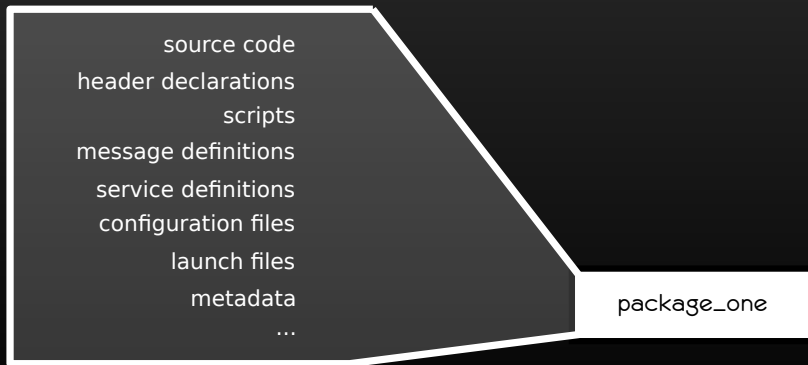
ROS Distributions

Delivering ROS packages to the masses

- source code
- header declarations
- scripts
- message definitions
- service definitions
- configuration files
- launch files
- metadata
- ...

ROS Distributions

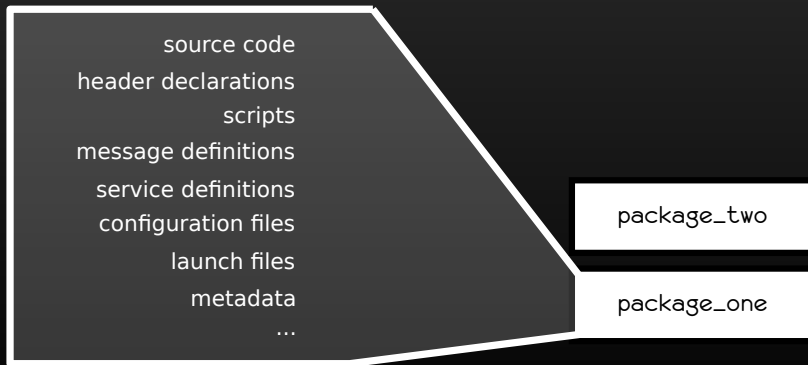
Delivering ROS packages to the masses



"package"

ROS Distributions

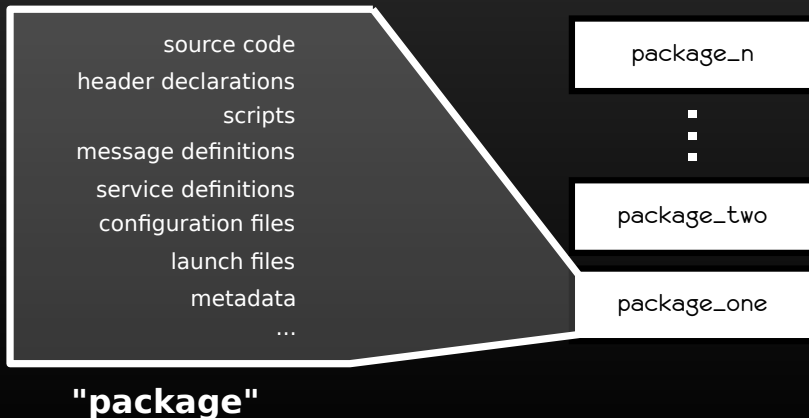
Delivering ROS packages to the masses



"package"

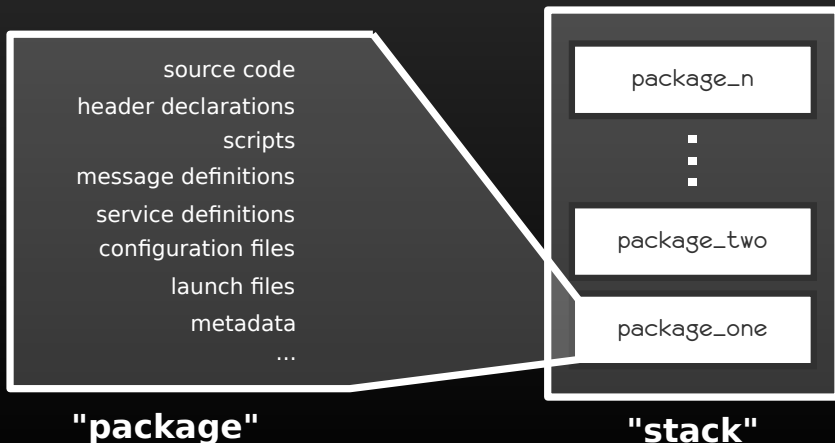
ROS Distributions

Delivering ROS packages to the masses



ROS Distributions

Delivering ROS packages to the masses





ROS Distributions

Delivering ROS packages to the masses

ROS Distributions

Delivering ROS packages to the masses



stack_a-0.4.0

ROS Distributions

Delivering ROS packages to the masses



stack_a-0.4.0



stack_b-1.0.2

ROS Distributions

Delivering ROS packages to the masses



stack_a-0.4.0



stack_b-1.0.2

...



stack_c-0.2.1

ROS Distributions

Delivering ROS packages to the masses



"distribution"

ROS Distributions

Delivering ROS packages to the masses



"distribution"

*ROS "Box Turtle"
March 2, 2010*

*ROS "C-Turtle"
August 2, 2010*

*ROS "Diamondback"
March 2, 2011*

ROS Distributions

Delivering ROS packages to the masses



"distribution"

*ROS "Box Turtle"
March 2, 2010*

*ROS "C-Turtle"
August 2, 2010*

*ROS "Diamondback"
March 2, 2011*

***Future: ROS "Electric Emys"
August, 2011***

ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

- `manifest.xml`

ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

- `manifest.xml`
 - Contains package metadata (author, license, url, etc)

ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

- `manifest.xml`
 - Contains package metadata (author, license, url, etc)
 - Specifies *system* and *package* dependencies

ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

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

ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

- `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, custom build flags, etc)



ROS Meta-Filesystem

Increasing codebase flexibility

The minimal representation of a ROS package is a directory in the `$ROS_PACKAGE_PATH` which contains a single file:

- `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, custom build flags, etc)

- `Makefile`

Just a proxy to build *this* package

ROS Meta-Filesystem

This meta-filesystem allows ROS (`rospack`, specifically) to locate *any* package in the designated path, be it at compile time or runtime.

ROS Meta-Filesystem

This meta-filesystem allows ROS (`rospack`, specifically) to locate *any* package in the designated path, be it at compile time or runtime.

Since ROS can find any package at any time, it enables packages to be moved around in the actual filesystem and greater codebase flexibility.

Building Code with ROS

Easier CMake

While ROS uses CMake (www.cmake.org) internally to compile and link code, the ROS build system it adds several useful features that make it easier to build ROS code.

Building Code with ROS

Easier CMake

While ROS uses CMake (www.cmake.org) internally to compile and link code, the ROS build system it adds several useful features that make it easier to build ROS code.

- `roscpp` *pulls* compile and linker flags out of ROS package manifests, so compiling against other ROS code is as easy as specifying the package name

Building Code with ROS

Easier CMake

While ROS uses CMake (www.cmake.org) internally to compile and link code, the ROS build system it adds several useful features that make it easier to build ROS code.

- `roscpp` *pulls* compile and linker flags out of ROS package manifests, so compiling against other ROS code is as easy as specifying the package name
- `roscpp` can be used to install system dependencies specified in a package's `manifest.xml`

Building Code with ROS

Easier CMake

While ROS uses CMake (www.cmake.org) internally to compile and link code, the ROS build system it adds several useful features that make it easier to build ROS code.

- `roscpp` *pulls* compile and linker flags out of ROS package manifests, so compiling against other ROS code is as easy as specifying the package name
- `roscpp` can be used to install system dependencies specified in a package's `manifest.xml`
- ROS CMake macros enable rapid building of executables, libraries, and automated regression tests

Thank you!

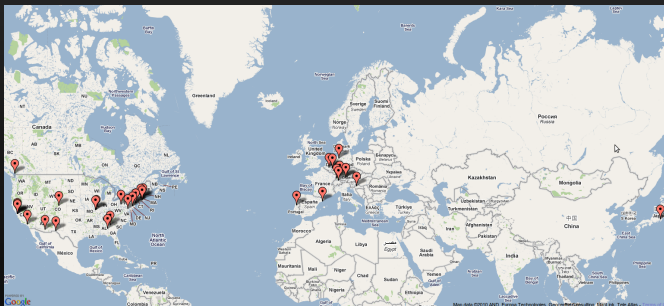
Thank you!

Now proceed to the ROS Beginner Tutorials!

<http://www.ros.org/wiki/ROS/Tutorials>

The ROS Community

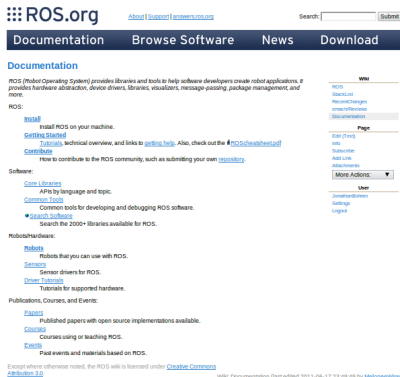
Researchers using common tools to enable collaboration



79 Institutional ROS Repositories, all over the world (July, 2011)
(jhu-lcsr-ros-pkg would make 80)

www.ros.org - The ROS Hub

A centralized location for ROS users and developers



The screenshot shows the ROS.org website interface. At the top, there is a search bar and a navigation bar with links for Documentation, Browse Software, News, and Download. The main content area is titled "Documentation" and provides an overview of ROS (Robot Operating System) as a framework for building robot applications. It lists various resources such as installation guides, getting started tutorials, and a repository for software. On the right side, there is a sidebar with a "Wiki" section containing links like "About ROS", "Getting Started", and "FAQ". Below the sidebar, there are sections for "User" and "More Actions".

ROS.org [About](#) | [Support](#) | [Answers.ros.org](#) Search:

Documentation Browse Software News Download

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:

- [Install](#) - Install ROS on your machine.
- [Getting Started](#) - Tutorials, technical overview, and links to [getting help](#). Also, check out the [#ROSChannel](#) on IRC!
- [Contribute](#) - How to contribute to the ROS community, such as submitting your own [repository](#).

Software:

- [Core Libraries](#) - APIs by language and topic.
- [Common Tools](#) - Common tools for developing and debugging ROS software.
- [Search Software](#) - Search the 2000+ libraries available for ROS.

Robots/hardware:

- [Robots](#) - Robots that you can use with ROS.
- [Sensors](#) - Sensor drivers for ROS.
- [Driver Tutorials](#) - Tutorials for supported hardware.

Publications, Courses, and Events:

- [Papers](#) - Published papers with open source implementations available.
- [Courses](#) - Courses using or teaching ROS.
- [Events](#) - Past events and materials based on ROS.

Except where otherwise noted, the ROS wiki is licensed under [Creative Commons Attribution 3.0](#).

Wiki: Documentation (last edited 2011-06-17 23:49:49 by [Melkoepfler](#))

Wiki

- ROS
- About ROS
- Recent Changes
- Search/Recent
- Documentation

Page

- Edit (7/0)
- Info
- Subpages
- Add Link
- Attachments
- More Actions:**

User

- Anonymous
- Settings
- Login

answers.ros.org - ROS Questions & Answers

Community-supported help for ROS users



The screenshot shows the ROS Answers website interface. At the top, there is a navigation bar with 'ROS' and 'Answers' links, along with tabs for 'questions', 'tags', 'people', 'badges', and 'ask a question'. A search bar is located on the right side of the navigation bar. Below the navigation bar, there are filters for 'all', 'unanswered', and 'favorites', and a 'Sort by' dropdown menu set to 'date'. The main content area displays a list of questions, each with a title, a brief description, and a 'View' button. The questions listed include:

- Using a python node in parallel mode
- Using the "Player" package
- How much horsepower is needed to run the kinect stack
- how to set JAVA_HOME in launch file?
- add a new frame to the tree
- upgrade to 1.1.0, or wait?
- Planner for non-holonomic/differential constraints
- error installing openni_kinect
- iterate points for OpenCV's Line function
- tabletop segmentation fails (open error)
- Error installing ROS on MacBook Pro OS X 10.6.7
- installing turtlebot software on beagleboard
- roslaunch: can't exec: no such file or directory
- depth writing using openni_camera depth image
- turtlebot with gyro but no kinect

On the right side of the page, there is a 'Contributors' section with a grid of profile pictures and an 'Interesting tags' section with a list of tags and an 'Add' button.

code.ros.org - Willow Garage, Inc Code

Hosting and project management for "official" packages

ROS code
Home My Staff Search Projects Help | ROS.org
Log in | Register new account

code.ros.org

New to ROS?

Please visit [ros.org](#) to find out more information on ROS, the Robot Operating System.

About code.ros.org

code.ros.org is a development site for ROS-related software, including the [ros-pkg](#) and [wg-ros-pkg](#) projects. [ros-pkg](#) provides general robot software capabilities, while [wg-ros-pkg](#) is a repository for [Willow Garage](#)-related software, including the [PR2 robot](#).

Repositories

The following ROS-related repositories are hosted on code.ros.org:

- [ros](#): ROS core software platform
- [ros-pkg](#): general robot software libraries for ROS, including navigation, 3-D visualization, coordinate transforms, drivers, and more.
- [wg-ros-pkg](#): PR2 software, Willow Garage research, and other Willow Garage-related software.
- [opencv](#): OpenCV computer vision library
- [karto](#): Karto mapping library from SRI International

Other Repositories

There are many repositories of ROS-related software available as open source. Please see <http://ros.org/wiki/Repositories> for the most up-to-date list.

Bug Reports/Feature Requests

Please see the [Tickets page on the ROS wiki](#).

Mailing List Archives

Please see the [Lurker archives](#).

Recent News

No news items found


Activity

Recently Registered Projects

- (2010-09-18) [ROS Enhancement Proposals](#)
- (2010-08-07) [continuous_pos](#)
- (2010-08-06) [embedded_ros](#)
- (2010-07-28) [release](#)
- (2010-06-23) [PR2 Debian packages](#)
- (2010-03-30) [Classes](#)
- (2010-03-16) [Karto](#)
- (2010-02-22) [Willow Garage Publications](#)
- (2009-12-02) [PR2 Documentation](#)
- (2009-11-09) [opencv](#)

Support and Docs

- [ROS and ROS Packages Documentation](#)
- [ROS Package Browser](#)
- [GForge Documentation](#)
- [GForge Help Forum & Docs](#)



ros mailing lists

Getting in touch with the developer community

- ROS Users - *for general ROS-related discussions*
<https://code.ros.org/mailman/listinfo/ros-users>
- ROS Developers - *for ROS core development*
<https://code.ros.org/mailman/listinfo/ros-developers>
- Other Lists & List Archives
<http://code.ros.org/lurker>