5435 FROM 11/16/2021 ROS2 SUMMARIES

Contents

ROS2 CHANGES OF NOTE

1_LIMITATIONS OF ROS1

This article captures the reasons for making breaking changes to the ROS API, hence the 2.0.

Original Author: Brian Gerkey

https://design.ros2.org/articles/why_ros2.html

Still, we were guided by the PR2 use case, the salient characteristics of which included:

- a single robot;
- workstation-class computational resources on board;
- no real-time requirements (or, any real-time requirements would be met in a special-purpose manner);
- excellent network connectivity (either wired or close-proximity high-bandwidth wireless);
- applications in research, mostly academia; and

- maximum flexibility, with nothing prescribed or proscribed (e.g., “we don’t wrap your main()”).

New use cases for ROS2

Of specific interest to us for the ongoing and future growth of the ROS community are the following use cases, which we did not have in mind at the beginning of the project:

- Teams of multiple robots: while it is possible to build multi-robot systems using ROS today, there is no standard approach, and they are all somewhat of a hack on top of the single-master structure of ROS.
- Small embedded platforms: we want small computers, including “bare-metal” micro controllers, to be first-class participants in the ROS environment, instead of being segregated from ROS by a device driver.
- Real-time systems: we want to support real-time control directly in ROS, including inter-process and inter-machine communication (assuming appropriate operating system and/or hardware support).
- Non-ideal networks: we want ROS to behave as well as is possible when network connectivity degrades due to loss and/or delay, from poor-quality WiFi to ground-to-space communication links.
- Production environments: while it is vital that ROS continue to be the platform of choice in the research lab, we want to ensure that ROS-based lab prototypes can evolve into ROS-based products suitable for use in real-world applications.
- Prescribed patterns for building and structuring systems: while we will maintain the underlying flexibility that is the hallmark of ROS, we want to provide clear patterns and supporting tools for features such as life cycle management and static configurations for deployment.

DISTRIBUTIONS

The expectation is to release new ROS 2 distributions once per year. (**WATCH IT AND EXPECT CHANGES ON THE FLY!**)

2_FOXY AND UBUNTU 20.04

https://index.ros.org/doc/ros2/Releases/Release-Foxy-Fitzroy/

Tier 1 platforms:

- Ubuntu 20.04 (Focal): `amd64` and `arm64`
- Mac macOS 10.14 (Mojave)
- Windows 10 (Visual Studio 2019)

Rolling Distribution¶

The Rolling distribution of ROS 2 serves as a staging area for future stable distributions of ROS 2 and as a collection of the most recent development releases. Unlike most stable ROS 2 distributions which have an initial release, a support window during which they are updated, and a definite end of support (see List of Distributions above) the Rolling distribution is continuously updated and is subject to in-place updates which will at times include breaking changes.

Packages released into the Rolling distribution will be automatically released into future stable distributions of ROS 2. Releasing a ROS 2 package into the Rolling distribution follows the same procedures as all other ROS 2 distributions.

3_GAZEBO, PYTHON, C++

Gazebo: Some goals of the refactoring were:

- Take advantage of new ROS 2 features, such as masterless discovery.
- Remove code which duplicates functionality already present in Gazebo.
- Reduce duplication by standardizing common functionality, such as how to set ROS namespaces, parameters and topic remapping.
- Modernize the codebase, making use of the latest SDFormat, Gazebo and Ignition APIs, as well as ROS 2's style guidelines and linters.
- Add tests and demos for all ported functionality.

Currently, the only supported Gazebo version is Gazebo 9.

http://gazebosim.org/tutorials?tut=ros2_overview

4_Louise Poubel gives an overview of ROS 1 AND ROS 2

Louise Poubel gives an overview of ROS (Robot Operating System) and Gazebo (a multirobot simulator), the problems they've been solving so far and what's on the roadmap for the future. In the second half of the talk, a hands-on demo walks through the creation of a robot in simulation and controlling and inspecting it using ROS 2, the next generation ROS. 37:14

https://qconsf.com/sf2018/presentation/open-source-robotics-hands-gazebo-and-ros-2

https://www.infoq.com/presentations/gazebo-ros-2/?utm_source=presentations&utm_medium=sf&utm_campaign=qcon

(There is a transcript with this video.)

5_Example and Code Walk through for publisher in Python/Ros2

https://index.ros.org/doc/ros2/Tutorials/Writing-A-Simple-Py-Publisher-And-Subscriber/#id7

```
import rclpy
from rclpy.node import Node


from std_msgs.msg import String



class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0
```

Node name and namespace parameters changed¶

The `Node` action parameters related to naming have been changed:

- `node_name` has been renamed to `name`
- `node_namespace` has been renamed to `namespace`
- `node_executable` has been renamed to `executable`
- `exec_name` has been added for naming the process associated with the node. Previously, users would have used the `name` keyword argument.

The old parameters have been deprecated.

COMMAND LINE CHANGES

`$ ros2 verb sub-verb <positional-argument> <optional-arguments>  (See ROS2 Cheat Sheets)`

`ros2 param list`

6_A few commands are added – See ROS2 Cheatsheet in Appendix

`$ ros2 pkg xml <package-name>`
outputs the entirety of the xml manifest of a given package.

https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/cli/cli_cheats_sheet.pdf

doctor A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.
Alias: **wtf** (where's the fire).
Arguments:

--report/-r	Output report of all checks.
--report-fail/-rf	Output report of failed checks only.
--include-warning/-iw	Include warnings as failed checks.

Examples:

```
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
$ ros2 doctor --include-warning --report-fail
```

or similarly,

```
$ ros2 wtf
```

7_ROS CLIENT LIBRARIES (RCL) AND NEW LANGUAGES

https://index.ros.org/doc/ros2/Concepts/ROS-2-Client-Libraries/

8_CONCEPTS

https://index.ros.org/doc/ros2/Concepts/#quick-overview-of-graph-concepts

- About different ROS 2 DDS/RTPS vendors
- About Quality of Service settings
- About ROS 2 interfaces

- About topic statistics
- About ROS 2 client libraries
- About logging and logger configuration
- About parameters in ROS 2

9_ROS2 WEBSITES TO EXPLORE

https://index.ros.org/

Welcome to ROS Index and ROS Discourse

ROS Index is the entry point for searching ROS and ROS 2 resources, including packages, repositories, system dependencies and documentation.

You can enter keywords and phrases in the search bar and then filter results by resource type, or you can browse the complete package, repository and system dependency lists under the **Index** tab.

Under the **Doc** tab, you'll find the official ROS 2 documentation, including installation instructions, tutorials, distribution features and summaries, contributing guides, and more.

https://discourse.ros.org/

10_ROS2 FOXY

https://index.ros.org/doc/ros2/Releases/Release-Foxy-Fitzroy/

Table of Contents

Foxy Fitzroy is the sixth release of ROS 2.

11_INTROSPECTION WITH COMMAND LINE TOOLS¶

Table of Contents ROS 2 includes a suite of command-line tools for introspecting a ROS 2 system.

https://index.ros.org/doc/ros2/Tutorials/Introspection-with-command-line-tools/

```
ros2 --help
```

12_DEEPER DETAILS OF ROS2 VS ROS1

12_Learn all about Robot Operating System 2, including how it's used and how it differs from the original ROS. (ROS Maker)

https://maker.pro/ros/tutorial/robot-operating-system-2-ros-2-introduction-and-getting-started

ROS 1 vs. ROS 2 Table Summary

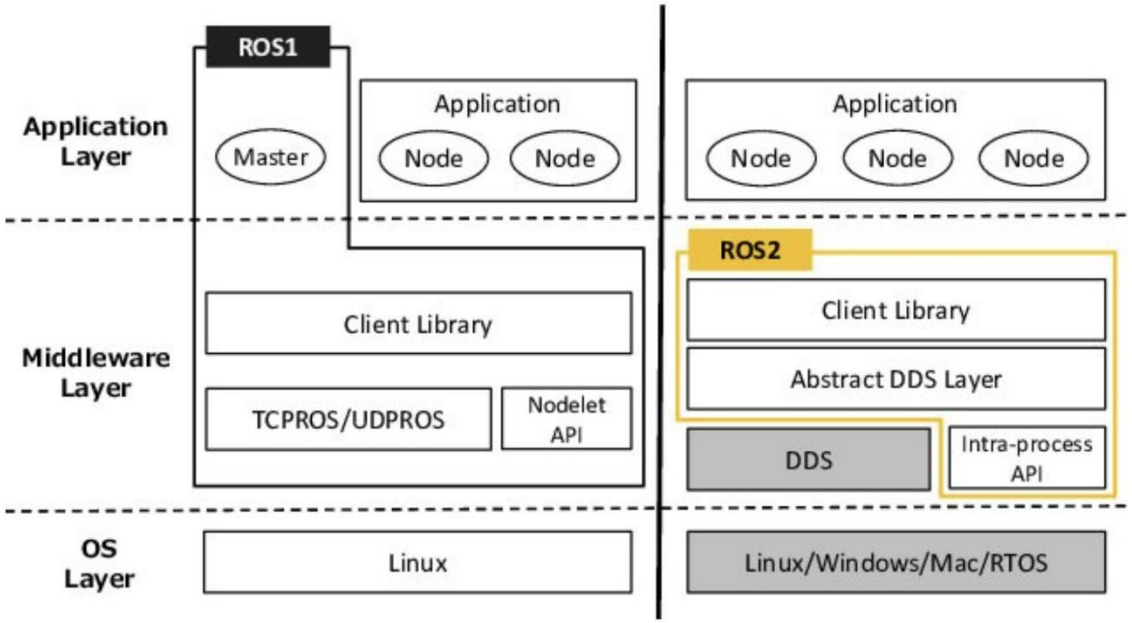
ROS	ROS 2
Uses TCPROS (custom version of TCP/IP) communication protocol	Uses DDS (Data Distribution System) for communication
Uses ROS Master for centralized discovery and registration. Complete communication pipeline is prone to failure if the master fails	Uses DDS distributed discovery mechanism. ROS 2 provides a custom API to get all the information about nodes and topics

ROS	ROS 2
ROS is only functional on Ubuntu OS	ROS 2 is compatible with Ubuntu, Windows 10 and OS X
Uses C++ 03 and Python2	Uses C++ 11 (potentially upgradeable) and Python3
ROS only uses CMake build system	ROS 2 provides options to use other build systems
Has a combined build for multiple packages invoked using a single CMakeLists.txt	Supports isolated independent builds for packages to better handle inter-package dependencies
Data Types in message files do not support default values	Data types in message files can now have default values upon initialization
roslaunch files are written in XML with limited capabilities	roslaunch files are written in Python to support more configurable and conditioned execution
Cannot support real-time behavior deterministically even with real-time OS	Supports real-time response with apt RTOS like RTPREEMPT

13_ROS2 ARCHITECTURE AND DDS

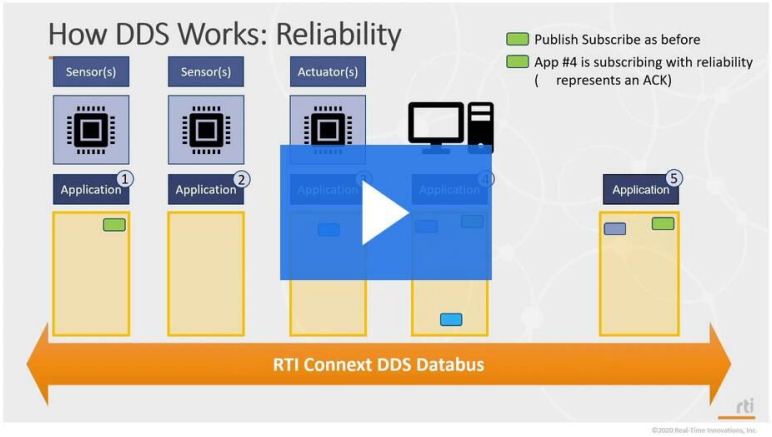
This is the big Claim to Fame for ROS 2 – No Master, Special Communications

13_a_Architecture



https://www.researchgate.net/publication/309128426_Exploring_the_performance_of_ROS2/figures?lo=1

13_b_Short_Video-of_DDS



[Data Distribution Service (DDS) for Complex Systems | RTI]

13_c_DDS-Security overview

https://design.ros2.org/articles/ros2_dds_security.html

The DDS-Security specification expands upon the DDS specification, adding security enhancements by defining a Service Plugin Interface (SPI) architecture, a set of builtin implementations of the SPIs, and the security model enforced by the SPIs. Specifically, there are five SPIs defined:

- **Authentication**: Verify the identity of a given domain participant.
- **Access control**: Enforce restrictions on the DDS-related operations that can be performed by an authenticated domain participant.
- **Cryptographic**: Handle all required encryption, signing, and hashing operations.

- **Logging**: Provide the ability to audit DDS-Security-related events.
- **Data tagging**: Provide the ability to add tags to data samples.

ROS 2's security features currently utilize only the first three. This is due to the fact that neither **Logging** nor **Data Tagging** are required in order to be compliant with the DDS-Security spec (see section 2.3), and thus not all DDS implementations support them. Let's delve a little further into those first three plugins.

13_d ROS on DDS

https://design.ros2.org/articles/ros_on_dds.html

13_e_Do you really want to understand DDS?

13_e_ROS 2 + DDS Interoperation 1:03:00 371 views •Oct 8, 2020 1:03:00

https://www.youtube.com/watch?v=GGqcrccWfeE&t=8s

- TIME: Starts with Vehicles
- Uses of DDS
- 10:00 What is DDS?
- 43:00 ROS2 and DDS

14_INTRODUCTION TO REAL-TIME SYSTEMS

This article is a brief survey of real-time computing requirements and methods to achieve real-time performance.

Original Author: Jackie Kay

https://design.ros2.org/articles/realtime_background.html

Some examples of real-time environments:

- The `RT_PREEMPT` Linux kernel patch, which modifies the Linux scheduler to be fully preemptible (3).
- Xenomai, a POSIX-compliant co-kernel (or hypervisor) that provides a real-time kernel cooperating with the Linux kernel. The Linux kernel is treated as the idle task of the real-time kernel's scheduler (the lowest priority task).
- RTAI, an alternative co-kernel solution.
- QNX Neutrino, a POSIX-compliant real-time operating system for mission-critical systems.

14_b Mike Moore's Slides for Bobble Bot

Slides_Real-Time Control of Balancing

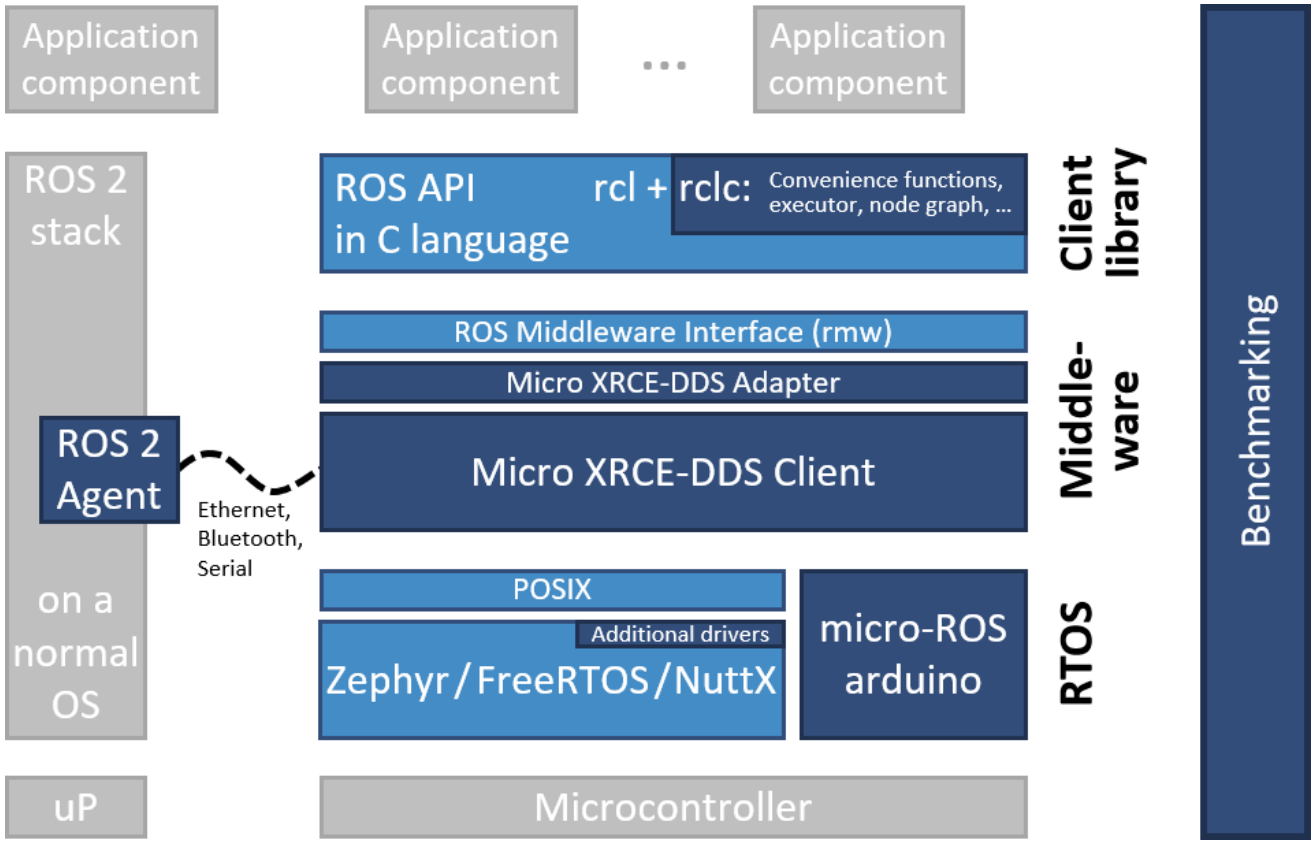
GET SLIDES 14_C FROM 5435 LECTURE REFERENCES 12/1/2020

14_c_Bobble_Bot_ super-owesome/bobble_controllers_ PREEMPT_RT

https://github.com/super-owesome/bobble_controllers

15_MICRO-ROS | ROS 2 FOR MICROCONTROLLERS

https://micro-ros.github.io/



micro-ROS- Short Video

micro-ROS puts the Robot Operating System on Microcontrollers _SHORT VIDEO
5,219 views •Nov 5, 2019 4:36

https://youtu.be/slMhPRnBVwM

APPENDIX 1 ROS REFERENCES – NOETIC AND ROS2

ROS AND PYTHON UPDATES 11/17/2020

WARNING: IT AIN'T OVER UNTIL THE TURTLE (ROS DISTRIBUTION) SWIMS! BE CAREFUL OF TAKING ANY STATEMENTS EVEN ON THE ROS WIKI AS THE FINAL WORD. ALL OF THE SOFTWARE, OS, AND ROS PACKAGES ARE SUBJECT TO CHANGES AND UPDATES. TEST, TEST, TEST.

Contents

ROS Noetic: What You Need to Know

July 15, 2020

https://varhowto.com/ros-noetic/#Python_3_in_Noetic

As a LTS version, ROS Noetic will be **supported for 5 years until May 2025**, which marks the End of Life (EOL) date. It is also good to know that all ROS LTS versions are supported for 5 years.

Noetic is also **the last and final official release of ROS 1**.

ROS Noetic is mainly developed for Ubuntu 20.04.

Check out what a developer says:

https://www.openrobotics.org/blog/2020/5/23/noetic-ninjemys-the-last-official-ros-1-release

Python 3 in Noetic

The main change is that the version of Python in ROS Noetic is bumped to Python 3 (3.8) in Ubuntu 20.04, compared to Python 2.7 in Ubuntu 18.04

One of the major difference from Python 2 (Python 2.7) to Python 3 is the `print` statement syntax. You will need to bracket the arguments when using `print` in Python 3. (This one got me in one of my programs.)

Gazebo 11 in Noetic

ROS Noetic is using Gazebo 11 now, compared to Gazebo 9 in ROS Melodic.

Python 3 and CMake are the 2 major changes. For migration to ROS Noetic, please read the migration guide on ROS Wiki: http://wiki.ros.org/noetic/Migration.

The point cloud library is PCL 1.10 in Noetic, compared to 1.8 in Melodic.

OpenCV is 4.2 now on Noetic from 3.2 from ROS Melodic.

It is also helpful to check out the ROS Noetic distribution file on GitHub to see all the packages currently available: https://github.com/ros/rosdistro/blob/master/noetic/distribution.yaml. Each package occupies 15 lines, so you can get a rough number by having the total line number divide by 15. You can also use the history function to check what are the packages released for ROS Noetic.6627 Lines/15 = 441 Packages

ROS2 COMMAND LINE TOOLS

Let's go over some of the commands:

https://index.ros.org/doc/ros2/Tutorials/Introspection-with-command-line-tools/

https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/cli/cli_cheats_sheet.pdf

ROS 2

Why ROS2 ?

Original Author: Brian Gerkey Explains why the switch to ROS2

We started work on ROS in November 2007. A lot has happened since then and we believe that it is now time to build the next generation ROS platform. In this article we will explain why.

https://design.ros2.org/articles/why_ros2.html

ROS2 Videos

I have listed a few videos here that help us understand the changes in ROS 2. Any information presented before the official release of a distribution and anything before 2020 should be taken as “possibilities” rather than a formal description of the software. Check the date on any document or video.

ROS2 for Beginners from BotBuilder - 14 short videos.

ROS2 Tutorial for Beginners (Foxy)

14 videos7,729 viewsLast updated on Jun 16, 2020

A series of short videos averaging about 10 to 12 minutes about ROS2 including Python and C++ examples.

ROS2 Basics #1 - Installing and Configuring Your ROS2 Environment

ROS2 Basics #2 - Introducing Turtlesim, Command Line

ROS2 Basics #3 - Understanding ROS2 Packages and ...

ROS2 Basics #4 - Understanding ROS2 Executables and Nodes

ROS2 Basics #5 - Understanding ROS2 Topics - YouTube

ROS2 Basics #6 - Understanding ROS2 Services

ROS2 Basics #7 - Understanding ROS2 Actions

ROS2 Basics #8 - Understanding ROS2 Parameters

ROS2 Basics #9 - Understanding ROS2 Launch File

ROS2 Basics #10 - Writing a Simple Publisher and Subscriber

ROS2 Basics #11 - Writing a Simple Publisher and Subscriber

ROS2 Basics #12 - Writing a Simple Service and Client (C++)

ROS2 Basics #13 - Writing a Simple Service and Client ...

ROS2 Basics #14 - ROS2 tools - RQt and Ros2bag

Discussion of migrations using Noetic, ROS2, Python and Gazebo. 25:43

https://www.youtube.com/watch?v=QAi6bORqQno

7:00 Goals for ROS2

11:25 Eloquent

17:00 Gazebo to Ignition

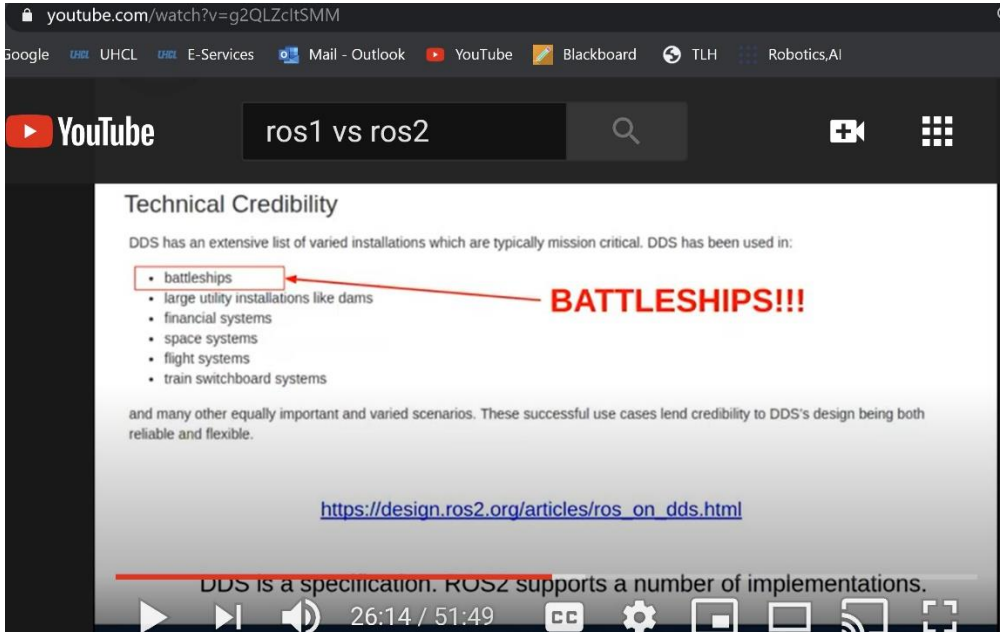
ROS1 versus ROS2 Gavin Suddrey 51:49 June 15, 2020

https://www.youtube.com/watch?v=g2QLZcItSMM

0-25:00 Review of ROS1

Start ROS 2 at about 24:00

25:00 Discussion of DDS – One of the Biggest differences. Data transport is supported by this format (Distributed Data Services).



His examples of Talker and Listener describe the changes in Python programs. Very detailed.

Do you really want to understand DDS?

ROS 2 + DDS Interoperation 1:03:00

371 views •Oct 8, 2020 1:03:00

https://www.youtube.com/watch?v=GGqcrccWfeE&t=8s

Starts with Vehicles

Uses of DDS

10:00 What is DDS?

43:00 ROS2 and DDS

APPENDIX 2 ROS2 CHEAT SHEETS

ROS 2 Cheats Sheet

Command Line Interface

All ROS 2 CLI tools start with the prefix 'ros2' followed by a command, a verb and (possibly) positional/optional arguments.

For any tool, the documentation is accessible with,

```
$ ros2 command --help
```

and similarly for verb documentation,

```
$ ros2 command verb -h
```

Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,

```
$ ros2 command [tab][tab]
```

Some of the examples below rely on:
ROS 2 demos package.

---

action Allows to manually send a goal and displays debugging information about actions.

Verbs:

info	Output information about an action.
list	Output a list of action names.
send_goal	Send an action goal.
show	Output the action definition.

Examples:

```
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \
 action_tutorials/action/Fibonacci "order: 5"
$ ros2 action show action_tutorials/action/Fibonacci
```

---

bag Allows to record/play topics to/from a rosbag.

Verbs:

info	Output information of a bag.
play	Play a bag.
record	Record a bag.

Examples:

```
$ ros2 info <bag-name>
$ ros2 play <bag-name>
$ ros2 record -a
```

---

component Various component related verbs.

Verbs:

list	Output a list of running containers and components.
load	Load a component into a container node.
standalone	Run a component into its own standalone container node.
types	Output a list of components registered in the ament index.
unload	Unload a component from a container node.

Examples:

```
$ ros2 component list
$ ros2 component load /ComponentManager \
 composition composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager 1
```

---

daemon Various daemon related verbs.

Verbs:

start	Start the daemon if it isn't running.
status	Output the status of the daemon.
stop	Stop the daemon if it is running

---

doctor A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.

Alias: **wtf** (where's the fire).

Arguments:

--report/-r	Output report of all checks.
--report-fail/-rf	Output report of failed checks only.
--include-warning/-iw	Include warnings as failed checks.

Examples:

```
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
$ ros2 doctor --include-warning --report-fail
```

or similarly,

```
$ ros2 wtf
```

---

extension_points List extension points.

---

extensions List extensions.

---

interface Various ROS interfaces (actions/topics/services)-related verbs. Interface type can be filtered with either of the following option, '--only-actions', '--only-msgs', '--only-srvs'.

Verbs:

list	List all interface types available.
package	Output a list of available interface types within one package.
packages	Output a list of packages that provide interfaces.
proto	Print the prototype (body) of an interfaces.
show	Output the interface definition.

Examples:

```
$ ros2 interface list
$ ros2 interface package std_msgs
$ ros2 interface packages --only-msgs
$ ros2 interface proto example_interfaces/srv/AddTwoInts
$ ros2 interface show geometry_msgs/msg/Pose
```

---

launch Allows to run a launch file in an arbitrary package without to 'cd' there first.

Usage:

```
$ ros2 launch <package> <launch-file>
```

Example:

```
$ ros2 launch demo_nodes_cpp add_two_ints.launch.py
```

---

lifecycle Various lifecycle related verbs.

Verbs:

get	Get lifecycle state for one or more nodes.
list	Output a list of available transitions.
nodes	Output a list of nodes with lifecycle.
set	Trigger lifecycle state transition.

---

msg (deprecated) Displays debugging information about messages.

Verbs:

list	Output a list of message types.
package	Output a list of message types within a given package.
packages	Output a list of packages which contain messages.
show	Output the message definition.

Examples:

```
$ ros2 msg packages
$ ros2 msg show geometry_msgs/msg/Pose
```

---

multicast Various multicast related verbs.

Verbs:

receive	Receive a single UDP multicast packet.
send	Send a single UDP multicast packet.

---

node Displays debugging information about nodes.

Verbs:

info	Output information about a node.
list	Output a list of available nodes.

Examples:

```
$ ros2 node info /talker
$ ros2 node list
```

---

param Allows to manipulate parameters.

Verbs:

delete	Delete parameter.
describe	Show descriptive information about declared parameters.
dump	Dump the parameters of a given node in yaml format, either in terminal or in a file.
get	Get parameter.
list	Output a list of available parameters.
set	Set parameter

Examples:

```
$ ros2 param delete /talker /use_sim_time
$ ros2 param get /talker /use_sim_time
$ ros2 param list
$ ros2 param set /talker /use_sim_time false
```

---

pkg Create a ros2 package or output package(s)-related information.

Verbs:

create	Create a new ROS2 package.
executables	Output a list of package specific executables.
list	Output a list of available packages.
prefix	Output the prefix path of a package.
xml	Output the information contained in the package xml manifest.

Examples:

```
$ ros2 pkg prefix std_msgs
$ ros2 pkg xml -t version
```

---

run Allows to run an executable in an arbitrary package without having to 'cd' there first.

Usage:

```
$ ros2 run <package> <executable>
```

Example:

```
$ ros2 run demo_node_cpp talker
```

---

security Various security related verbs.

Verbs:

create_key	Create key.
create_permission	Create keystore.
generate_artifacts	Create permission.
list_keys	Distribute key.
create_keystore	Generate keys and permission files from a list of identities and policy files.
distribute_key	Generate XML policy file from ROS graph data.
generate_policy	List keys.

Examples (see sros2 package):

```
$ ros2 security create_key demo_keys /talker
$ ros2 security create_permission demo_keys /talker \
 policies/sample_policy.xml
$ ros2 security generate_artifacts
$ ros2 security create_keystore demo_keys
```

---

service Allows to manually call a service and displays debugging information about services.

Verbs:

call	Call a service.
find	Output a list of services of a given type.
list	Output a list of service names.
type	Output service's type.

Examples:

```
$ ros2 service call /add_two_ints \
 example_interfaces/AddTwoInts "a: 1, b: 2"
$ ros2 service find rcl_interfaces/srv/ListParameters
$ ros2 service list
$ ros2 service type /talker/describe_parameters
```


Verbs:

list	Output a list of available service types.
package	Output a list of available service types within one package.
packages	Output a list of packages which contain services.
show	Output the service definition.

---

test Run a ROS2 launch test.

---

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

Verbs:

bw	Display bandwidth used by topic.
delay	Display delay of topic from timestamp in header.
echo	Output messages of a given topic to screen.
find	Find topics of a given type type.
hz	Display publishing rate of topic.
info	Output information about a given topic.
list	Output list of active topics.
pub	Publish data to a topic.
type	Output topic's type.

Examples:

```
$ ros2 topic bw /chatter
$ ros2 topic echo /chatter
$ ros2 topic find rcl_interfaces/msg/Log
$ ros2 topic hz /chatter
$ ros2 topic info /chatter
$ ros2 topic list
$ ros2 topic pub /chatter std_msgs/msg/String \
 'data: Hello ROS 2 world'
$ ros2 topic type /rosout
```