D3.17

Micro-ROS middleware interface Software Release Y4

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D3.17
Deliverable name	Micro-ROS middleware interface
Date	June 2021
Dissemination level	public
Workpackage and task	WP3 - Task 3.2
Author	Borja Outerelo Gamarra (eProsima)
Contributors	Pablo Garrido Sánchez (eProsima)
Keywords	Micro-ROS, robotics, ROS2, microcontroller, middleware
Abstract	This document provides links to the released software and documentation for deliverable D3.17 *Micro-ROS middleware interface* of the Task 3.2 *MicroROS middleware interface (urmw)*.

Contents

1 Summary

micro-ROS provides an RMW implementation using the latest code from eProsima's middleware for devices with low computing and memory capacities, Micro XRCE-DDS. In this version, we have a C implementation for part of the middleware primitives declared by the ROS *rmw* upper layer.

In micro-ROS, *rmw_microxrcedds* will be used as the default one by *rmw_implementation* despite the fact that during 2021 other implementations have appeared. One example of them is *rmw_embeddedrtps*.

The *rmw_microxrcedds* library is a ROS2 Middleware Abstraction Interface used by micro-ROS. This library defines the interface used by upper layers in ROS2 stack and implemented using some middleware in the lower layers.

2 Acronyms and keywords

Term	Definition
RMW	ROS2 Middleware
XRCE	Extremely Resource Constrained Environments
DDS	Data Distribution Service
rmw_microxrcedds	Micro-ROS middleware for XRCE-DDS Client

3 Overview to Results

This document provides links to the released software and documentation for deliverable D3.17 *Micro-ROS middleware interface* of the Task 3.2 *MicroROS middleware interface (urmw).*

4 Links to Software Repositories

The rmw_microxrccedds is provided as a ROS2 package available at:

- Git repository: https://github.com/microROS/rmw-microxrcedds.git

Branch	Latest commit	ROS 2 version
main	f738f9c	rolling
galactic	50cdbf8	galactic
foxy	1155805	foxy
dashing	cee44c0	dashing/eloquent
crystal	5c55709	crystal

The rosidl_typesupport_microxrcedds is provided as a ROS2 packages available at:

- Git repository: https://github.com/microROS/rosidl_typesupport_microxrcedds.git
  Package names:
- rosidl_typesupport_microxrcedds_c

Branch	Latest commit	ROS 2 version
main	7efacee	rolling
galactic	35b1212	galactic
foxy	197b1ae	foxy
dashing	e99734f	dashing
crystal	cff523d	crystal

API references for most important parts of the micro-ROS framework can be found in micro.ros.org/docs/api/overview/. Documentation of the project has been migrated to micro.ros.org and a build system tool has been created in order to ease the utilization.

The micro-ROS build system is available at:

- Git repository: https://github.com/micro-ROS/micro_ros_setup.git

Branch	Latest commit	ROS 2 version
main	0c61c67	rolling
galactic	92599b1	galactic
foxy	e920c89	foxy
dashing	ffbd034	dashing/eloquent
crystal	2138d50	crystal

5 Annex 1: GitHub documentation (rmw)

Content of README.md of rmw_microxrccedds repo from 25th November 2021.

5.1 Overview

All packages contained in this repository are a part of the Micro-ROS project stack. For more information about micro-ROS project click here.

5.2 Packages

The repository contains the following packages:

5.2.1 rmw_microxrcedds_c

This layer is the ROS 2 Middleware Abstraction Interface written in C. This package provides a middleware implementation for XRCE-DDS (rmw layer). The implementation wraps the latest code from eProsima's Micro XRCE-DDS client to communicate with the DDS world. This library defines the interface used by upper layers in the ROS 2 stack, and that is implemented using XRCE-DDS middleware in the lower layers.

5.2.1.1 Library configuration

This RMW implementation can be configured via CMake arguments, its usual to configure them via `colcon.meta` file in a micro-ROS enviroment.

Most of these configuration are related to memory management because this RMW implementation tries to fully rely on static memory assignations. This leads to an upper bound in memory assignations, which is configured by the user before the build process. By default, the package sets the values for all memory bounded. The upper bound is configurable by a file that sets the values during the build process.

More details about RMW Micro XRCE-DDS can be found here. All the configurable parameters are:

Table ommited due to size. It can be checked here: https://github.com/micro-ROS/rmw_microxrcedds/tree/galactic#library-configuration

5.3 Purpose of the Project

This software is not ready for production use. It has neither been developed nor tested for a specific use case. However, the license conditions of the applicable Open Source licenses allow you to adapt the software to your needs. Before using it in a safety relevant setting, make sure that the software fulfills your requirements and adjust it according to any applicable safety standards, e.g., ISO 26262.

5.4 License

This repository is open-sourced under the Apache-2.0 license. See the LICENSE file for details.

For a list of other open-source components included in this repository, see the file 3rd-party-licenses.txt.

5.5 Known Issues/Limitations

There are no known limitations.

6 Annex 2: GitHub documentation (type support)

Content of README.md of rosidl_typesupport_microxrcedds repo from 25th November 2021.

6.1 Overview

All packages contained in this repository are a part of the micro-ROS project stack. For more information about the micro-ROS project click here.

6.2 Packages

The repository contains the following packages:

6.2.1 rmw_typesupport_microxrcedds_c

This package aims to give support to the rmw layer for ros messages in C language.

6.2.1.1 Includes

- Support for serialization / serialization code, generated during the build process, for each ROS 2 interface.
- Support for unbounded types for incoming messages using static memory.
- Support for building message support using ament extensions for finding not built interfaces.

6.3 Purpose of the Project

This software is not ready for production use. It has neither been developed nor tested for a specific use case. However, the license conditions of the applicable Open Source licenses allow you to adapt the software to your needs. Before using it in a safety relevant setting, make sure that the software fulfills your requirements and adjust it according to any applicable safety standards, e.g., ISO 26262.

6.4 License

This repository is open-sourced under the Apache-2.0 license. See the LICENSE file for details.

For a list of other open-source components included in ROS 2 system_modes, see the file 3rd-party-licenses.txt.

6.5 Known Issues/Limitations

There are no known limitations.

7 Annex 2: GitHub documentation (build system)

Content of README.md of micro_ros_setup repo from 25th November 2021.

This ROS 2 package is the entry point for building micro-ROS apps for different embedded platforms.

7.1 Supported platforms

This package is the **official build system for micro-ROS**. It provides tools and utils to cross-compile micro-ROS with just the common ROS 2 tools for these platforms:

RTOS	Platform	Version	Example
FreeRTOS	Renesas RA6M5	Renesas e2 studio	`renesas_ra ra6m5`
FreeRTOS	Olimex STM32-E407	STM32CubeMX latest	`freertos olimex-stm32-e407`
FreeRTOS	ST Nucleo F446RE 1	STM32CubeMX latest	`freertos nucleo_f446re`
FreeRTOS	ST Nucleo F446ZE 1	STM32CubeMX latest	`freertos nucleo_f446ze`
FreeRTOS	ST Nucleo F746ZG 1	STM32CubeMX latest	`freertos nucleo_f746zg`
FreeRTOS	ST Nucleo F767ZI 1	STM32CubeMX latest	`freertos nucleo_f767zi`
FreeRTOS	Espressif ESP32	v8.2.0	`freertos esp32`
FreeRTOS	Crazyflie 2.1	v10.2.1 - CF 2020.06	`freertos crazyflie21`
Zephyr	Olimex STM32-E407	v2.6.0	`zephyr olimex-stm32-e407`
Zephyr	ST B-L475E-IOT01A	v2.6.0	`zephyr discovery_l475_iot1`
Zephyr	ST Nucleo H743ZI 1	v2.6.0	`zephyr nucleo_h743zi`
Zephyr	Zephyr emulator	v2.6.0	`zephyr host`
Mbed	ST B-L475E-IOT01A	v6.6	`mbed disco_l475vg_iot01a`
-	Library 3	-	`generate_lib`
Linux	*Host 2*	Ubuntu 18.04/20.04	`host`
Android	AOSP 4	Latest	`android generic`

1 Community supported, may have lack of official support

2 Support for compiling apps in a native Linux host for testing and debugging

3 a valid CMake toolchain with custom crosscompilation definition is required

4 Community supported, may have lack of official support

7.1.1 Standalone build system tools

micro-ROS also offers some other ways to crosscompile it for different platforms. These other options are secondary tools and may not have full support for all features. Currently micro-ROS is also available as:

- a standalone **micro-ROS component for Renesas e2 studio and RA6M5**: this package enables the integration of micro-ROS in Renesas e2 studio and RA6M5 MCU family.
- a standalone **micro-ROS component for ESP-IDF**: this package enables the integration of micro-ROS in any Espressif ESP32 IDF project.
- a standalone **micro-ROS module for Zephyr RTOS**: this package enables the integration of micro-ROS in any Zephyr RTOS workspace.
- a standalone **micro-ROS module for Mbed RTOS**: this package enables the integration of micro-ROS in any Mbed RTOS workspace.
- a standalone **micro-ROS module for NuttX RTOS**: this package enables the integration of micro-ROS in any NuttX RTOS workspace.
- a standalone **micro-ROS module for Microsoft Azure RTOS**: this package enables the integration of micro-ROS in a Microsoft Azure RTOS workspace.
- a set of **micro-ROS utils for STM32CubeMX and STM32CubeIDE**: this package enables the integration of micro-ROS in STM32CubeMX and STM32CubeIDE.
- a precompiled set of **Arduino IDE libraries**: this package enables the integration of micro-ROS in the Arduino IDE for some hardware platforms.
- a precompiled set of **Raspberry Pi Pico SDK libraries**: this package enables the integration of micro-ROS in the Raspberry Pi Pico SDK.

7.2 Dependencies

This package targets the **ROS 2** installation. ROS 2 supported distributions are:

ROS 2 Distro	State	Branch
Crystal	EOL	`crystal`
Dashing	EOL	`dashing`
Foxy	Supported	`foxy`
Galactic	Supported	`galactic`
Rolling	Supported	`main`

Some other prerequisites needed for building a firmware using this package are:

```
sudo apt install python3-rosdep
```

Building for Android needs Latest Android NDK to be installed and the following environment variables to be set: - `ANDROID_ABI`: CPU variant, refer here for details. - `ANDROID_NATIVE_API_LEVEL`: Android platform version, refer here for details. - `ANDROID_NDK`: root path of the installed NDK.

7.3 Quick start

Download here the micro-ROS docker image that contains a pre-installed client and agent as well as some compiled examples.

7.4 Building

Create a ROS 2 workspace and build this package for a given ROS 2 distro (see table above):

```
source /opt/ros/$ROS_DISTRO/setup.bash

mkdir uros_ws && cd uros_ws

git clone -b main https://github.com/micro-ROS/micro_ros_setup.git
    src/micro_ros_setup

rosdep update && rosdep install --from-path src --ignore-src -y

colcon build

source install/local_setup.bash
```

Once the package is built, the firmware scripts are ready to run.

You can find tutorials for moving your first steps with micro-ROS on an RTOS in the micro-ROS webpage.

7.4.1 Creating micro-ROS firmware

Using the `create_firmware_ws.sh [RTOS] [Platform]` command, a firmware folder will be created with the required code for building a micro-ROS app. For example, for our reference platform, the invocation is:

```
# Creating a FreeRTOS + micro-ROS firmware workspace
ros2 run micro_ros_setup create_firmware_ws.sh freertos olimex-stm32-e407

# Creating a Zephyr + micro-ROS firmware workspace
ros2 run micro_ros_setup create_firmware_ws.sh zephyr olimex-stm32-e407
```

7.4.2 Configuring micro-ROS firmware

By running `configure_firmware.sh` command the installed firmware is configured and modified in a pre-build step. This command will show its usage if parameters are not provided:

```
ros2 run micro_ros_setup configure_firmware.sh [configuration] [options]
```

By running this command without any argument the available demo applications and configurations will be shown.

Common options available at this configuration step are: - `--transport` or `-t`: `udp`, `serial` or any hardware specific transport label - `--dev` or `-d`: agent string descriptor in a serial-like transport (optional) - `--ip` or `-i`: agent IP in a network-like transport (optional) - `--port` or `-p`: agent port in a network-like transport (optional)

Please note that each RTOS has its configuration approach that you might use for further customization of these base configurations. Visit the micro-ROS webpage for detailed information about RTOS configuration.

In summary, the supported configurations for transports are:

	FreeRTOS	Zephyr	Mbed
Olimex STM32-E407	UART, Network	USB, UART	-
ST B-L475E-IOT01A	-	USB, UART, Network	UART
Crazyflie 2.1	Custom Radio Link	-	-
Espressif ESP32	UART, WiFI UDP	-	-
ST Nucleo F446RE 1	UART	-	-
ST Nucleo F446ZE 1	UART	-	-
ST Nucleo H743ZI 1	-	UART	-
ST Nucleo F746ZG 1	UART	UART	-
ST Nucleo F767ZI 1	UART	-	-

1 Community supported, may have lack of official support

7.4.3 Building micro-ROS firmware

By running `build_firmware.sh` the firmware is built:

```
ros2 run micro_ros_setup build_firmware.sh
```

7.4.4 Flashing micro-ROS firmware

In order to flash the target platform run `flash_firmware.sh` command. This step may need some platform-specific procedure to boot the platform in flashing mode:

```
ros2 run micro_ros_setup flash_firmware.sh
```

7.5 Building micro-ROS-Agent

Using this package is possible to install a ready to use **micro-ROS-Agent**:

```
ros2 run micro_ros_setup create_agent_ws.sh
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.sh
ros2 run micro_ros_agent micro_ros_agent [parameters]
```

7.6 Contributing

As it is explained along this document, the firmware building system takes **four steps**: creating, configuring, building and flashing.

New combinations of platforms and RTOS are intended to be included in `config` folder. For example, the scripts for building a **micro-ROS** app for **Crazyflie 2.1** using **FreeRTOS** is located in `config/freertos/crazyflie21`.

This folder contains up to four scripts: - `create.sh`: gets a variable named `$FW_TARGETDIR` and installs in this path all the dependencies and code required for the firmware. - `configure.sh`: modifies and configure parameters of the installed dependencies. This step is **optional**. - `build.sh`: builds the firmware and create a platform-specific linked binary. - `flash.sh`: flashes the binary in the target platform.

Some other required files inside the folder can be accessed from these scripts using the following paths:

```
# Files inside platform folder
$PREFIX/config/$RTOS/$PLATFORM/

# Files inside config folder
$PREFIX/config
```

7.7 Purpose of the Project

This software is not ready for production use. It has neither been developed nor tested for a specific use case. However, the license conditions of the applicable Open Source licenses allow you to adapt the software to your needs. Before using it in a safety relevant setting, make sure that the software fulfills your requirements and adjust it according to any applicable safety standards, e.g., ISO 26262.

7.8 License

This repository is open-sourced under the Apache-2.0 license. See the LICENSE file for details.

For a list of other open-source components included in ROS 2 system_modes, see the file 3rd-party-licenses.txt.

7.9 Known Issues / Limitations

There are no known limitations.

If you find issues, please report them.