D3.6

Micro-ROS middleware interface Software Release Y3

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D3.6
Deliverable name	Micro-ROS middleware interface
Date	June 2020
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 D3.5 *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*, as it is the only rmw implementation found, will be used as the default one by *rmw_implementation* (*rmw_implementation* uses *rmw_fastrps_cpp* as the default rmw implementation in case it is located, otherwise it will use the first available, alphabetically sorted). This unique implementation avoids the dependency with the third-party library Poco, used for the dynamic load of rmw_implementations.

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 interface package

3 Overview to Results

This document provides links to the released software and documentation for deliverable D3.6 *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
/feature/foxy_migration	b5c7bc0	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
  - rosidl_typesupport_microxrcedds_cpp

Branch	Latest commit	ROS 2 version
/feature/foxy_migration	f54877f	foxy
dashing	e99734f	dashing
crystal	cff523d	crystal

Documentation of the project has been migrated to micro-ROS.github.io 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-build.git

Branch	Latest commit	ROS 2 version
/feature/foxy_migration	111db03	foxy
dashing	ffbd034	dashing/eloquent
crystal	2138d50	crystal

5 Annex 1: GitHub documentation (rmw)

5.1 Architecture

The *rmw-microxrcedds* library provides a definition to the API declared in *rmw_implementation*, the same used in ROS 2, and uses the same upper *rmw* as in ROS 2. It can be divided into these main components:

- Node
- Publisher
- Subscription
- Service Client
- Service Server
- Wait on subscriptions and services requests and replies.
- Guard conditions

Apart from these components, it provides a diverse set of utilities, among which:

- Initialization and shut-down functions
- Wait functionality to listen to incoming topics in subscribers and services requests and replies.
- Functions to query on ROS graph
- Memory functions
- Name checkers

The middleware used to implement these components is Micro XRCE-DDS.

5.1.1 Nodes

This component provides the upper layer, *rcl*, with the capability to create Nodes. In our case, a Node initializes a Micro XRCE-DDS participant in the Micro XRCE-DDS Agent.

In micro-ROS there is a limited amount of nodes that can be created. This limitation can be configured in the user configuration before building the application.

5.1.2 Publishers

With a node, the *rcl* can create a publisher to a topic upon user request. To create the publisher, the upper layers provide the type support for the type of the topic along with the topic name itself. The creation of a rmw_publisher creates a publisher, a datawriter and registers the topic in the Micro XRCE-DDS Agent.

In *rmw_microxrcedds* the creation of the entities can be configured to use XML or refs. The fundamental differences between these two approaches are:

- XML are generated at runtime while refs are statically pre-configured in the Micro XRCE-DDS Agent. This implies, memorywise, that refs are less memory demanding than XML representations.

- Refs reduce the payload size, so they provide an improvement in bandwidth usage compared to that implied by XML.

In this version, there is a micro-ROS-Agent package that generates the Micro XRCE-DDS Agent and its configuration automatically, using the ROS message definitions build products.

Once a publisher is created *rmw_microxrcedds* implements a publishing API. This will take a non-serialized message for a topic, serialize it using the type support provided to the publisher creation function and then push the message to the Micro XRCE-DDS Agent.

5.1.3 Subscriptions

Analogously to publishers, subscribers can be created using an already existing node. On subscription creation, a subscriber and a datareader are created in the Micro XRCE-DDS Agent, and the topic is registered.

A extremely relevant thing to notice is that topic messages are not received until a wait is performed. For more information on the reasons behind this behavior, please read the official Micro XRCE-DDS documentation along with the XRCE-DDS protocol it implements.

On each wait call, the Micro XRCE-DDS session managed by the *rmw-microxrcedds* is run for a given time, during which the messages related with subscriptions or services are handled through callbacks. These callbacks keep an history, such that multiple subscriptions or service-related requests in every wait call are allowed.

5.1.4 Waits

For message receptions, *rmw_wait* must be performed. On this call, *rmw-microxrcedds* will ask all the publications to get new data from their topics. That request uses the Micro XRCE-DDS request data function, which asks the Micro XRCE-DDS Agent for the latest messages on the topics.

After requesting data, the *rmw-microxrcedds* waits on the Micro XRCE-DDS session till a timeout occurs or at least one new topic message arrives. The subscription is flagged as pending to read data from.

Once the wait gets triggered, a take data is called from the subscription with incoming new messages. This call deserialises the data using the type support provided to the subscription upon creation and passes the deserialised data to the upper layers which will call the user callback.

5.1.5 Service requester

With a node, the *rcl* can create a service requester (also known as service) related to a pair of *request* and *reply* types. To create the service requester, the upper layers provide both type support for the service types along with the service name itself. The creation of a rmw_service creates a requester and registers the topics in the Micro XRCE-DDS Agent.

Analogously to the creation of publishers and subscribers, the creation of these entities can be configured to use XML or refs.

Once a publisher is created, *rmw_microxrcedds* implements a request/reply API. This will take a non-serialized message for a request, serialize it using the type support provided to the requester creation function and then push the message to the Micro XRCE-DDS Agent. Once the request it done, it provides functions for taking replies related to the sent request.

Similarly to what happens with subscribers, reply messages are not received until a wait is performed. For more information on the reasons behind this behavior, please read the official Micro XRCE-DDS documentation along with the XRCE-DDS protocol it implements.

5.1.6 Service replier

Analogously to a service requester, a service replier (also known as client) can be created using an already existing node. Upon service replier creation, a replier is created on the Micro XRCE-DDS Agent, and the topic is registered.

The operations of these entities are analogous to those performed by the service requesters: the API provides functionality to take the requests received and to publish responses through the Micro XRCE-DDS Agent.

5.2 Configuration

The middleware implementation uses static memory assignations. Because of this, memory assignation are upper bounded and must be configured by the user before the build process. By default, the package sets the values for all memory bounds. The upper bound is configurable by a file that sets the values during the build process. These configuration parameters can be modified using CMake arguments during build time.

- RMW_UXRCE_TRANSPORT: Sets Micro XRCE-DDS transport to use. (udp | serial | custom)
- RMW_UXRCE_IPV: Sets Micro XRCE-DDS IP version to use. (ipv4 | ipv6)
- RMW_UXRCE_CREATION_MODE: Sets creation mode in Micro XRCE-DDS. (xml | refs)
- RMW_UXRCE_STREAM_HISTORY: This value sets the number of MTUs to buffer. Micro XRCE-DDS client configuration provides their size
- RMW_UXRCE_MAX_HISTORY: This value sets the number of history slots available for RMW subscriptions, requests and replies
- RMW_UXRCE_MAX_SESSIONS: This value sets the maximum number of Micro XRCE-DDS sessions.
- RMW_UXRCE_MAX_NODES: This value sets the maximum number of nodes.
- RMW_UXRCE_MAX_PUBLISHERS: This value sets the maximum number of publishers for an application.
- RMW_UXRCE_MAX_SUBSCRIPTIONS: This value sets the maximum number of subscriptions for an application.
- RMW_UXRCE_MAX_SERVICES: This value sets the maximum number of services for an application.
- RMW_UXRCE_MAX_CLIENTS: This value sets the maximum number of clients for an application.
- RMW_UXRCE_NODE_NAME_MAX_LENGTH: This value sets the maximum number of characters for a node name.

- RMW_UXRCE_TOPIC_NAME_MAX_LENGTH: This value sets the maximum number of characters for a topic name.
- RMW_UXRCE_TYPE_NAME_MAX_LENGTH: This value sets the maximum number of characters for a type name.
- RMW_UXRCE_XML_BUFFER_LENGTH: This value sets the maximum number of characters for a XML buffer.
- RMW_UXRCE_REF_BUFFER_LENGTH: This value sets the maximum number of characters for a reference buffer.

5.3 API guide

```
1 rmw_ret_t rmw_init(const rmw_init_options_t * options, rmw_context_t * context)
```

Initialize all required parameters and functionalities for using the rmw_microxrcedds. It must be called before calling anything else.

options RMW options.

context RMW context.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
1 rmw_node_t * rmw_create_node(rmw_context_t * context, const char * name, const char * namespace_,
      size_t domain_id, bool localhost_only)
```

Create a new node. A new participant with the Micro XRCE-DDS Agent will be opened.

name Node name.

namespace namespace.

domain_id Domain id.

localhost_only ROS2 localhost usage only. This parameter is not used.

Return Pointer to the owner node. If error, a NULL value is returned.

```
1 rmw_ret_t rmw_destroy_node(rmw_node_t * node)
```

Destroy node. The associated session with Micro XRCE-DDS Agent will be closed.

node Pointer to the owner node.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
const rmw_guard_condition_t * rmw_node_get_graph_guard_condition(const rmw_node_t * node)
```

No implemented yet.

```
rmw_publisher_t * rmw_create_publisher(const rmw_node_t * node, const rosidl_message_type_support_t
    * type_support, const char * topic_name, const rmw_qos_profile_t * qos_policies, const
    rmw_publisher_options_t * publisher_options)
```

Create a new publisher. Required entities for the publisher in the Micro XRCE-DDS agent will be created.

node Pointer to the owner node.

type_support Used type support.

topic_name Pointer to topic name string.

qos_policies Pointer to quality of service pointer.

publisher_options ROS2 related publisher options

Return Pointer to the created publisher. If error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_publisher(rmw_node_t * node, rmw_publisher_t * publisher)
```

Destroy an existing publisher. All created entities in the Micro XRCE-DDS Agent will be destroyed.

node Pointer to the owner node.

publisher Pointer to the publisher to destroy.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_publish(const rmw_publisher_t * publisher, const void * ros_message,
    rmw_publisher_allocation_t * allocation)
```

Publish a message using an existing publisher.

publisher Existing publisher pointer.

ros_message Pointer to message to publish.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_publish_serialized_message(const rmw_publisher_t * publisher,  const
    rmw_serialized_message_t * serialized_message,  rmw_publisher_allocation_t * allocation)```

No implemented yet.

```C
rmw_subscription_t * rmw_create_subscription(const rmw_node_t * node,  const
    rosidl_message_type_support_t * type_support,  const char * topic_name,  const rmw_qos_profile_t
    * qos_policies,  const rmw_subscription_options_t * subscription_options)
```

Create a new subscription. Required entities for the publisher in the Micro XRCE-DDS agent will be created.

node Pointer to the owner node.

type_support Used type support.

topic_name Pointer to topic name string.

qos_policies Pointer to quality of service polices.

subscription_options ROS2 related subscriber options

Return Pointer to the created subscription. In error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_subscription(rmw_node_t * node, rmw_subscription_t * subscription)
```

Destroy an existing subscription. All created entities in the Micro XRCE-DDS Agent will be destroyed.

node Pointer to the owner node.

subscription Pointer to the subscription to destroy.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_take(  const rmw_subscription_t * subscription,  void * ros_message,  bool * taken,
    rmw_subscription_allocation_t * allocation)
```

Take a received subscription message.

subscription Pointer to the subscription object.

ros_message Pointer where the message will be stored.

taken Indicates if the message has been taken.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
```

- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_take_with_info(const rmw_subscription_t * subscription, void * ros_message, bool *
    taken, rmw_message_info_t * message_info, rmw_subscription_allocation_t * allocation)
```

Take a received subscription message with info.

subscription Pointer to the subscription object.

ros_message Pointer where the message will be stored.

taken Indicates if the message has been taken.

message_info This parameter is not used.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_take_serialized_message( const rmw_subscription_t * subscription,
    rmw_serialized_message_t * serialized_message, bool * taken, rmw_subscription_allocation_t *
    allocation)
```

No implemented yet.

```
rmw_ret_t rmw_take_serialized_message_with_info(const rmw_subscription_t * subscription,
    rmw_serialized_message_t * serialized_message, bool * taken, rmw_message_info_t * message_info,
     rmw_subscription_allocation_t * allocation)
```

No implemented yet.

```
rmw_client_t * rmw_create_client(const rmw_node_t * node, const rosidl_service_type_support_t *
    type_support, const char * service_name, const rmw_qos_profile_t * qos_policies)
```

Create a new client or service requester. Required entities for the publisher in the Micro XRCE-DDS agent will be created.

node Pointer to the owner node.

type_support Used type support.

service_name Pointer to service name string.

qos_policies Pointer to quality of service polices.

Return Pointer to the created client. In error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_client(rmw_node_t * node, rmw_client_t * client)
```

Destroy an existing client. All created entities in the Micro XRCE-DDS Agent will be destroyed.

node Pointer to the owner node.

client Pointer to the client to destroy.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_send_request(const rmw_client_t * client, const void * ros_request, int64_t *
    sequence_id)
```

Sends a request using an existing client.

client Existing client pointer.

ros_message Pointer to request to send.

sequence_id Returned request identificator used for match with a future response.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_take_response(const rmw_client_t * client, rmw_request_id_t * request_header, void *
    ros_response, bool * taken)
```

Takes a response using an existing client.

client Existing client pointer.

request_header Returned reponse related info.

ros_response Pointer to response to be taken.

taken Returned bool indicating if response was taken.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_service_server_is_available(const rmw_node_t * node, const rmw_client_t * client, bool
    * is_available)
```

No implemented yet.

```
rmw_service_t * rmw_create_service(const rmw_node_t * node, const rosidl_service_type_support_t *
    type_support, const char * service_name, const rmw_qos_profile_t * qos_policies)
```

Create a new server or service replier. Required entities for the publisher in the Micro XRCE-DDS agent will be created.

node Pointer to the owner node.

type_support Used type support.

service_name Pointer to service name string.

qos_policies Pointer to quality of service polices.

Return Pointer to the created server. In error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_service(rmw_node_t * node, rmw_service_t * service)
```

Destroy an existing service. All created entities in the Micro XRCE-DDS Agent will be destroyed.

node Pointer to the owner node.

service Pointer to the service to destroy.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_take_request(const rmw_service_t * service, rmw_request_id_t * request_header, void *
    ros_request, bool * taken)
```

Takes a request using an existing service server.

service Existing service pointer.

request_header Returned request related info.

ros_request Pointer to request to be taken.

taken Returned bool indicating if request was taken.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_send_response(const rmw_service_t * service, rmw_request_id_t * request_header, void *
    ros_response)
```

Sends a response using an existing service.

service Existing service pointer.

request_header Reponse related info.

ros_response Pointer to response to send.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_serialize(const void * ros_message, const rosidl_message_type_support_t * type_support,
    rmw_serialized_message_t * serialized_message)
```

No implemented yet.

```
rmw_ret_t rmw_deserialize(const rmw_serialized_message_t * serialized_message, const
    rosidl_message_type_support_t * type_support, void * ros_message)
```

No implemented yet.

```
rmw_guard_condition_t * rmw_create_guard_condition(rmw_context_t * context)
```

Create a new guard condition.

Return Created guard condition pointer. If error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_guard_condition(rmw_guard_condition_t * guard_condition)
```

Destroy a guard condition.

guard_condition Pointer to guard condition to destroy.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_trigger_guard_condition(const rmw_guard_condition_t * guard_condition)
```

Triggers a guard condition.

guard_condition Pointer to guard condition to trigger.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_wait_set_t * rmw_create_wait_set(rmw_context_t * context, size_t max_conditions)
```

Create a new wait set.

Return: Created wait set pointer. If error, a NULL pointer is returned.

```
rmw_ret_t rmw_destroy_wait_set(rmw_wait_set_t * wait_set)
```

Destroy a wait set.

wait_set Pointer to wait set to destroy.

Return:

- RMW_RET_OK

- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_wait(rmw_subscriptions_t * subscriptions, rmw_guard_conditions_t * guard_conditions,
    rmw_services_t * services, rmw_clients_t * clients, rmw_wait_set_t * wait_set, const rmw_time_t *
    wait_timeout)
```

Waits until one of the given events occurs.

subscriptions Pointer to subscriptions list to wait. Set to NULL to ignore.

guard_conditions Pointer to guard conditions list to wait. Set to NULL to ignore. This parameter not supported yet.

services Pointer to services list to wait. Set to NULL to ignore. This parameter not supported yet.

clients Pointer to clients list to wait. Set to NULL to ignore. This parameter not supported yet.

wait_set Pointer to wait set list to wait. Set to NULL to ignore. This parameter not supported yet.

wait_timeout Max time to wait until a timeout occurs.

Return:

- RMW_RET_OK
- RMW_RET_ERROR
- RMW_RET_TIMEOUT

```
rmw_ret_t rmw_get_node_names(const rmw_node_t * node,  rcutils_string_array_t * node_names,
    rcutils_string_array_t * node_namespaces)
```

No implemented yet.

```
rmw_ret_t rmw_count_publishers(const rmw_node_t * node, const char * topic_name, size_t * count)
```

No implemented yet.

```
rmw_ret_t rmw_count_subscribers(const rmw_node_t * node, const char * topic_name, size_t * count)
```

No implemented yet.

```
rmw_ret_t rmw_get_gid_for_publisher(const rmw_publisher_t * publisher, rmw_gid_t * gid)
```

No implemented yet.

```
rmw_ret_t rmw_compare_gids_equal(const rmw_gid_t * gid1, const rmw_gid_t * gid2, bool * result)
```

No implemented yet.

```
rmw_ret_t rmw_set_log_severity(rmw_log_severity_t severity)
```

No implemented yet.

```
rmw_ret_t rmw_get_topic_names_and_types(const rmw_node_t * node, rcutils_allocator_t * allocator,
    bool no_demangle, rmw_names_and_types_t * topic_names_and_types)
```

No implemented yet.

```
rmw_ret_t rmw_get_service_names_and_types(const rmw_node_t * node, rcutils_allocator_t * allocator,
    rmw_names_and_types_t * service_names_and_types)
```

No implemented yet.

5.4 Roadmap

Here we will enumerate all the planned features that will be released in the future.

- Graph support.
- Full static memory allocation support

6 Annex 2: GitHub documentation (type support)

In Micro-ROS, as in regular ROS2, message types are defined using .msg files. During Micro-ROS compilation the .msg files are used to generate a type support library with all the needed code to create, serialise and deserialise message types.

This process is automatic, and the only interaction required from the user is to create the message definition files and to use the generated code and utilities provided by rclc.

6.1 Architecture

For micro-ROS, the type support used is *rosidl_typesupport_microxrcedds_c* for C language. This ROS2 package generates Micro XRCE-DDS and Micro-CDR specific code to handle all the message types.

In the case of micro-ROS, Micro XRCE-DDS type support generates four functions:

- cdr_serialize
- cdr_deserialise
- get_serialized_size
- max_serialized_size

Those four functions are equivalent to the ones generated by Micro XRCE-DDS Gen tool from Micro XRCE-DDS implementation. In Micro-ROS, *rosidl_typesupport_microxrcedds_c* generates code straight from .msg files without using intermediate IDLs nor Micro XRCE-DDS Gen. This generation uses a group of EmPy templates you can find in the *rosidl_typesupport_microxrcedds_c* package.

6.2 API guide

6.2.1 Message type support

The generated message type support has a struct that contains the API used to handle the message serialisation and deserialisation.

```
const char * package_name_
```

Package name where the generated message code belongs.

```
const char * message_name_;
```

Message name

```
bool cdr_serialize(const void * untyped_ros_message, ucdrBuffer * cdr)
```

Serialize the message into a micro cdr buffer.

untyped_ros_message Message pointer to serialize.

cdr Micro buffer.

return True if successful.

```
bool cdr_deserialize(ucdrBuffer * cdr, void * untyped_ros_message, uint8_t * raw_mem_ptr, size_t
    raw_mem_size)
```

Deserialize the serialized message contained into the micro cdr buffer.

cdr Micro buffer.

untyped_ros_message Message struct pointer where the message will be deserialised.

raw_mem_ptr Byte memory pointer used to deserialise unbounded message data. If NULL, unbounded data will be not deserialised.

raw_mem_size Byte memory size.

return True if successful.

```
uint32_t get_serialized_size(const void *);
```

Get the se rialized message size.

return Size.

```
size_t max_serialized_size(bool full_bounded);
```

Get the maximum serialized message size.

6.2.2 Service type support

The generated service type support has a struct that contains the API used to handle the service. This API will provide a message type support for the client request message and message type support for the service request message.

```
const char * package_name_;
```

Package name where the generated message code belongs.

```
const char * service_name_;
```

Service name.

```
const rosidl_message_type_support_t * request_members_();
```

Message type support for the request.

return Type support pointer.

```
const rosidl_message_type_support_t * response_members_();
```

Message type support API for the message response.

return Type support pointer.

6.3 Code generation for C type-support

The starting point of the type support generation is a set of .msg files. These files are the input for *rosidl_generator_c* ROS2 package which processes them and starts the type support code generation during the applications build.

rosidl_generator_c generates the *.h* that the user should include to use its type. The code files generated by *rosidl_generator_c* provide you with the .msg definition in C along with the type support for it.

This type support, in this case, is tied with the code generated by *rosidl_typessuport_c*. *rosidl_typessuport_c* is another ROS2 package which generates functions to get the underlying middleware type support for each type. In our case, as we support *rosidl_typesupport_microxrcedds_c*, *rosidl_typessuport_c* generates a function which makes the association between the type and the type support used for Micro XRCE-DDS.

rosidl_typesupport_microxrcedds_c ROS2 package generates middleware-specific functions per each .msg found. This middleware-specific functions in our case include serialisation and deserialisation using Micro-CDR as in any other Micro XRCE-DDS application. Those functions get stored in a *rosidl_message_type_support_t* C structure which is used in the ROS2 messages serialisation and deserialisation later on.

As a result of this process, you will end up having a type representation and type support for it. The former should be used to represent data of that type, whereas the latter encode the structures which provide information to the publishers and subscribers on how to handle (serialise/deserialise) the type.

6.3.1 Usage

To use any type, you need to include the generated *.h* which provides you with enough utilities to create messages of that type. Also, it would help if you had a way to get the type support

used by this type, and for this purpose you have to get the type support functions. For simplicity, the rclc provides a macro to help you get the right function names which give you the required *rosidl_message_type_support_t* tied to your type.

The *RCLC_GET_MSG_TYPE_SUPPORT* macro obtains the associated *rosidl_message_type_support_t* autogenerated from the msg using *rosidl_generator_c*, *rosidl_typessuport_c* and *rosidl_typesupport_microxrcedds_c* ROS2 packages.

6.4 Usage examples

In this section, an example of how to generate a simple message will be described.

6.4.1 package.xml file

The ROS2 package manifest is an XML file called package.xml that must satisfy the following requirements:

- It must be a *member_of_group* of the *rosidl_interface_packages* group.
- It must be *buildtool_depend* dependent of the *rosidl_default_generators*

```
...
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
...
```

6.4.2 Message.msg file

For this example, the ROS2 message will be as described below.

```
int32 data
```

6.4.3 CMakeList.txt file

Finds the rosidl_default_generators package with REQUIRED rule.

```
find_package(rosidl_default_generators REQUIRED)
```

Call the *rosidl_generate_interfaces* macro giving the message relative path, the message dependencies and the name of the generation target.

```
rosidl_generate_interfaces(
    ${PROJECT_NAME}
    "Message.msg"
    DEPENDENCIES "builtin_interfaces"
    ADD_LINTER_TESTS
)
```