Ros2 R & D Brief

ROS2 Brief

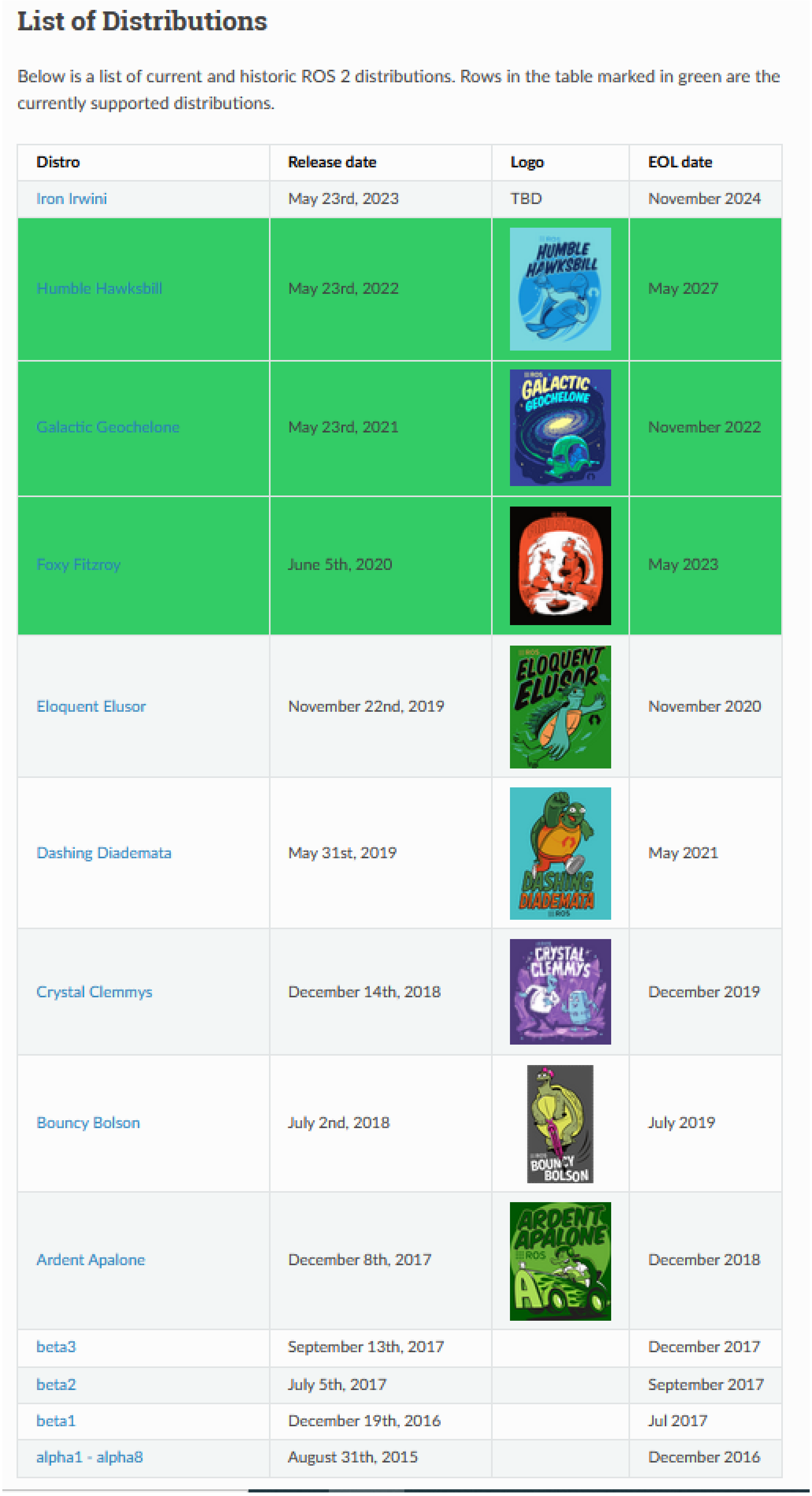
Ros2 is a huge update of ros1.

Ros1 was born in 2007. With the support of the active open source community, its functions are constantly enriched and the number of codes is constantly increasing. However, its overall design is not very scientific, lacking in security, real-time and robustness, and does not conform to industrial and specific industrial applications.

The leading team of ROS is aware of this problem, but ros1 has been accumulated for a long time. Some important modifications involving the bottom layer will make ros1 more unstable and inevitably encounter a large number of ros1 package code compatibility problems. Instead of mending, it is better to recreate a more scientific and stable ros2.

Currently, noetic of ros1 will stop supporting in 2025. For the sake of a large number of existing code bases of ros1, it is feasible to keep existing projects in ros1. However, it is undoubtedly inevitable to migrate ros2 in the future, so it is necessary to learn to migrate and use ros2 now.

Ros2, like ros1, is released once a year. After testing, jumble and Ubuntu 22.04 are not stable enough, and galactic will stop supporting it. Therefore, foxy version + Ubuntu 20.04 is selected. At present, the core functions of ros2 have been improved, and some third-party packages are still being adapted.



List of Distributions

Below is a list of current and historic ROS 2 distributions. Rows in the table marked in green are the currently supported distributions.

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023	TBD	November 2024
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018
beta3	September 13th, 2017		December 2017
beta2	July 5th, 2017		September 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		December 2016

Characteristic

The update of ros2 is all-round, so I will only briefly describe what I know at present.

API

The bottom layer of ros2 is written in C language, and its name is RCL library. The rclcpp and rclpy called in the upper layer program are packaged on the RCL library. The advantage of this is to make the call APIs of CPP and python programs more uniform and similar. At the same time, when the function of ros2 is updated, the RCL library is directly updated, and the support of CPP and Python is added.

Language

CPP

Cpp11, cpp1a4 and cpp17 standards are used by default, which is more modern than the cpp98 standard used by ros1 by default. Support more security and efficiency features.

Python

Give up python2 completely. Embrace python3 completely.

Node

For a more standardized node writing standard, you must create a class that inherits from the node object (for example, rclcpp:: node in CPP and rclpy.node.node in Python). Easy for team development.

Launch

Unlike ros1's extensive XML format launch files, ros2 recommends using Python files to write launch files, providing more flexibility in startup.

signal communication

Ros2 no longer has the master master node in ros1. The child nodes in ros1 need to register with the master node before communicating with each other. This is a centralized architecture. While ros2 adopts the distributed node mode, and the nodes can discover each other without any master node. It avoids the collapse of the whole system due to the collapse of the master node, and makes the distributed deployment of ros2 more flexible.

At the same time, the service request in ros1 is blocked, and the client will be in the stop the world state before receiving the response.

The service of ros2 is designed to be asyn asynchronous, so the client will not get stuck in receiving the response from the server. However, blocking mode can also be used if desired.

Differences in specific use

Only the common parts are listed

ROS1	*ROS2*	*introduction*
catkin_make	colcon build	compile
roslaunch	ros2 launch	Start ROS
rostopic list	ros2 topic list	
rostopic echo	ros2 topic echo	
rosrun	ros2 run	
rosrun rqt_graph rqt_graph	rqt_graph	
rosrun rqt_tf_tree rqt_tf_tree	ros2 run tf2_tools view_frames.py	Ros2 will save a PDF file in the terminal path, which cannot be viewed directly
rviz	rviz2	

Install ROS2 foxy

The installation process can be seen on the official website： https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

Install ROS 2 package

Update software library

```
sudo apt update
```

The ROS 2 package is built on frequently updated Ubuntu systems. It is always recommended that you ensure that your system is up-to-date before installing new software packages.

```
sudo apt upgrade
```

It is recommended to install the full version of ROS, including ROS base rviz tutorial and other software.

```
sudo apt install ros-foxy-desktop
```

· If only ROS core software package is required

```
sudo apt install ros-foxy-ros-base
```

Packages that may be required for subsequent development：

```
sudo apt install ros-foxy-turtlesim
sudo apt install ros-foxy-xacro
```

```
sudo apt install python3-pip
```

```
sudo apt install python3-colcon-common-extensions
```

```
sudo apt install python3-vcstool
```

```
sudo apt-get install ros-foxy-joint-state-publisher-gui
```

Environment configuration

Configure Environment

Execute the following command to configure the terminal environment. Execute this command every time you open the terminal.

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

Or use once and for all, and automatically configure the terminal every time you open it.

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

Install Gazebo

Gazebo is a powerful simulation platform provided by ROS

Just install gazebo 11 and related ROS support package：

```
sudo apt install gazebo11 ros-foxy-gazebo-ros-pkgs
```

For navigation, you can install the following packages

```
sudo apt install ros-foxy-cartographer
```

```
sudo apt install ros-foxy-cartographer-ros
```

```
sudo apt install ros-foxy-navigation2
```

```
sudo apt install ros-foxy-nav2-bringup
```

```
sudo apt install ros-foxy-gazebo-ros2-control
```

```
sudo apt install ros-foxy-ros2-control ros-foxy-ros2-controllers
```

Compile

Enter the source code package

```
cd ~s/yahboomcar_ws
```

```
colcon build --symlink-install
```

```
echo 'source ~/yahboomcar_ws/install/setup.bash' >> ~/.bashrc
```

Colcon provides many parameter options (common parameters):

1.--symlink-install ： Use symbolic links instead of copying files. For example, taking dynamic link libraries as an example, symbolic links will be used in the install directory to point to the library files generated in the build directory (such as *. So) Without this option, both directories will have the library file

2.--packages-select： Compile only the specified package, such as: colcon build --packages-select autoware_map_msgs vector_map_msgs

3.--packages-ignore： Ignore the specified package, as above

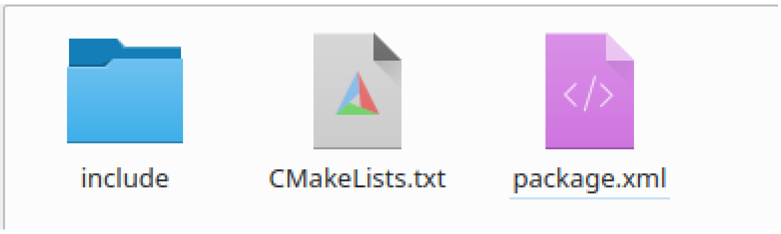
4--continue-on-error： Continue to compile other modules after compilation error

5--cmake-args， --ament-cmake-args， --catkin-cmake-args： Pass parameters to the corresponding package

Unlike ros1, ros2 should recompile colcon build after modifying any files

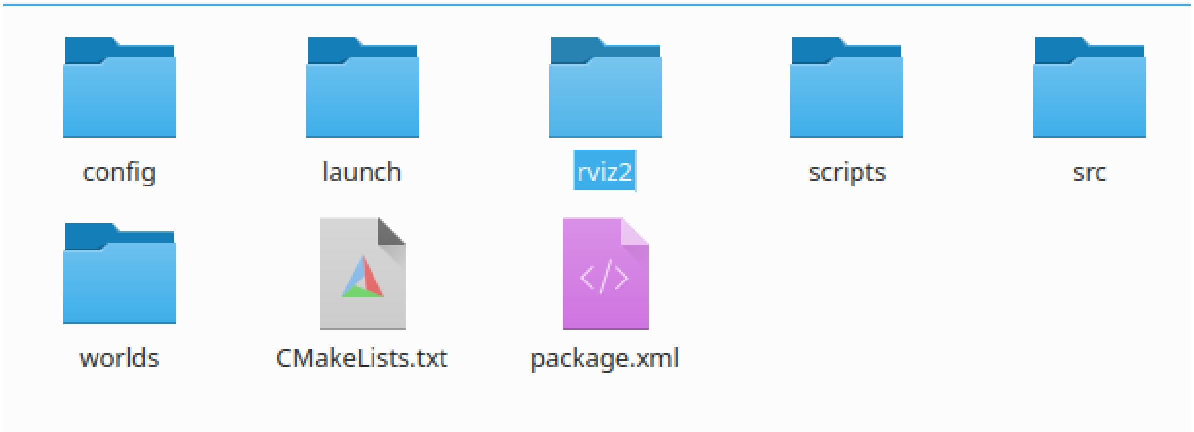
Package configuration

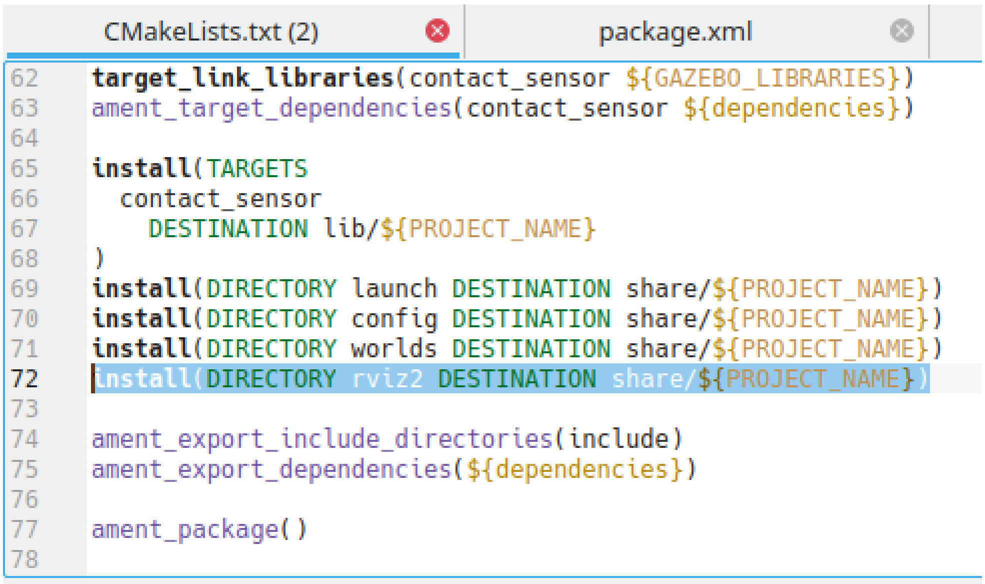
Visible in the created package



Modify the package name in project () of cmakelists.txt and the name of package.xml to modify the name of the software package.

At the same time, in order to use the files in the folder in the software package, you need to install in cmakelist.





```
target_link_libraries(contact_sensor ${GAZEBO_LIBRARIES})
ament_target_dependencies(contact_sensor ${dependencies})

install(TARGETS
  contact_sensor
    DESTINATION lib/${PROJECT_NAME}
)
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
install(DIRECTORY config DESTINATION share/${PROJECT_NAME})
install(DIRECTORY worlds DESTINATION share/${PROJECT_NAME})
install(DIRECTORY rviz2 DESTINATION share/${PROJECT_NAME})

ament_export_include_directories(include)
ament_export_dependencies(${dependencies})

ament_package()

```