Micro-ros tutorial

I. **Building micro-ros agent** 2

II. **Running micro-ros on ESP32 (Arduino framework)** 3

III. **Cross compile custom micro-ros static library for PC** 4

IV. **Cross compile custom micro-ros static library for KL730** ... 9

V. **Building Custom UDP transport** 10

VI. **Create a micro-ros project** 13

VII. **Content code** 16

I. Building micro-ros agent

1. Building micro-ros workspace

```
# create and run docker container
# --net=host : net configuration in container behave as host
# -v /dev:/dev : mount the connected devices into container
# --name microros : container names microros
# ros:humble : a ros2 docker image provide by ros, version humble
$ docker run -it --net=host -v /dev:/dev --privileged --name microros ros:humble

# inside the docker container
# source the ros2 command
$ source /opt/ros/$ROS_DISTRO/setup.bash

# create microros workspace folder
$ mkdir microros_ws && cd microros_ws

# clone micro-ros tools
$ git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git src/micro_ros_setup

# update dependencies using rosdep
$ sudo apt update && rosdep update
$ rosdep install --from-paths src --ignore-src -y
$ sudo apt-get install python3-pip

# build micro-ros tools and source them
$ colcon build
$ source install/local_setup.bash
# exit docker container
$ exit
```

2. Save the built micro-ros as docker image

```
# outside the docker container
# save the container names microros as a docker image
$ docker commit microros microros_base

# delete old container
$ docker rm microros

# run saved micro-ros docker container
$ docker run --rm -it --net=host -v /dev:/dev --privileged --name microros microros_base

# go to workspace folder
$ cd /microros_ws
```

3. Create agent

```
# inside the docker container
# download micro-ROS-Agent packages
$ ros2 run micro_ros_setup create_agent_ws.sh

# build agent
$ ros2 run micro_ros_setup build_agent.sh
$ source install/local_setup.bash
```

4. Running agent

```
# inside the docker container
# using udp transport and listen on port 8888
$ ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888

# using serial transport and listen on device ttyUSB0
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0

# Ctrl + c to stop agent
```

II. Running micro-ros on ESP32 (Arduino framework)

1. **Download the pre-compiled library**

   https://github.com/micro-ROS/micro_ros_arduino/tree/humble

2. **Import library**

   Sketch → Include Library → add .ZIP library

3. **Open example**

   File → Examples → micro-ros-arduino → micro-ros_publisher

4. **Flash**

   Click upload

5. **Run agent**

   # using serial transport as I.4 metioned

   # might need press reset button on esp32

III. Cross compile custom micro-ros static library for PC

1. Run micro-ros docker container

```
# run saved micro-ros docker container
$ docker run --rm -it --net=host -v /dev:/dev --privileged --name microros microros_base

# go to workspace folder
$ cd /microros_ws
```

2. Run generate library command

```
# this step will download source code
$ ros2 run micro_ros_setup create_firmware_ws.sh generate_lib
```

3. Create toolchain.cmake and colcon.meta file

```
# create file and paste the content below
$ nano toolchain.cmake
```

```
# toolchain.cmake for host PC
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_CROSSCOMPILING 0)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

set(CMAKE_C_COMPILER /usr/bin/gcc)
set(CMAKE_CXX_COMPILER /usr/bin/g++)

set(CMAKE_C_COMPILER_WORKS 1 CACHE INTERNAL "")
set(CMAKE_CXX_COMPILER_WORKS 1 CACHE INTERNAL "")

set(FLAGS "-O2 -ffunction-sections -fdata-sections" CACHE STRING "" FORCE)

set(CMAKE_C_FLAGS_INIT "-std=c11 ${FLAGS}" CACHE STRING "" FORCE)
set(CMAKE_CXX_FLAGS_INIT "-std=c++11 ${FLAGS}" CACHE STRING "" FORCE)

set(__BIG_ENDIAN__ 0)
```

$ nano colcon.meta

```
# colcon.meta for PC
{
    "names": {
        "tracetools": {
            "cmake-args": [
                "-DTRACETOOLS_DISABLED=ON",
                "-DTRACETOOLS_STATUS_CHECKING_TOOL=OFF"
            ]
        },
        "rosidl_typesupport": {
            "cmake-args": [
                "-DROSIDL_TYPESUPPORT_SINGLE_TYPESUPPORT=ON"
            ]
        },
        "rcl": {
            "cmake-args": [
                "-DBUILD_TESTING=OFF",
                "-DRCL_COMMAND_LINE_ENABLED=OFF",
                "-DRCL_LOGGING_ENABLED=OFF"
            ]
        },
        "rcutils": {
            "cmake-args": [
                "-DENABLE_TESTING=OFF",
                "-DRCUTILS_NO_FILESYSTEM=ON",
                "-DRCUTILS_NO_THREAD_SUPPORT=ON",
                "-DRCUTILS_NO_64_ATOMIC=ON",
                "-DRCUTILS_AVOID_DYNAMIC_ALLOCATION=ON"
            ]
        },
        "microxrcedds_client": {
            "cmake-args": [
                "-DUCLIENT_PIC=OFF",
                "-DUCLIENT_PROFILE_UDP=OFF",
                "-DUCLIENT_PROFILE_TCP=OFF",
                "-DUCLIENT_PROFILE_DISCOVERY=OFF",
                "-DUCLIENT_PROFILE_SERIAL=OFF",
                "-UCLIENT_PROFILE_STREAM_FRAMING=ON",
                "-DUCLIENT_PROFILE_CUSTOM_TRANSPORT=ON"
            ]
        },
        "rmw_microxrcedds": {
            "cmake-args": [
                "-DRMW_UXRCE_MAX_NODES=1",
                "-DRMW_UXRCE_MAX_PUBLISHERS=5",
                "-DRMW_UXRCE_MAX_SUBSCRIPTIONS=5",
                "-DRMW_UXRCE_MAX_SERVICES=1",
                "-DRMW_UXRCE_MAX_CLIENTS=1",
                "-DRMW_UXRCE_MAX_HISTORY=4",
                "-DRMW_UXRCE_TRANSPORT=custom"
            ]
        }
    }
}
```

4. Building custom library

```
# building library
# the built library will be place at firware/build
$ ros2 run micro_ros_setup build_firmware.sh $(pwd)/toolchain.cmake $(pwd)/colcon.meta
```

5. Copy the built library to host

```
# outside container
# open new terminal
$ docker cp microros:/microros_ws/firmware/built .
```

6. Flatten the folder structure

```
# use this python script to flatten folder structure
# python3 flatten.py --path build/include
```

```
import os
import shutil
import argparse

def flatten_one_dir(src):
    for file in os.listdir(src):
        if file == os.path.basename(src):
            tarDir = os.path.join(src, file)
            for tarFile in os.listdir(tarDir):
                if os.path.isdir(tarFile):
                    tarFileDir = os.path.join(tarDir, tarFile)
                    dstFileDir = os.path.join(src, tarFile)
                else:
                    tarFileDir = tarDir
                    dstFileDir = src
                shutil.copytree(tarFileDir, dstFileDir, dirs_exist_ok=True)
            shutil.rmtree(tarDir)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--path', type=str, default='', help='path to include')
    opt = parser.parse_args()
    PATH = opt.path
for file in sorted(os.listdir(PATH)):
    src = os.path.join(PATH, file)
    print(src)
    flatten_one_dir(src)
```

IV. Cross compile custom micro-ros static library for KL730

1. Change the toolchain.cmake file, other steps are same

```
# toolchain.cmake for KL730
set(CMAKE_SYSTEM_NAME Linux)    # Change this to your target system
set(CMAKE_CROSSCOMPILING 1)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

# Specify the compilers from the SDK
set(CMAKE_C_COMPILER /vtcs_toolchain/leipzig/usr/bin/aarch64-linux-gcc)
set(CMAKE_CXX_COMPILER /vtcs_toolchain/leipzig/usr/bin/aarch64-linux-g++)

set(CMAKE_C_COMPILER_WORKS 1 CACHE INTERNAL "")
set(CMAKE_CXX_COMPILER_WORKS 1 CACHE INTERNAL "")

# Set the sysroot for the SDK (if applicable)
set(CMAKE_SYSROOT /vtcs_toolchain/leipzig/aarch64-buildroot-linux-gnu/sysroot)

# Add the SDK library path
set(SDK_LIBRARY_PATH /vtcs_toolchain/leipzig/usr/lib)

# Add the library path to the CMake search path
link_directories(${SDK_LIBRARY_PATH})

# Optionally, set the library paths for the linker
set(CMAKE_FIND_LIBRARY_PATH ${SDK_LIBRARY_PATH})
# Set compilation flags if necessary
set(CMAKE_C_FLAGS "-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -Os -g0 -D_FORTIFY_SOURCE=1" CACHE STRING "Buildroot CFLAGS")
set(CMAKE_CXX_FLAGS "-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -Os -g0 -D_FORTIFY_SOURCE=1" CACHE STRING "Buildroot CXXFLAGS")
set(__BIG_ENDIAN__ 0)
```

V. Building Custom UDP transport

1. Overview

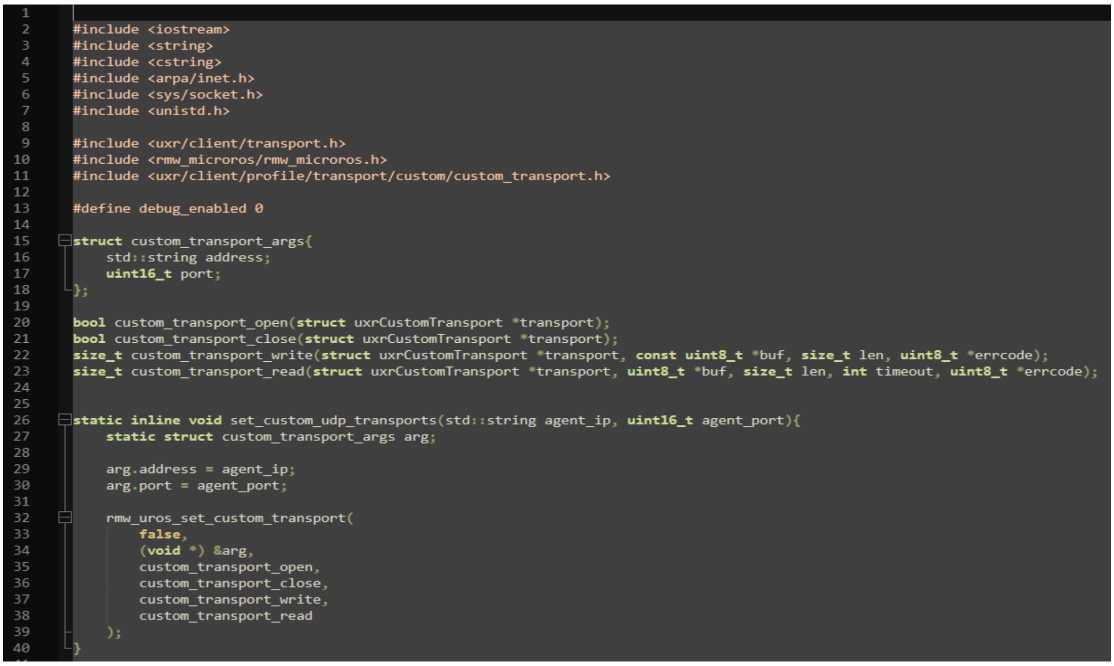
Micro-ros have micro-ros client API to let users creating their own custom transport method. User must complete these 4 functions to achieve custom transport.

```
# open transport
bool my_custom_transport_open(uxrCustomTransport* transport)
# close transport
bool my_custom_transport_close(uxrCustomTransport* transport)
# write data
size_t my_custom_transport_write(
        uxrCustomTransport* transport,
        const uint8_t* buffer,
        size_t length,
        uint8_t* errcode)
# read data
size_t my_custom_transport_read(
        uxrCustomTransport* transport,
        uint8_t* buffer,
        size_t length,
        int timeout,
        uint8_t* errcode)
```

And call this function to connect API

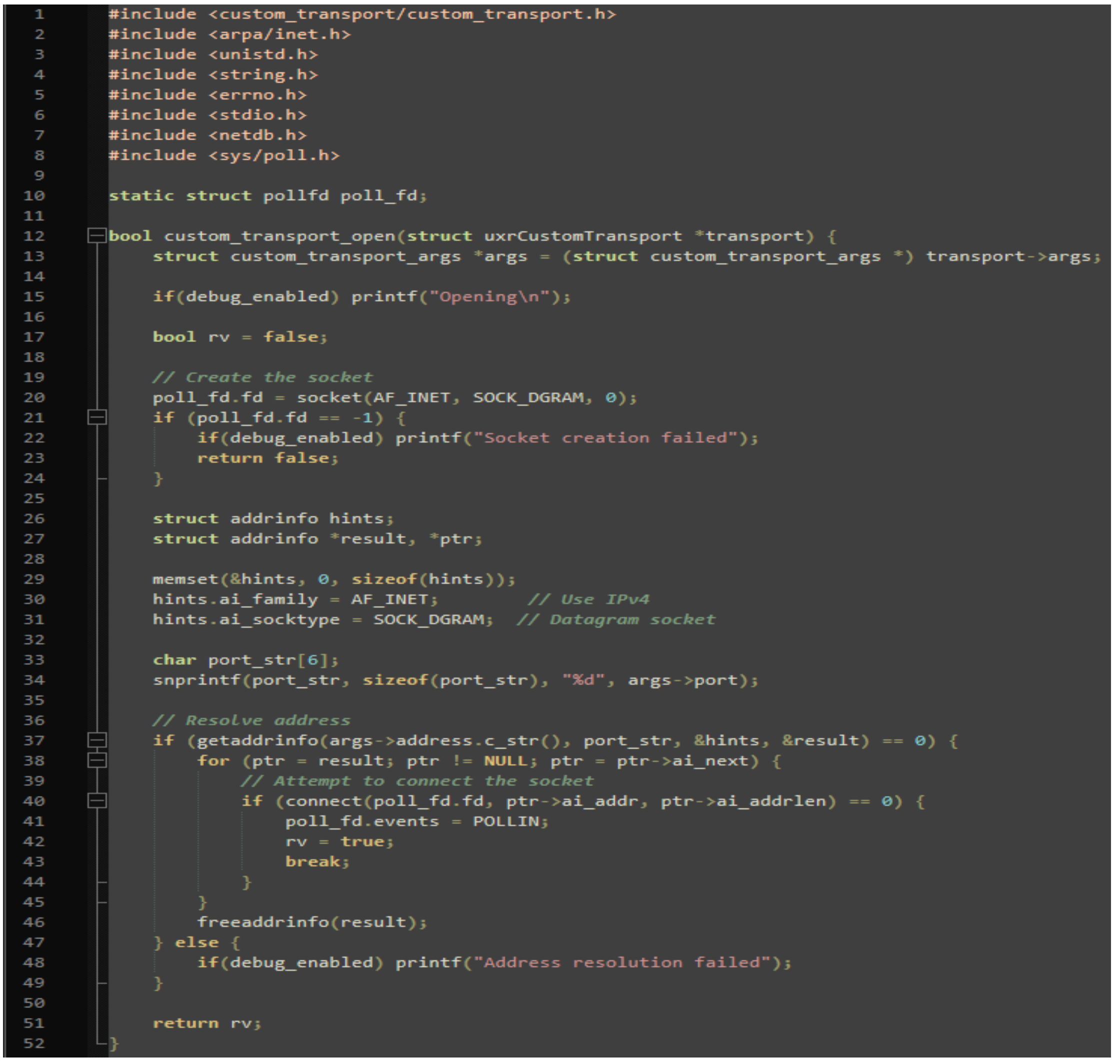
```
rmw_uros_set_custom_transport(
    true, // Framing enabled here. Using Stream-oriented mode.
    (void *) &args,
    my_custom_transport_open,
    my_custom_transport_close,
    my_custom_transport_write,
    my_custom_transport_read
);
```

2. Create custom_transport.h

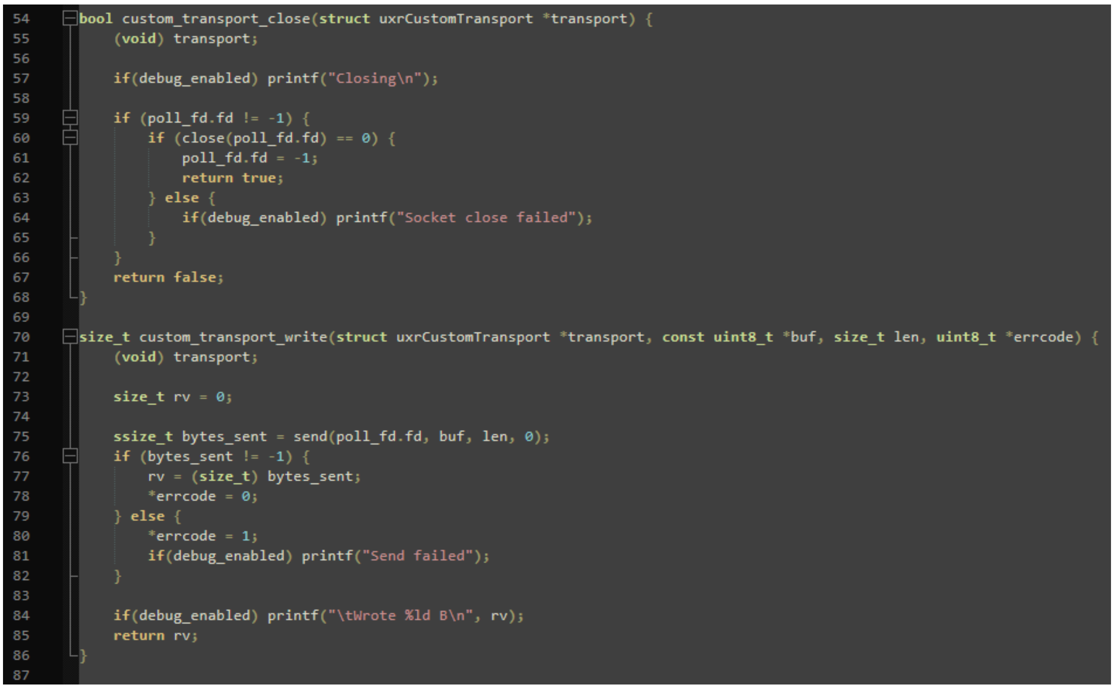


```cpp

#include <iostream>
#include <string>
#include <cstring>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#include <uxr/client/transport.h>
#include <rmw_microros/rmw_microros.h>
#include <uxr/client/profile/transport/custom/custom_transport.h>

#define debug_enabled 0

struct custom_transport_args{
    std::string address;
    uint16_t port;
};

bool custom_transport_open(struct uxrCustomTransport *transport);
bool custom_transport_close(struct uxrCustomTransport *transport);
size_t custom_transport_write(struct uxrCustomTransport *transport, const uint8_t *buf, size_t len, uint8_t *errcode);
size_t custom_transport_read(struct uxrCustomTransport *transport, uint8_t *buf, size_t len, int timeout, uint8_t *errcode);


static inline void set_custom_udp_transports(std::string agent_ip, uint16_t agent_port){
    static struct custom_transport_args arg;

    arg.address = agent_ip;
    arg.port = agent_port;

    rmw_uros_set_custom_transport(
        false,
        (void *) &arg,
        custom_transport_open,
        custom_transport_close,
        custom_transport_write,
        custom_transport_read
    );
}
```

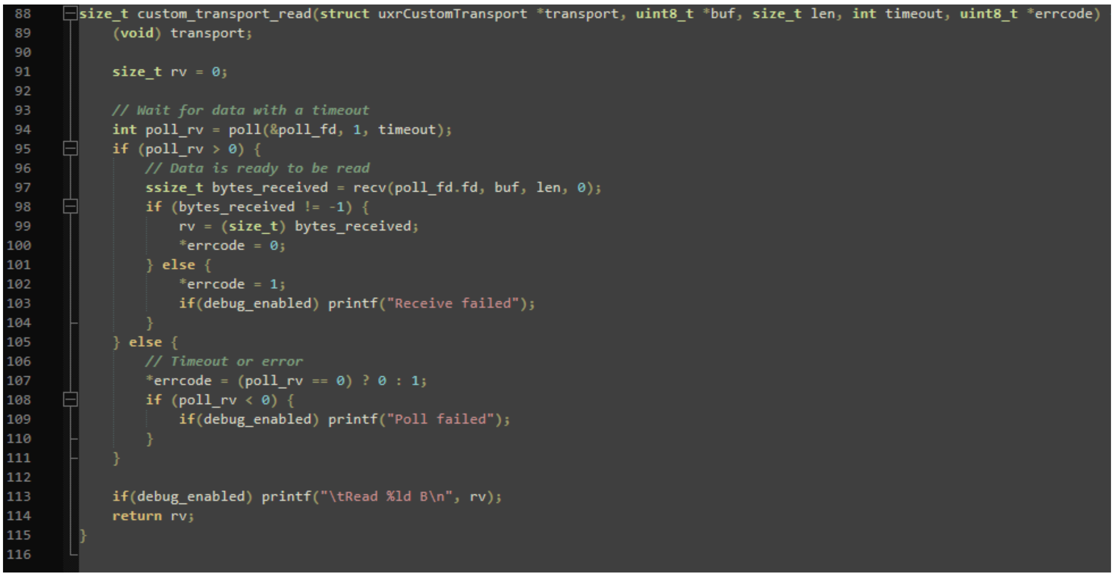
3. Create custom_transport.cpp



```cpp
#include <custom_transport/custom_transport.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <netdb.h>
#include <sys/poll.h>

static struct pollfd poll_fd;

bool custom_transport_open(struct uxrCustomTransport *transport) {
    struct custom_transport_args *args = (struct custom_transport_args *) transport->args;

    if(debug_enabled) printf("Opening\n");

    bool rv = false;

    // Create the socket
    poll_fd.fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (poll_fd.fd == -1) {
        if(debug_enabled) printf("Socket creation failed");
        return false;
    }

    struct addrinfo hints;
    struct addrinfo *result, *ptr;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;          // Use IPv4
    hints.ai_socktype = SOCK_DGRAM;  // Datagram socket

    char port_str[6];
    snprintf(port_str, sizeof(port_str), "%d", args->port);

    // Resolve address
    if (getaddrinfo(args->address.c_str(), port_str, &hints, &result) == 0) {
        for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {
            // Attempt to connect the socket
            if (connect(poll_fd.fd, ptr->ai_addr, ptr->ai_addrlen) == 0) {
                poll_fd.events = POLLIN;
                rv = true;
                break;
            }
        }
        freeaddrinfo(result);
    } else {
        if(debug_enabled) printf("Address resolution failed");
    }

    return rv;
}
```



```c
bool custom_transport_close(struct uxrCustomTransport *transport) {
    (void) transport;

    if(debug_enabled) printf("Closing\n");

    if (poll_fd.fd != -1) {
        if (close(poll_fd.fd) == 0) {
            poll_fd.fd = -1;
            return true;
        } else {
            if(debug_enabled) printf("Socket close failed");
        }
    }
    return false;
}

size_t custom_transport_write(struct uxrCustomTransport *transport, const uint8_t *buf, size_t len, uint8_t *errcode) {
    (void) transport;

    size_t rv = 0;

    ssize_t bytes_sent = send(poll_fd.fd, buf, len, 0);
    if (bytes_sent != -1) {
        rv = (size_t) bytes_sent;
        *errcode = 0;
    } else {
        *errcode = 1;
        if(debug_enabled) printf("Send failed");
    }

    if(debug_enabled) printf("\tWrote %ld B\n", rv);
    return rv;
}

```



```c
size_t custom_transport_read(struct uxrCustomTransport *transport, uint8_t *buf, size_t len, int timeout, uint8_t *errcode)
    (void) transport;

    size_t rv = 0;

    // Wait for data with a timeout
    int poll_rv = poll(&poll_fd, 1, timeout);
    if (poll_rv > 0) {
        // Data is ready to be read
        ssize_t bytes_received = recv(poll_fd.fd, buf, len, 0);
        if (bytes_received != -1) {
            rv = (size_t) bytes_received;
            *errcode = 0;
        } else {
            *errcode = 1;
            if(debug_enabled) printf("Receive failed");
        }
    } else {
        // Timeout or error
        *errcode = (poll_rv == 0) ? 0 : 1;
        if (poll_rv < 0) {
            if(debug_enabled) printf("Poll failed");
        }
    }

    if(debug_enabled) printf("\tRead %ld B\n", rv);
    return rv;
}

```

VI. Create a micro-ros project

1. Place the custom_transport into folder

```
├──── custom_transport
│    ├──── custom_transport.cpp
│    └──── custom_transport.h
```

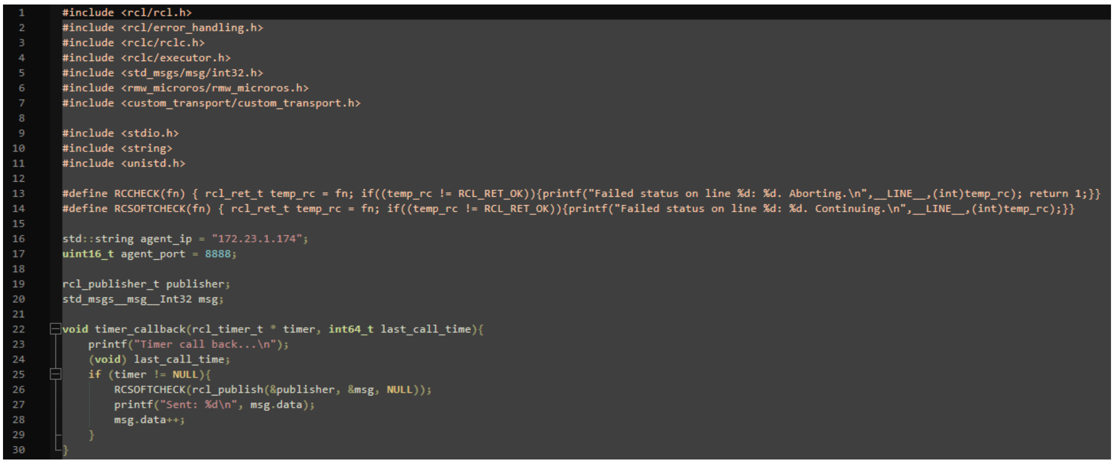
2. Rename built static library from build to libmicroros

3. Place custom_transport into libmicros/include

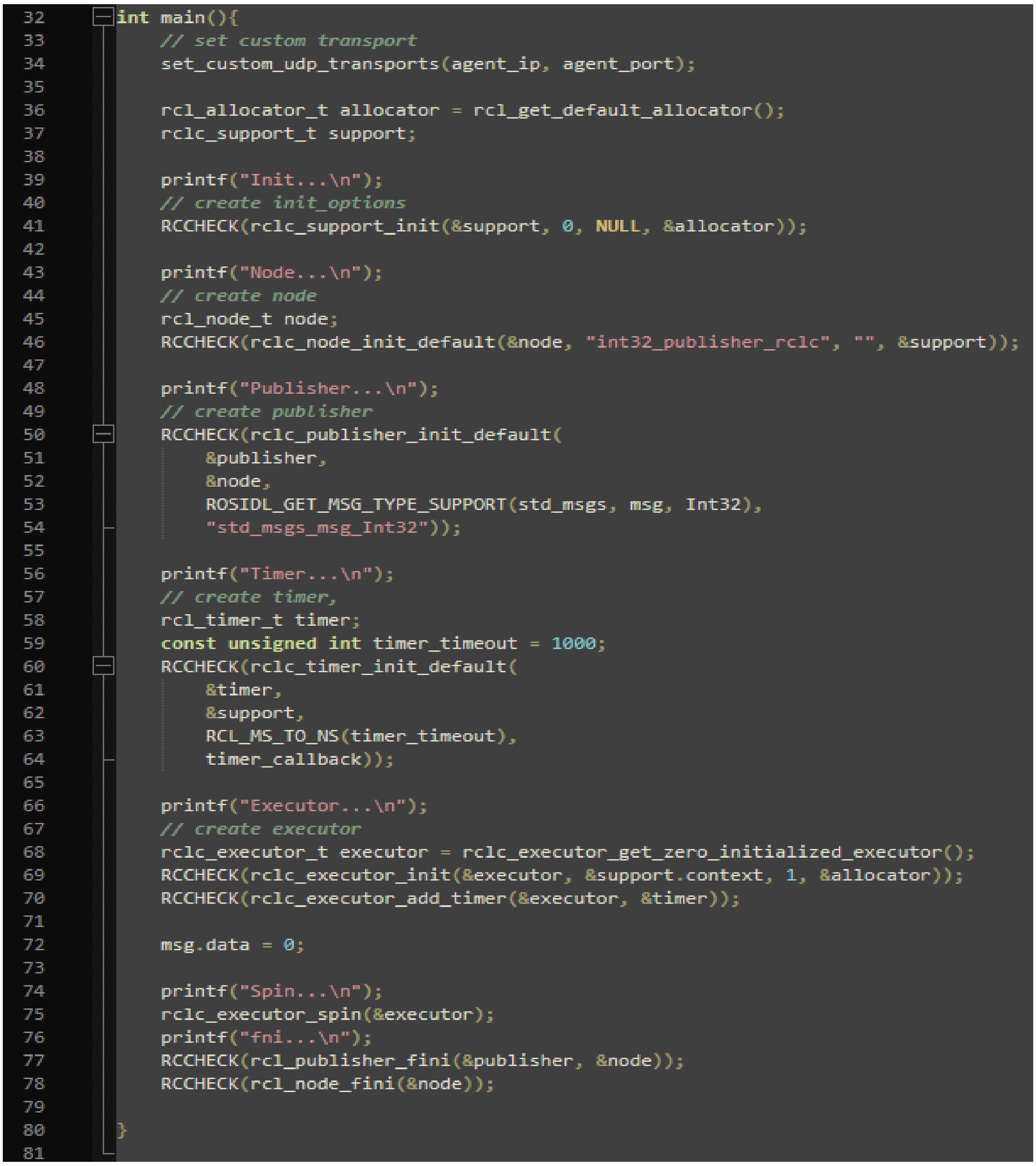
```
├──── libmicroros
│    ├──── include
│    │    ├──── actionlib_msgs
│    │    ├──── action_msgs
│    │    ├──── builtin_interfaces
│    │    ├──── composition_interfaces
│    │    ├──── custom_transport
│    │    ├──── diagnostic_msgs
│    │    ├──── example_interfaces
│    │    ├──── geometry_msgs
│    │    ├──── libyaml_vendor
│    │    ├──── lifecycle_msgs
│    │    ├──── micro_ros_msgs
│    │    ├──── micro_ros_utilities
│    │    ├──── nav_msgs
│    │    ├──── rcl
│    │    ├──── rcl_action
│    │    ├──── rclc
│    │    ├──── rclc_lifecycle
│    │    ├──── rclc_parameter
│    │    ├──── rcl_interfaces
│    │    ├──── rcl_lifecycle
│    │    ├──── rcl_logging_interface
│    │    ├──── rcutils
│    │    ├──── rmw
│    │    ├──── rmw_microros
│    │    ├──── rmw_microxrcedds_c
│    │    ├──── rosgraph_msgs
│    │    ├──── rosidl_runtime_c
│    │    ├──── rosidl_typesupport_c
│    │    ├──── rosidl_typesupport_interface
│    │    ├──── rosidl_typesupport_introspection_c
│    │    ├──── rosidl_typesupport_microxrcedds_c
│    │    ├──── sensor_msgs
│    │    ├──── shape_msgs
│    │    ├──── statistics_msgs
│    │    ├──── std_msgs
│    │    ├──── std_srvs
│    │    ├──── stereo_msgs
│    │    ├──── test_msgs
│    │    ├──── tracetools
│    │    ├──── trajectory_msgs
│    │    ├──── ucdr
│    │    ├──── unique_identifier_msgs
│    │    ├──── uxr
│    │    └──── visualization_msgs
│    └──── libmicroros.a
```

4. Write code

main.cpp

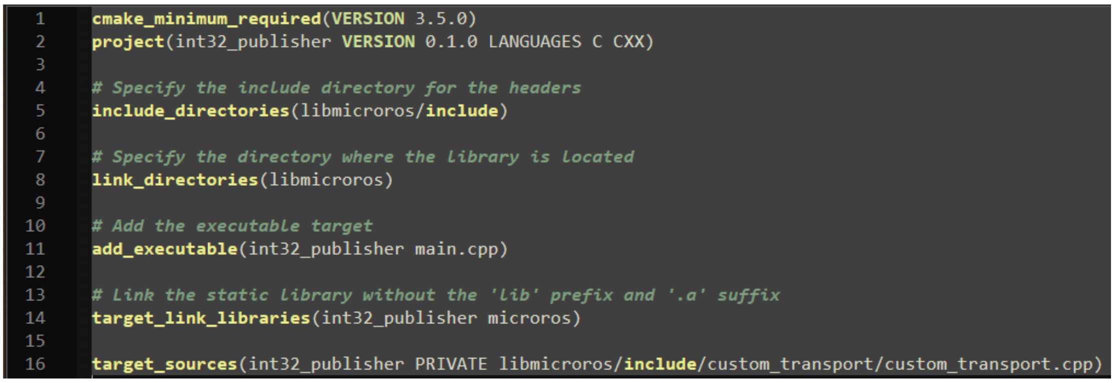


```cpp
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>
#include <rmw_microros/rmw_microros.h>
#include <custom_transport/custom_transport.h>

#include <stdio.h>
#include <string>
#include <unistd.h>

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){printf("Failed status on line %d: %d. Aborting.\n",__LINE__,(int)temp_rc); return 1;}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){printf("Failed status on line %d: %d. Continuing.\n",__LINE__,(int)temp_rc);}}

std::string agent_ip = "172.23.1.174";
uint16_t agent_port = 8888;

rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;

void timer_callback(rcl_timer_t * timer, int64_t last_call_time){
    printf("Timer call back...\n");
    (void) last_call_time;
    if (timer != NULL){
        RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
        printf("Sent: %d\n", msg.data);
        msg.data++;
    }
}
```



```cpp
int main(){
    // set custom transport
    set_custom_udp_transports(agent_ip, agent_port);

    rcl_allocator_t allocator = rcl_get_default_allocator();
    rclc_support_t support;

    printf("Init...\n");
    // create init_options
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

    printf("Node...\n");
    // create node
    rcl_node_t node;
    RCCHECK(rclc_node_init_default(&node, "int32_publisher_rclc", "", &support));

    printf("Publisher...\n");
    // create publisher
    RCCHECK(rclc_publisher_init_default(
        &publisher,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "std_msgs_msg_Int32"));

    printf("Timer...\n");
    // create timer,
    rcl_timer_t timer;
    const unsigned int timer_timeout = 1000;
    RCCHECK(rclc_timer_init_default(
        &timer,
        &support,
        RCL_MS_TO_NS(timer_timeout),
        timer_callback));

    printf("Executor...\n");
    // create executor
    rclc_executor_t executor = rclc_executor_get_zero_initialized_executor();
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    RCCHECK(rclc_executor_add_timer(&executor, &timer));

    msg.data = 0;

    printf("Spin...\n");
    rclc_executor_spin(&executor);
    printf("fni...\n");
    RCCHECK(rcl_publisher_fini(&publisher, &node));
    RCCHECK(rcl_node_fini(&node));

}
```

CMakeList.txt



```
cmake_minimum_required(VERSION 3.5.0)
project(int32_publisher VERSION 0.1.0 LANGUAGES C CXX)

# Specify the include directory for the headers
include_directories(libmicroros/include)

# Specify the directory where the library is located
link_directories(libmicroros)

# Add the executable target
add_executable(int32_publisher main.cpp)

# Link the static library without the 'lib' prefix and '.a' suffix
target_link_libraries(int32_publisher microros)

target_sources(int32_publisher PRIVATE libmicroros/include/custom_transport/custom_transport.cpp)
```

5. Project structure

```
├──── int32_publisher
│     ├──── CMakeLists.txt
│     ├──── libmicroros
│     └──── main.cpp
```

6. Compile (PC)

```
$ mkdir build && cd build

$ cmake ..

$ make
```

7. Run agent

```
# using udp transport as I.4
```

8. Run micro-ros code

```
# this step will try to connect to agent

# if agent is not opened will cause fail

$ ./int32_publisher
```

9. Check if host has received data

```
# list all ros2 topic

$ ros2 topic list


# see the value it has trasnport

$ ros2 topic echo /std_msgs_msg_Int32
```

VII. Content code

https://github.com/ZaGabriel/microros.git