D4.1

Micro-ROS Client library Software Release Y1

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D4.1
Deliverable name	Micro-ROS Client library
Date	December 2018
Dissemination level	public
Workpackage and task	4.1
Author	Borja Outerelo Gamarra (eProsima)
Contributors	Javier Moreno (eProsima)
Keywords	micro-ROS, robotics, ROS, microcontrollers, rclc, client library
Abstract	This document provides links to the released software and documentation for deliverable D4.1 *Micro-ROS client library Software Release Y1* of the Task 4.1 *micro-ROS client library* (*urcl*).

Contents

1 Summary

micro-ROS rclc is the ROS client library for C language. This client library provides you with the tools to interact with other ROS2 systems, from within a C program.

Currently it provides a subset of the ROS concepts compared to other user libraries as rclcpp and rclpy.

2 Acronyms and keywords

Term	Definition
RCLC	ROS Client Library for the C language.
RCLCPP	ROS Client Library for the Cpp language.
RCLPY	ROS Client Library for the pyhton language.
ROS2	Robot Operating System

3 Overview to Results

This document provides links to the released software and documentation for deliverable D4.1 *Micro-ROS client library Software Release Y1* of the Task 4.1 *micro-ROS client library* (*urcl*).

4 Links to Software Repositories

The Micro-ROS client library package is provided as a package of ROS2:

- Git repository: https://github.com/microROS/rclc
  Package name: rclc
  Package path: ./rclc
  Commit: f0816ba3e1e60663d7d79700366d0f88a8cf4262

Package documentation:

- Git repository: https://github.com/microROS/micro-ROS-doc
  Path: ./rclc
  Commit: d7864d8073a3645a950a63a9dc9b764f34ee4d77

5 Annex 1: GitHub documentation

5.1 Architecture

rclc is composed of four main C structures:

1. rclc_node_t:

Represents a ROS node. This structure holds the rcl_node along with the list of subscriptions created by the user.

1. rclc_publisher_t.

Represents a ROS publisher. It holds the rcl_publisher instance and keeps track of the node which created it.

1. rclc_subscription_t.

Represents a ROS subscription. Composed by the rcl_subscription along with the message type_support for the subscribed topic. It contains a pointer to the user callback that is used on every new message on the topic.

1. rclc_message_type_support_t.

Used to get the type required functions like serialisation/deserialisation along with the type representation in C language.

This type-support is generated from .msg files tying the functionality to the specific middleware used. To obtain the implementation you want, rclc provides with a utility macro: *RCLC_GET_MSG_TYPE_SUPPORT*.

Message type support is composed by the size of the concrete type used and the concrete middleware type-support for this type. To get this concrete type information we use a type-support system described in detail here.

5.1.1 Building blocks: Nodes, Subscriptions, Publishers and Type-support

The user application can be built using rclc main building blocks: Nodes, Subscriptions, Publishers and the type-support mechanism.

Nodes are the main object which the user needs to create first. From this object, subscriptions and publishers can be created. Nodes handle subscriptions and trigger user callbacks.

Subscriptions and publishers interact using ROS messages. These messages are interchanged using different named topics. Those ROS messages types are supported thanks to the type-support method described here.

5.2 API guide

5.2.1 rclc

```
rclc_ret_t rclc_init(int argc, char const * const * argv)
```

A function that initialises all micro-ROS stack in a cascade way, it initialises rcl and rcl initialises rmw. Arguments are passed down to the rcl layer. This function should be called before any other one.

argc Init args count.

argv Init args pointer.

return rcl response

```
bool rclc_ok(void)
```

Checks the status of rcl initialization.

return True if OK.

5.2.2 Node

```
rclc_node_t * rclc_create_node(const char * name, const char * namespace_)
```

Creates a ROS node. It initializes lower layer nodes, rcl_node and rmw_node.

name Pointer to the node name string.

_namespace__ Pointer to the namespace string.

return

```
rclc_ret_t rclc_destroy_node(rclc_node_t * node)
```

Removes a ROS node and all its related publishers and subscribers.

node The pointer to the node to be destroyed.

return rcl response

```
void rclc_spin_node_once(rclc_node_t * node, int64_t time_out_ms)
```

This function listens to the ROS subscriptions from the ROS node for a given time. It triggers user callback calls with ROS messages received on subscriptions.

node Pointer to the node to spin.

time_out_ms Spin timeout.

```
void rclc_spin_node(rclc_node_t * node)
```

Listens to the ROS subscriptions from the ROS node until the user closes the application. It triggers user callback calls with ROS messages received on subscriptions.

node Pointer to the node to spin.

5.2.3 Publisher

```
rclc_publisher_t * rclc_create_publisher(rclc_node_t * node, const rclc_message_type_support_t
    type_support, const char * topic_name, size_t queue_size)
```

The rclc_create_publisher function creates a ROS publisher within the given node. The created publisher publish on the ROS topic *topic_name* using *type_support* as the type for the ROS messages.

node Pointer to the owner node.

type_support Pointer to the message type support.

topic_name Pointer to the topic name string.

queue_size Queue size.

return Pointer to the created publisher.

```
rclc_ret_t rclc_destroy_publisher(rclc_publisher_t * publisher)
```

Removes the publisher.

publisher Pointer to the publisher to be destroyed.

return rcl response

```
rclc_ret_t rclc_publish(const rclc_publisher_t * publisher, const void * ros_message)
```

Publish a ROS message using the ROS topic tied to the *publisher*.

publisher Pointer to the publisher.

ros_message Pointer to the ROS message.

return rcl response

5.2.4 Subscription

```
rclc_subscription_t * rclc_create_subscription(rclc_node_t * node, const rclc_message_type_support_t
    type_support, const char * topic_name, rclc_callback_t callback, size_t queue_size,bool
    ignore_local_publications)
```

Creates a ROS subscription within the given node. The created subscription listens on the ROS topic *topic_name* using *type_support* as the type for the ROS messages. Publications on *topic_name* trigger call to the user *callback* provided. *rclc_spin_node* functions trigger those calls.

node Pointer to the owner node.

type_support Pointer to the message type support.

topic_name Pointer to the topic name string.

callback Pointer to the function that will be called when incoming new message.

queue_size Queue size.

ignore_local_publications Set to true to ignore local publications.

return Pointer to the created subscriber.

```
1 rclc_ret_t rclc_destroy_subscription(rclc_subscription_t * subscription)
```

Removes ROS subscription.

subscription Pointer to the subscription to be destroyed.

return rcl response

5.2.5 Type-support

```
1 RCLC_GET_MSG_TYPE_SUPPORT(pkg, dir, msg)
```

Creates a ROS message type-support for the type pkg::dir::msg. This type-support has been created at compile time using user messages definitions (.msg) as sources.

pkg Package name.

dir Type support directory.

msg Message name.

return Pointer to the type support.

5.2.6 Executor

```
1 rclc_executor_t * rclc_create_executor()
```

Creates a rclc_executor_t, which processes callbacks in ROS Nodes.

return Created executor pointer.

```
1 rclc_ret_t rclc_destroy_executor(rclc_executor_t * executor)
```

Destroys a rclc_executor_t.

executor Pointer to the executor to be destroyed.

return rcl response

```
rclc_ret_t rclc_executor_add_node(rclc_executor_t * executor, const rclc_node_t * node)
```

Adds a ROS Node to the internal list of nodes in the executor.

executor Pointer to the owner executor.

node Pointer to the node to be added.

return rcl response

```
rclc_ret_t rclc_executor_spin()
```

Causes the executor to process callbacks generated by ROS Nodes.

return rcl response

5.3 Usage example

This section describes an example of how to make a simple publisher and subscriber.

5.3.1 Publisher

Step 1.

Init the rclc layer.

```
size_t args_count = 0;
void * args_pointer = NULL;
rclc_ret_t init_response;
init_response = rclc_init(args_count, args_pointer);
if (init_response != RCL_RET_OK)
{
    // Error
}
```

Step 2 Create a new node.

```
const char * node_name = "Example_publisher"
const char * node_namespace = ""
rclc_node_t* node     = rclc_create_node(node_name, node_namespace);
```

Step 3 Get the message type support for an int32 message.

```
const rclc_message_type_support_t type_support;
type_support = RCLC_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32)
```

Step 4 Create a new publisher in the previously-created node.

```
const char* topic_name = "publish_example"
rclc_publisher_t* publisher;
publisher = rclc_create_publisher(node, type_support, topic_name, 1);
if (publisher == NULL)
{
    // Error
}
```

Step 5 Publish each second.

```
std_msgs__msg__Int32 msg;
msg.data = 0;
rclc_ret_t publish_response;
 while (rclc_ok())
{
    publish_response = rclc_publish(publisher, (const void*)&msg);
    if (publish_response != RCL_RET_OK)
    {
        // Error
    }
    rclc_spin_node_once(node, 1000);
}
```

5.3.2 Subscriber

Step 1.

Init the rclc layer.

```
size_t args_count = 0;
void * args_pointer = NULL;
rclc_ret_t init_response;
init_response = rclc_init(args_count, args_pointer);
if (init_response != RCL_RET_OK)
{
    // Error
}
```

Step 2 Create a new node.

```
const char * node_name = "Example_subscriber"
const char * node_namespace = ""
rclc_node_t* node     = rclc_create_node(node_name, node_namespace);
```

Step 3 Get the message type support for an int32 message.

```
const rclc_message_type_support_t type_support;
type_support = RCLC_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32)
```

Step 4 Create a new subscription in the created node.

```
const char* topic_name = "subscriber_example"
rclc_subscription_t* subscription;
subscription = rclc_create_subscription(node, type_support, topic_name, on_message_callback, 1,
    false);
if (subscription == NULL)
{
    // Error
}
```

Note: The callback function must be defined following the underneath signature. This function will be called every time a new message arrives.

```
void on_message_callback(const void* msgin)
{
    const std_msgs__msg__Int32* msg = (const std_msgs__msg__Int32*)msgin;
    // message usage
}
```

Step 5 Spin the node.

```
rclc_spin_node(node);
```

5.4 Roadmap

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

- Services support.
- Graph support.