Orientalmotor

ROS Package

Modbus RTU Node

Technical Material

4th 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 simply by distributing ROS messages.

■Applicable Product

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 20.04.2 LTS (Linux)
ROS distribution: noetic
ROS version: 1.15.11

■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 *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 Caution

(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 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 manual without permission of Oriental Motor Co., Ltd. is prohibited.

(5) The contents and information in this manual are as of April 2022. The contents of this document are subject to change without notice for improvement.

(6) This manual 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.

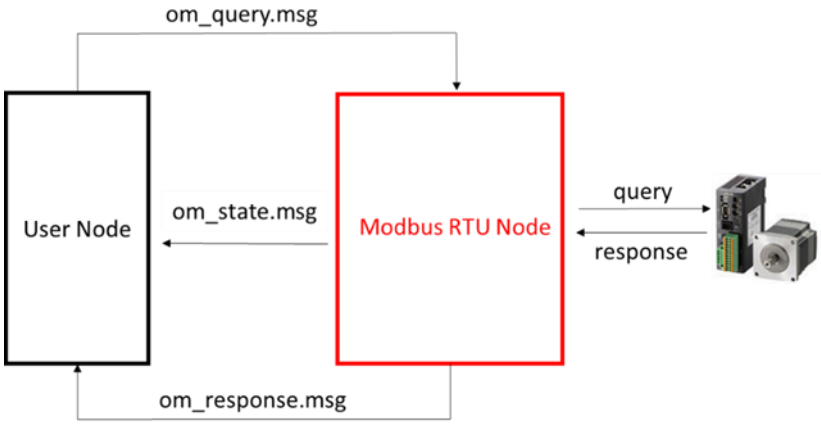
(7) This document is intended for those with ROS and Linux expertise. Please note that inquiries related to installation and usage of ROS, and Linux are not supported.

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/catkin_ws/src) in the "src" folder of the user work folder. After that, perform build with the catkin_make command, and if the build succeeds, installation is complete.

① Creating a user working folder

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws
$ catkin_make
```

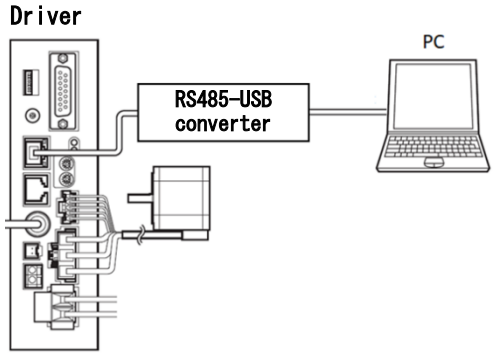
② Build

```
$ cd ~/catkin_ws
$ catkin_make
```

*Though the user work folder can be created in any location, here it is created in the following folder.
User working folder: /home/catkin_ws/
*There are times when a file that should be automatically generated during the build is not generated and the build fails. By performing build repeatedly, the file is generated and the build will succeed.

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

First, enter the "roscore" command from the terminal to launch the master. Then start the Modbus RTU Node and user node from another 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 default value will be set.

Argument name	Type	Description
com	String type	Device file name corresponding to serial port Default value ("/dev/ttyUSB0")
topicID	Numerical value type	ID (0 to 15) for identifying nodes and messages when multiple Modbus RTU Nodes are activated. Default value (0)
baudrate	Numerical value type	Baud rate [bps] 9600, 19200, 38400, 57600, 115200, 230400 Default value (9600)
updateRate	Numerical value type	Cycle to distribute to user nodes [Hz] (status only) 0 to 1000 Default value (1) *When the value is 0, regular distribution of status is not executed. *The maximum cycle that can be delivered changes depending on the environment.
firstGen	String types	Specify the 1st generation slave ID (1 to 31) "1,2,3...31" Default value (empty string)
secondGen	String types	Specify the 2nd generation slave ID (1 to 31) "1,2,3...31" Default value (empty string)
globalID	String types	Share Control Global ID used in ID Share mode. ID Share mode is not used when -1 is specified. If omitted by the launch command, globalID will be treated as -1. Range: -1, 1 to 127 Initial value: -1
asixNum	String types	This is the number of driver axes that communicate in ID Share mode. If omitted in the launch command, axisNum is treated as 1. Range: 1-5 Default 1

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

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

```
$ roslaunch om_modbus_master om_modbusRTU.launch com: ="/dev/ttyUSB0" topicID: =1 baudrate: =115200 updateRate: =1000 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

```
$ roslaunch om_modbus_master om_modbusRTU.launch com: ="/dev/ttyUSB0" topicID: =1 baudrate: =230400 updateRate: =1000 firstGen: "" 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.
*The node name for Modbus RTU Node is om_modbusRTU□. The numerical value of the topicID is entered in the box (□).
*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 delivered.

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 delivered from the user node. The Modbus RTU Node receives the delivered message, converts it to a query and sends it to the driver. When delivering 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 60 in ID Share mode (depending on the number of set axes)
read_num	int8	Number of read data (1 to 32) *1 to 60 in ID Share mode (depending on the number of set axes)
data[64]	int32	Write data to register (specified only for write and read&write)

4.2 Response subscription

For read, and read&write, the response is parsed and delivered to the user node. For write, the response of the driver is delivered 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 the 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 delivered 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 has 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 delivering 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 delivered 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_modbusRTU.launch
 |
 ├─ msg
 |     ├─ om_query.msg
 |     ├─ om_response.msg
 |     └─ om_state.msg
 |
 ├─ src
 |     ├─ om_base.h
 |     ├─ om_broadcast.cpp
 |     ├─ om_first_gen.h
 |     ├─ om_idshare_mode.cpp
 |     ├─ om_node.h
 |     ├─ om_ros_message.h
 |     └─ om_second_gen.h
 |
 ├─ sample
 |     ├─ cpp
 |     |     ├─ AZ
 |     |     |    ├─ sample1_1.cpp
 |     |     |    ├─ sample1_2.cpp
 |     |     |    ├─ sample2_1.cpp
 |     |     |    ├─ sample2_2.cpp
 |     |     |    └─ sample3.cpp
 |     |     |
 |     |     └─ BLV
 |     |          ├─ sample1_1.cpp
 |     |          ├─ sample1_2.cpp
 |     |          ├─ sample2.cpp
 |     |          └─ sample3.cpp
 |     |
 |     └─ python
 |          ├─ AZ
 |          |    ├─ idshare_sample1_1.py
 |          |    ├─ idshare_sample1_2.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
│               ├── idshare_sample1_1.py
│               └── idshare_sample1_2.py
│
├── CMakeLists.txt
└── package.xml
```

The contents of each file are shown below

Name	Description
om_modbusRTU.launch	Specify nodes and parameters to start with roslaunch
om_query.msg	Query data definition
om_response.msg	Response data definition
om_state.msg	Status data definition
om_ros_message.cpp	ROS communication class source file
om_ros_message.h	ROS communication class header file
om_base.cpp	Serial communication class source file
om_base.h	Serial communication class header file
om_first_gen.cpp	Modbus RTU base class (1st generation) source file
om_firts_gen.h	Modbus RTU base class (1st generation) header file
om_second_gen.cpp	Modbus RTU derived class (2nd generation) source file
om_second_gen.h	Modbus RTU derived class (2nd generation) header file
om_broadcast.cpp	Modbus RTU derived class (broadcast) source file
om_broadcast.h	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.h	ID Share mode header file
om_node.h	Main function header file
ICheckData.h	Interface
IConvertQueryAndResponse.h	Interface
ICheckIdShareMode.h	Interface
ISetMessage.h	Interface
ISetResponse.h	Interface
CMakeLists.txt	Build settings
package.xml	Information about the package name, version, author, etc.
sample1_1(.cpp、.py)	Write sample (AZ, BLV)
sample1_2(.cpp、.py)	Read sample (AZ, BLV)
sample2(.cpp, .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(.cpp, .py)	Motor operation (stored data operation) sample (AZ)
sample2_2(.cpp, .py)	Motor operation (direct data operation) sample (AZ)
sample3(.cpp, .py)	Motor operation (2 axes) and detection position monitor sample (AZ) Motor operation (2 axes) and detection speed monitor sample (BLV)
BLV-R/idshare_sample1_1.py	Sample of motor operation (Read, Write) by ID Share mode
BLV-R/idShare_sample1_2.py	Sample of motor operation (Read&Write) by ID Share mode
AZ/idshare_sample1_1.py	Sample of motor operation (Read, Write) by ID Share mode
AZ/idshare_sample1_2.py	Sample of motor operation (Read&Write) by ID Share mode

6 Build settings

ROS uses its own build system catkin. This is an extended CMake, and CMake describes the information necessary for building in "CMakeLists.txt". The settings of "CMakeLists.txt" are shown below.

```
## CMake version
cmake_minimum_required(VERSION 2.8.3)

## Package name
om_modbus_master

## C++11 is used
add_compile_options(-std=c++11)

## Specify the dependent package
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)

## Generate messages in the 'msg' folder
add_message_files(
   FILES
   om_query.msg
   om_response.msg
   om_state.msg
)

## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  std_msgs
)

## Transmit the information specific to catkin which is necessary to generate CMake file to the build system.
catkin_package(
 INCLUDE_DIRS include
CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
)
```

```
## include directory
include_directories(include ${catkin_INCLUDE_DIRS})

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

add_dependencies(om_modbusRTU_node ${${PROJECT_NAME}_EXPORTED_TARGETS}${catkin_EXPORTED_TARGETS})

## Specify which library the executable target should link against
target_link_libraries(om_modbusRTU ${catkin_LIBRARIES})
```

7 Sample code

User node samples (C++, Python) are 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 ~/catkin_ws/src/om_modbus_master/sample/python/BLV
$ python sample1_1.py
```

*User working folder: /home/catkin_ws/

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

```
$ roslaunch om_modbus_master om_modbusRTU.launch com:="/dev/ttyUSB0" topicID:=1 baudrate:=115200
updateRate:=1000 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 ~/catkin_ws/src/om_modbus_master/sample/python/AZ
$ python idshare_sample1_1.py
```

*User working folder: /home/catkin_ws/

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

```
$ roslaunch om_modbus_master om_modbusRTU.launch com:="/dev/ttyUSB0" topicID:=1 baudrate:=230400
updateRate:=1000 firstGen:="" secondGen:= "1,2," globalID:="10" axisNum:= "2"
```


ORIENTAL MOTOR Co., Ltd.
April 15,2022