Orientalmotor

ROS2 Package

Modbus RTU Node

Operating Manual

6th Edition

Contents

1 Introduction

■Modbus RTU Node

Modbus RTU Node is a node that controls products that support Modbus RTU. Modbus RTU control can be achieved just by distributing ROS messages.

■Applicable Products

Stepping motor: AZ Series (Built-In Controller Type, Pulse Input Type with RS-485 communication)
AR Series (Built-In Controller Type)
RKⅡ Series (Built-in Controller Type)
CVD Series (RS-485 Communication Type)
Brushless motor: BLH Series (RS-485 Communication Type)
BLE Series (RS-485 Communication Type)
BLV Series
BLV Series R Type

■System requirements

- OS: Ubuntu 22.04 LTS
- ROS2 Distribution: Humble Hawksbill

■How to use the applicable products

Please refer to the user's manual for the details of the applicable products.

■Connection to RS-485 communication connector

Personal computers generally do not have a terminal for RS-485 communication. Therefore, in order to perform RS-485 communication from a personal computer, RS-485 communication devices (RS-485 communication board, USB-RS-485 communication conversion cable, etc.) are required. As we do not provide RS-485 communication devices, they have to be prepared at customers. For the connection between a RS-485 communication device and our driver, please refer to the user's manual of the product.

■RS-485 Communication Parameter

Only communication ID and Baudrate can be changed for RS-485 communication parameters. All other communication parameters are used with the contents shown below.

RS-485 Communication Parameter	Setting Range
Communication ID (Modbus)	1-31 Note: 1-15 for BLH
Baudrate (Modbus)	9600, 19200, 38400, 57600, 115200, 230400 bps The 230400 bps applies AZ, CVD, BLH and BLV R type only.
Communication Order (Modbus) AZ, CVD, BLH and BLV R type only	Initial Value (0: EvenAddress-HighWord & Big-Endian)
Communication Parity	Initial Value (1: even parity)
Communication Stop Bit	Initial Value (0: 1 bit)
Communication Timeout (Modbus) [ms]	Initial Value (0: No monitoring performed)
Communication Error Alarm (Modbus)	Initial Value (3)
Transmission Waiting Time (Modbus)[ms]	Initial Value (3) AZ, CVD, BLH, BLV, R type Initial Value (10) AR, RKⅡ, BLE, BLV
Silent Interval (Modbus) [ms] AZ, CVD, BLH and BLV R type only	Initial Value (0: set automatically)
Slave error detection response (Modbus) AZ, CVD, BLH and BLV R type only	Initial Value (1: an exception response is returned)
Group ID Initial Value (Modbus) AZ, CVD, BLH and BLV R type only	Initial Value (-1: Disabled)

2 Cautions

(1) For the construction of the system, check the specifications of each equipment and apparatus that constitute the system, and use the product having sufficient margin in the rating and performance. Take safety measures such as a safety circuit to minimize the risk or danger even if a failure occurs.

(2) To ensure safe use of the system, obtain instruction manuals or operating manuals for each device and equipment that constitute the system. Check the contents related to safety including “Safety Precautions” or “Safety Summaries” prior to use.

(3) Customers are required to confirm the standards, regulations, and restrictions that the system should comply with.

(4) Copying, reproducing, or redistributing all or part of this document without permission of Oriental Motor Co., Ltd. is prohibited.

(5) The contents of this document are as of March 2023. The contents of this document are subject to change without notice for improvement.

(6) This document describes the procedures to establish communication connection of the equipment. It does not describe the operation, installation and wiring methods of each device and equipment. For details other than the procedures of communication connection, refer to the operation manuals of the applicable products.

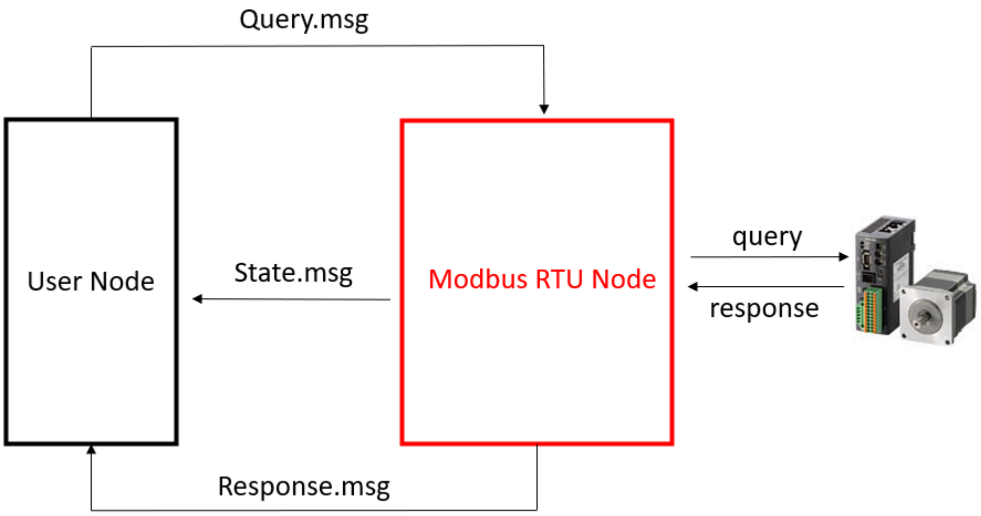
(7) This document is intended for those with ROS and Linux expertise. Please note that inquiries related to installation and usage of ROS, and Linux are not supported.

3 Preparation

3.1 Overview

Modbus RTU Node provides a wrapper from Modbus RTU communication to standardized ROS messages.
Processing is performed in the following flow.

1) User node distributes query data to Modbus RTU Node.
2) Modbus RTU Node subscribes the distributed data.
3) If the driver is communicable, the query is sent to the driver.
4) The response from the driver is analyzed and distributed to the user node.
5) Status is updated if an error occurs. Status is distributed to the user node at regular intervals.



3. 2 How to install Modbus RTU Node

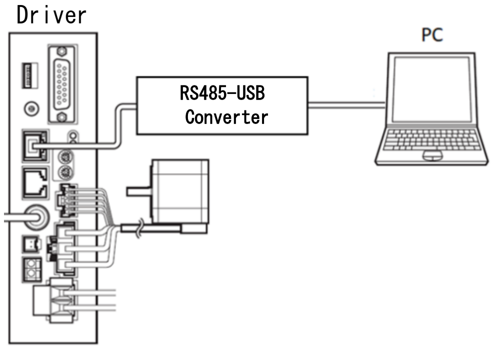
Create a user work folder in any location and perform build once. Then unzip the Modbus RTU Node package. After unzipping, copy and paste the unzipped folder to (/home/< ROS2_workspace> src) in the "src" folder of the user work folder. After that, perform build with the colon command, and when the build succeeds, installation is complete.

Build

```
$ cd ~/<ROS2_workspace>
$ source /opt/ros/humble/setup.bash
$ colcon build
```

3. 3 Connecting PC and driver

The connection method of a PC and driver is shown as below. Connect a commercially available LAN cable to the RS-485 communication connector of the driver, convert it to USB with an RS485-USB converter, and connect it to the PC. The RS485-USB converter has to be prepared by customers.



3. 4 How to start Node

Start the Modbus RTU Node and user node from the terminal. Communication starts when the user node performs distribution. Modbus RTU Node is launched by the launch command, which is a function of ROS. If the argument is omitted, the initial value will be set.

Parameter	Type	Description
com	String type	Device file name corresponding to serial port Initial value (”/dev/ttyUSB0”)
topicID	Numerical value type	IDs for identifying nodes and messages when multiple Modbus RTU Nodes are activated. Range: 0-15 Default: 0
baudrate	Numerical value type	Baud rate [bps] Range: 9600, 19200, 38400, 57600, 115200, 230400 Initial value: 9600
updateRate	Numerical value type	Frequency to distribute to user nodes [Hz] (status only) Range: 0-1000 Initial value: 100 *When the value is 0, regular distribution of status is not executed. *The maximum frequency that can be distributed varies depending on the environment.
firstGen	String type	Specify the 1st generation slave ID (1 to 31) “1.2, 3 to 31” Initial value (empty string)
secondGen	String type	Specify the 2nd generation slave ID (1 to 31) “1.2, 3 to 31 ” Initial value (empty string)
globalID	Numerical value type	Share Control Global ID used in ID Share mode. ID Share mode is not used when -1 is specified. Range: -1, 1-127 Initial value: -1
axisNum	Numerical value type	This is the number of driver axes that communicate in ID Share mode. Range: 1-31 Initial value: 1

※ Applicable series for ID Share mode: BLV R type, AZ Series miniDriver

(Example 1) When using two pieces of the BLV (slave ID=1, 2) and AZ (slave ID=3, 4) each.

```
$ source /opt/ros/humble/setup.bash
$ source ~/<ROS2_workspace>/install/local_setup.bash
$ ros2 launch om_modbus_master om_modbus_master_launch.py com:="/dev/ttyUSB0" topicID:=0
baudrate:=115200 updateRate:=100 firstGen:="1,2," secondGen:="3,4,"
```

(Example 2) When using two pieces of the AZ (slave ID=1, 2) set with Share Control Global ID=10 and baudrate=230400[bps] in ID Share mode

```
$ source /opt/ros/humble/setup.bash
$ source ~/<ROS2_workspace>/install/local_setup.bash
$ ros2 launch om_modbus_master om_modbus_master_launch.py com:="/dev/ttyUSB0" topicID:=0
baudrate:=230400 updateRate:=100 secondGen:="1,2," globalID:=10 axisNum:=2
```

*1st generation: AR, RKII, BLE, BLV; 2nd generation: AZ, CVD, BLH, BLV R type.

*Up to 8 slave IDs can be specified for the 1st generation and 2nd generation altogether.

Since the name for com differs depending on the environment, enter ls /dev/tty in the terminal to check the serial device name, and enter the displayed device name.

*To use the USB port, it is necessary to give the read/write authority of the USB port with the following command.

```
$ sudo chmod 666 /dev/ttyUSB0
```

*When the same serial port is specified to start multiple nodes, the nodes will be forcibly terminated when a message from the user node is distributed.

4 Message

This section describes the messages used with Modbus RTU Node. The initial value of each message is 0.

4.1 Distribution of query data

The following messages are distributed from the user node. The Modbus RTU Node receives the distributed message, converts it to a query and sends it to the driver. When distributing a message, the topic name and message name have to be declared. The topic name is “om_query□”, where the topicID specified when Modbus RTU Node is started is entered in the box (□). The message name is ″om_query” and is fixed.

Message name	Type	Description
slave_id	int8	Slave ID (0-31)
func_code	int8	Function code (0: read, 1: write, 2: read&write)
write_addr	int32	Upper register address that is the starting point for write
read_addr	int32	Upper register address that is the starting point for read
write_num	int8	Number of write data (1 to 32) *1 to 64 in ID Share mode*1
read_num	int8	Number of read data (1 to 32) *1 to 64 in ID Share mode*1
data[64]	int32	Write data to register (specified only for write and read&write)

*1 Depending on the amount of data, query/response length may exceed 256 bytes, and Error occurs.

4.2 Response subscription

For read, and read&write, the response is parsed and distributed to the user node. For write, the response of the driver is distributed as it is. When subscribing to response data, the topic name and message name have to be declared. The topic name is “om_response□”, where the topicID specified when Modbus RTU Node is started is entered in the box (□). The message name is ″om_response” and is fixed.

Message name	Type	Description
slave_id	int8	Slave ID of subscribed response
func_code	int8	Function code of subscribed response
data[64]	int32	Result of parsing the response from the driver

4.3 Status subscription

Status information is distributed in a periodical cycle by the Modbus RTU Node. The frequency is specified by the updateRate argument of the launch command. When subscribing to Status, the topic name and message name have to be declared. The topic name is “om_state□”, where the topicID specified when Modbus RTU Node is started is entered in the box (□). The message name is ″om_response” and is fixed.

Message name	Type	Description
state_driver	int8	0: Communication possible 1: Communication in progress
state_mes	int8	0: No message 1: Message arrived 2: Message error
state_error	int8	0: No error 1: No response 2: Exceptional response

4.4 Node communication interval

When distributing continuously from the user node, provide spacing of the time until the status state_driver changes from 1 to 0. When the status is not used, increase the interval by 50 [ms] or more (100 [ms] or more when using the read&write of the 1st generation). When a message is distributed while state_driver is 1 (communication is in progress), it will be discarded and no errors will occur.

5 Package configuration

The configuration of the package is as follows.

```
om_modbus_master
 │
 ├── include
 │      └── om_modbus_master
 │             ├── ICheckData.h
 │             ├── ICheckIdShareMode.h
 │             ├── IConvertQueryAndResponse.h
 │             ├── ISetMessage.h
 │             ├── ISetResponse.h
 │             ├── om_base.h
 │             ├── om_broadcast.h
 │             ├── om_first_gen.h
 │             ├── om_idshare_mode.h
 │             ├── om_node.h
 │             ├── om_ros_message.h
 │             └── om_second_gen.h
 │
 ├── launch
 │      └── om_modbus_master_launch.py
 │
 ├── src
 │      ├── om_base.cpp
 │      ├── om_broadcast.cpp
 │      ├── om_first_gen.cpp
 │      ├── om_idshare_mode.cpp
 │      ├── om_node.cpp
 │      ├── om_ros_message.cpp
 │      └── om_second_gen.cpp
 │
 ├── sample
 │      │
 │      ├── AZ
 │      │     ├── idshare1.py
 │      │     ├── idshare2.py
 │      │     ├── sample1_1.py
 │      │     ├── sample1_2.py
 │      │     ├── sample2_1.py
 │      │     ├── sample2_2.py
 │      │     └── sample3.py
 │      │
 │      ├── BLV
 │      │     ├── sample1_1.py
 │      │     ├── sample1_1.py
 │      │     ├── sample1_2.py
 │      │     ├── sample2.py
 │      │     └── sample3.py
 │      │
 │      └── BLV_R
 │      │      ├── idshare1.py
 │      │      └── idshare2.py
 │      │
 │      └── utils
 │             ├── clientasync.py
 │             └── const.py
 │
```

```
├── CMakeLists.txt
└── package.xml
```

```
om_msgs
 |
├── msg
|       ├── Query.msg
|       ├── Response.msg
|       └── State.msg
|
├── CMakeLists.txt
└── package.xml
```

The content of each file is shown below.

Name	Description
om_modbus_master_launch.py	Specify nodes and parameters to start
Query.msg	Query data definition
Response.msg	Response data definition
State.msg	Status data definition
om_ros_message.cpp	ROS communication class source file
om_ros_message.hpp	ROS communication class header file
om_base.cpp	Serial communication class source file
om_base.hpp	Serial communication class header file
om_first_gen.cpp	Modbus RTU base class (1st generation) source file
om_firts_gen.hpp	Modbus RTU base class (1st generation) header file
om_second_gen.cpp	Modbus RTU derived class (2nd generation) source file
om_second_gen.hpp	Modbus RTU derived class (2nd generation) header file
om_broadcast.cpp	Modbus RTU derived class (broadcast) source file
om_broadcast.hpp	Modbus RTU derived class (broadcast) header file
om_node.cpp	Main function source file
om_idshare_mode.cpp	ID Share mode source file
om_idshare_mode.hpp	ID Share mode header file
om_node.hpp	Main function header file
ICheckData.hpp	Interface
IConvertQueryAndResponse.hpp	Interface
ICheckIdShareMode.hpp	Interface
ISetMessage.hpp	Interface
ISetResponse.hpp	Interface
CMakeLists.txt	Build settings
package.xml	Information about the package name, version, author, etc.
sample1_1.py	Write sample (AZ, BLV)
sample1_2.py	Read sample (AZ, BLV)
sample2.py	Motor operation sample (BLV) 3-wire operation. At the time of shipment, operation data No.0 and No.1 are set to VR1, and since the rotation speed is specified by an external setter, the information is written to operation data No.2.
sample2_1.py	Motor operation (stored data operation) sample (AZ)

sample2_2.py	Motor operation (direct data operation) sample (AZ)
sample3.py	Motor operation (2 axes) and detection position monitor sample (AZ) Motor operation (2 axes) and detection speed monitor sample (BLV)
BLV_R/idshare1.py	Sample of motor operation (Read, Write) by ID Share mode
BLV_R/idShare2.py	Sample of motor operation (Read&Write) by ID Share mode
AZ/idshare1.py	Sample of motor operation (Read, Write) by ID Share mode
AZ/idshare2.py	Sample of motor operation (Read&Write) by ID Share mode

6 Build settings

The settings of “CMakeLists.txt” are shown below.

```
## CMake version
cmake_minimum_required(VERSION 3.8)

## Package name
project(om_modbus_master)

## C++14 is used
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

# show all warning
if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

## Specify the dependent package
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(om_msgs REQUIRED)

## Target the executable file to build
add_executable(om_modbusRTU_node
  src/om_node.cpp
  src/om_ros_message.cpp
  src/om_base.cpp
  src/om_first_gen.cpp
  src/om_second_gen.cpp
  src/om_broadcast.cpp
  src/om_idshare_mode.cpp
)

ament_target_dependencies(om_modbusRTU_node rclcpp std_msgs om_msgs)

## include directory
include_directories(include)

# Install Launch files
install(
  DIRECTORY launch
  DESTINATION share/${PROJECT_NAME}
)

install(
  TARGETS om_modbusRTU_node
  DESTINATION lib/${PROJECT_NAME}
)



install(
  DIRECTORY include/om_modbus_master
  DESTINATION include
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

# Register the package to resource index of “ament”
ament_package()
```

7 Sample code

User node sample code (Python) is included in the ″sample″ folder. The BLV Python sample code can be executed by inputting the following code into the terminal. (Start Modbus RTU Node before executing the sample code) Refer to the comments in the sample source code for sample content.

```
$ cd ~/<ROS2_workspace>/src/om_modbus_master/sample/BLV
$ python3 sample1_1.py
```

*User working folder: /home/<ROS2_workspace>/

Modbus RTU Node is launched by the launch command shown below. (When the BLV sample is used)

```
$ ros2 launch om_modbus_master om_modbus_master_launch.py com:="/dev/ttyUSB0" topicID:=0 baudrate:=115200 updateRate:=100 firstGen:="1,2,"
```

Also, when using the AZ in ID Share mode, the Python sample code can be executed by entering the following code in the terminal.
(Start Modbus RTU Node before executing the sample code)

```
$ cd ~/<ROS2_workspace>/src/om_modbus_master/sample/AZ
$ python3 idshare1.py
```

*User working folder: /home/<ROS2_workspace>/

Modbus RTU Node is launched by the launch command shown below. (When the AZ ID Share mode sample is used)

```
$ ros2 launch om_modbus_master om_modbus_master_launch.py com:="/dev/ttyUSB0" topicID:=0 baudrate:=230400 updateRate:=100 secondGen:= "1,2," globalID:=10 axisNum:= 2
```

Oriental Motor and its licensors shall not be liable for any direct or indirect loss, damage, etc. (including, but not limited to damage to hardware or other software, loss of business profits, business interruption, loss of business information etc.)

ORIENTAL MOTOR Co., Ltd.
March 28, 2023