

Mitsubishi Electric Industrial Robot

MELFA API for Linux Instruction Manual

SAFETY PRECAUTIONS

Read the separate Safety Manual thoroughly before using this product. Take precautions as necessary.

Before using this product, please read this manual and the relevant manuals carefully and pay full attention to safety to handle the product correctly.

The precautions given in this manual are concerned with this product only. For the safety precautions of the robot system, refer to the user's manual for the peripheral equipment used.

In this manual, the safety precautions are classified into three levels: "⚠ DANGER", "⚠ WARNING", and "⚠ CAUTION".

 DANGER	Precaution indicating cases where there is a risk of operator fatality or serious injury if handling is mistaken. Always observe these precautions to safely use the robot.
 WARNING	Precaution indicating cases where the operator could be subject to fatalities or serious injuries if handling is mistaken. Always observe these precautions to safely use the robot.
 CAUTION	Precaution indicating cases where operator could be subject to injury or physical damage could occur if handling is mistaken. Always observe these precautions to safely use the robot.

Under some circumstances, failure to observe the precautions given under "⚠ CAUTION" may lead to serious consequences.

Observe the precautions of both levels because they are important for personal and system safety.

Make sure that the end users read this manual and then keep the manual in a safe place for future reference.

A. These show precautions based on the Ordinance on Industrial Safety and Health (Articles 36, 104, 150, 151).

 DANGER	<ul style="list-style-type: none">● Provide a fence or enclosure during operation to prevent contact of the operator and robot. → Installation of safety fence
 WARNING	<ul style="list-style-type: none">● Prepare a device that allows operation to be stopped immediately during teaching work. (This also applies to maintenance work with the power source turned ON.) → Setting of emergency stop switch

CAUTION

- All teaching work must be carried out by an operator who has received special training.
(This also applies to maintenance work with the power source turned ON.)
→ Enforcement of safety training
 - For teaching work, prepare a work plan related to the methods and procedures of operating the robot, and to the measures to be taken when an error occurs or when restarting. Carry out work following this plan.
(This also applies to maintenance work with the power source turned ON.)
→ Preparation of work plan
 - During teaching work, place a sign indicating that teaching work is in progress on the start switch, etc.
(This also applies to maintenance work with the power source turned ON.)
→ Indication of teaching work in progress
 - Establish a set of signaling methods to the related operators for starting work, and follow these methods.
→ Signaling of operation start
 - As a principle turn the power OFF during maintenance work. Place a sign indicating that maintenance work is in progress on the start switch, etc.
→ Indication of maintenance work in progress
 - Before starting work, inspect the robot, emergency stop switch and other related devices, etc., and confirm that there are no errors.
→ Inspection before starting work
-

B. This shows precaution points given in the "Safety Manual".

Refer to the Safety Manual for further details.

DANGER

- When automatic operation of the robot is performed using multiple control devices (GOT, programmable controller, push-button switch), the interlocking of operation rights of the devices, etc. must be designed by the customer.
- Do not connect the Handy GOT when using the GOT direct connection function of this product. Failure to observe this may result in property damage or bodily injury because the Handy GOT can automatically operate the robot regardless of whether operation rights are enabled or not.
- Do not connect the Handy GOT to a programmable controller when using an iQ Platform compatible product with the CR800-R controller. Failure to observe this may result in property damage or bodily injury because the Handy GOT can automatically operate the robot regardless of whether the operation rights are enabled or not.
- Do not remove the SSCNET III cable while power is supplied to the multiple CPU system or the servo amplifier. Do not look directly at light emitted from the end of SSCNET III connectors or SSCNET III cables of the Motion CPU or the servo amplifier. Doing so may cause discomfort. (Reference: SSCNET III employs a Class 1 or equivalent light source as specified in JIS C 6802 and IEC 60825-1 (domestic standards in Japan).)
- Do not remove the SSCNET III cable while power is supplied to the controller. Do not look directly at light emitted from the end of SSCNET III connectors or SSCNET III cables. Doing so may cause discomfort.
(Reference: SSCNET III employs a Class 1 or equivalent light source as specified in JIS C 6802 and IEC 60825-1 (domestic standards in Japan).)
- Attach the cap to the SSCNET III connector after disconnecting the SSCNET III cable. If the cap is not attached, dirt or dust may adhere to the connector pins, resulting in deterioration connector properties, and leading to malfunction.

WARNING

- Securely install the hand and tool, and securely grasp the workpiece.
Failure to observe this could lead to personal injuries or damage if the object comes off or flies off during operation.
 - Securely ground the robot and controller.
Failure to observe this may result in malfunction due to noise, or even electric shock.
 - When carrying out teaching work in the robot's movement range, always secure the priority right for the robot control. Failure to observe this may result in bodily injury or property damage if the robot is able to start with external commands.
 - When the robot arm has to be moved by hand from an external area, do not place hands or fingers in the openings. Failure to observe this could lead to hands or fingers getting caught in the robot depending on its posture.
-

CAUTION

- Use the robot within the environment given in the specifications.
Failure to do so could lead to a decrease in reliability or faults.
(Temperature, humidity, atmosphere, noise environment, etc.)
 - Transport the robot in the designated transportation posture.
Failure to do so may result in bodily injury or breakdown if the robot is dropped.
 - Always use the robot installed on a secure table.
Use in an unstable posture could lead to positional deviation and vibration.
 - Wire the cable as far away from noise sources as possible.
If placed near a noise source, positional deviation or malfunction could occur.
 - Do not apply excessive force on the connector or excessively bend the cable.
Failure to observe this could lead to contact defects or wire breakage.
 - Make sure that the workpiece weight, including the hand, does not exceed the rated load or tolerable torque.
Exceeding these values could lead to errors or faults.
 - Indicate the operation state during robot operation.
Failure to indicate the state could lead to operators approaching the robot or to incorrect operation.
 - Keep the jog speed as low as possible, and always watch the robot. Failure to do so could lead to interference with the workpiece or peripheral devices.
 - After editing the program, always confirm the operation with step operation before starting automatic operation. Failure to do so could lead to interference with peripheral devices because of programming mistakes, etc.
 - Make sure that if the safety fence entrance door is opened during automatic operation, the door is locked or that the robot will automatically stop. Failure to do so could lead to personal injuries.
 - Never carry out modifications based on personal judgments, or use non-designated maintenance parts. Failure to observe this may result in breakdown or malfunction.
 - Do not stop the robot or apply emergency stop by turning OFF the controller's main power.
If the controller's main power is turned OFF during automatic operation, the robot accuracy could be adversely affected. Moreover, the robot arm may drop or drift into peripheral devices or other objects.
 - Do not turn OFF the main power to the controller while rewriting the internal information of the controller such as the program or parameters.
If the main power of the robot controller is turned OFF while rewriting programs or parameters during auto operation, internal robot controller information may be corrupted.
 - Make sure there are no mistakes in the wiring. Connecting differently to the way specified in the manual can result in errors, such as the emergency stop not being released.
In order to prevent errors occurring, be sure to check that all functions (such as the teaching pendant emergency stop, customer emergency stop, and door switch) are working properly after wiring is finished.
 - To maintain the security (confidentiality, integrity, and availability) of the robot and the system against unauthorized access, DoS^{*1} attacks, computer viruses, and other cyberattacks from unreliable networks and devices via network, take appropriate measures such as firewalls, virtual private networks (VPNs), and antivirus solutions.
Mitsubishi Electric Corporation will not be held responsible for any robot or system problems arising from DoS attacks, unauthorized access, computer viruses, and cyberattacks.
^{*1} DoS: A denial-of-service (DoS) attack disrupts services by overloading systems or exploiting vulnerabilities, resulting in a denial-of service (DoS) state.
-

CONTENTS

SAFETY PRECAUTIONS	1
INTRODUCTION	7
CHAPTER 1 OUTLINE	8
1.1 Contents of the download file	8
1.2 Directory structure	9
1.3 Versioning	9
1.4 Glossary	10
CHAPTER 2 SYSTEM REQUIREMENTS	11
2.1 Computer	11
2.2 Robot controller	12
CHAPTER 3 HOW TO USE	13
3.1 System configuration	13
3.2 Connecting hardware	14
3.3 Installation	15
3.4 C++ (CMake)	16
3.5 Python	17
3.6 ROS 2	18
CHAPTER 4 PROGRAMMING LIBRARY	22
4.1 Precautions for use	22
4.2 Error No.	22
4.3 Namespace list (C++ only)	22
4.4 Class list.	22
4.5 RobotController class	23
Method list	23
Field list	24
Constructor	24
Connection	25
Disconnection	26
Timeout setting	27
Timeout acquisition	28
Basic information acquisition	29
Servo ON/OFF	30
Override settings	31
Override acquisition	32
Slot state acquisition	33
Program start	34
Program stop	35
Program reset	36
Program slot initialization	37
Direct execution	38
Read variable contents (integer type)	39
Read variable contents (real number type)	40
Read variable contents (string type)	41
Read variable contents (position type)	42

Read variable contents (joint type)	43
Error reset	44
Current position acquisition (joint)	45
Current position acquisition (XYZ)	46
4.6 BasicInfo class	47
Method list	47
Field list	47
4.7 SlotState class	48
Method list	48
Field list	48
4.8 Joint class	49
Method list	49
Field list	49
4.9 Position class	50
Method list	50
Field list	50
 CHAPTER 5 SAMPLE CODE	 52
REVISIONS	55
TRADEMARKS	56
COPYRIGHT INFORMATION	56

INTRODUCTION

Before using the library, read the "END-USER SOFTWARE LICENSE AGREEMENT" thoroughly. Use of the MELFA library is permitted on condition that you agree to the terms of the agreement.

Make sure that you have fully understood the contents of this manual before using the Library.

1 OUTLINE

Thank you for purchasing Mitsubishi Electric MELFA industrial robots.

This document is the instruction manual for MELFA API for Linux.

MELFA API for Linux is a programming interface (communication middleware) that allows developers to flexibly control MELFA robots from a Linux environment.

By using this API, it is possible to utilize robot control functions in a Linux-based environment and promote smart production systems.

1.1 Contents of the download file

The download file is a compressed file. The contents of the decompressed file are shown in the table below.

Contents	Description
Programming library	A general-purpose programming language library for operating MELFA robots. The following languages (supported by ROS 2 Humble) are supported. <ul style="list-style-type: none">• C++ (C++17)• Python (Python 3.10)
Sample code	A simple code example using the programming library.

1.2 Directory structure

The directory structure is shown in the table below.

Directory/File name	Description
■ melfa_api_for_linux	Root directory of this API
■ lib	Directory for storing the library
■ libmelfa_api.so	Shared API library
■ api	Directory that stores APIs for each language
■ cpp	Directory for C++
■ melfa_api.hpp	Header file for C++
■ python	Python Directory
■ melfa_api.py	Python module
■ sample	Directory for storing samples
■ cpp	Directory for storing C++ samples
■ main.cpp	C++ sample code
■ python	Directory for storing Python samples
■ main.py	Sample Python code

1.3 Versioning

The compressed files use semantic versioning.

X.Y.Z

Example) 1.10.123

X: Major version. This is incremented when incompatible API changes are made.

Y: Minor version. This is incremented when compatible API changes are made.

Z: Patch version. This is incremented when compatible bug fixes are made.

1.4 Glossary

Term	Description
MELFA API for Linux	Name of the product.
Real-time external control command (MXT)	A function that allows real-time external control of Mitsubishi Electric robot arms. * Arm and I/O control only. Program execution and error reset is not possible.
MELFA ROS2 Driver	ROS 2 package for operating Mitsubishi Electric robot controllers. The operating environment is Ubuntu (Linux) OS. MXT is used in the communications. https://github.com/Mitsubishi-Electric-Asia/melfa_ros2_driver
Robot Operating System (ROS)	An open-source framework equipped with drivers, algorithms, developer tools, and more to support robot applications. It runs mainly on Linux OS (Ubuntu). (In recent years, it has also started to work on Windows.)
Ubuntu	One of the Linux distributions and free software. Ubuntu is one of the operating systems supported by ROS.

2 SYSTEM REQUIREMENTS

2.1 Computer

The system requirements for the computer needed to run this API are shown in the table below.

Item	Description
Operating system	Ubuntu 22.04 LTS
CPU / Graphics	64-bit architecture
Main memory	4 GB or more
Communication interface	Ethernet port

2.2 Robot controller

The target robot controller and communication method for this API are shown in the table below.

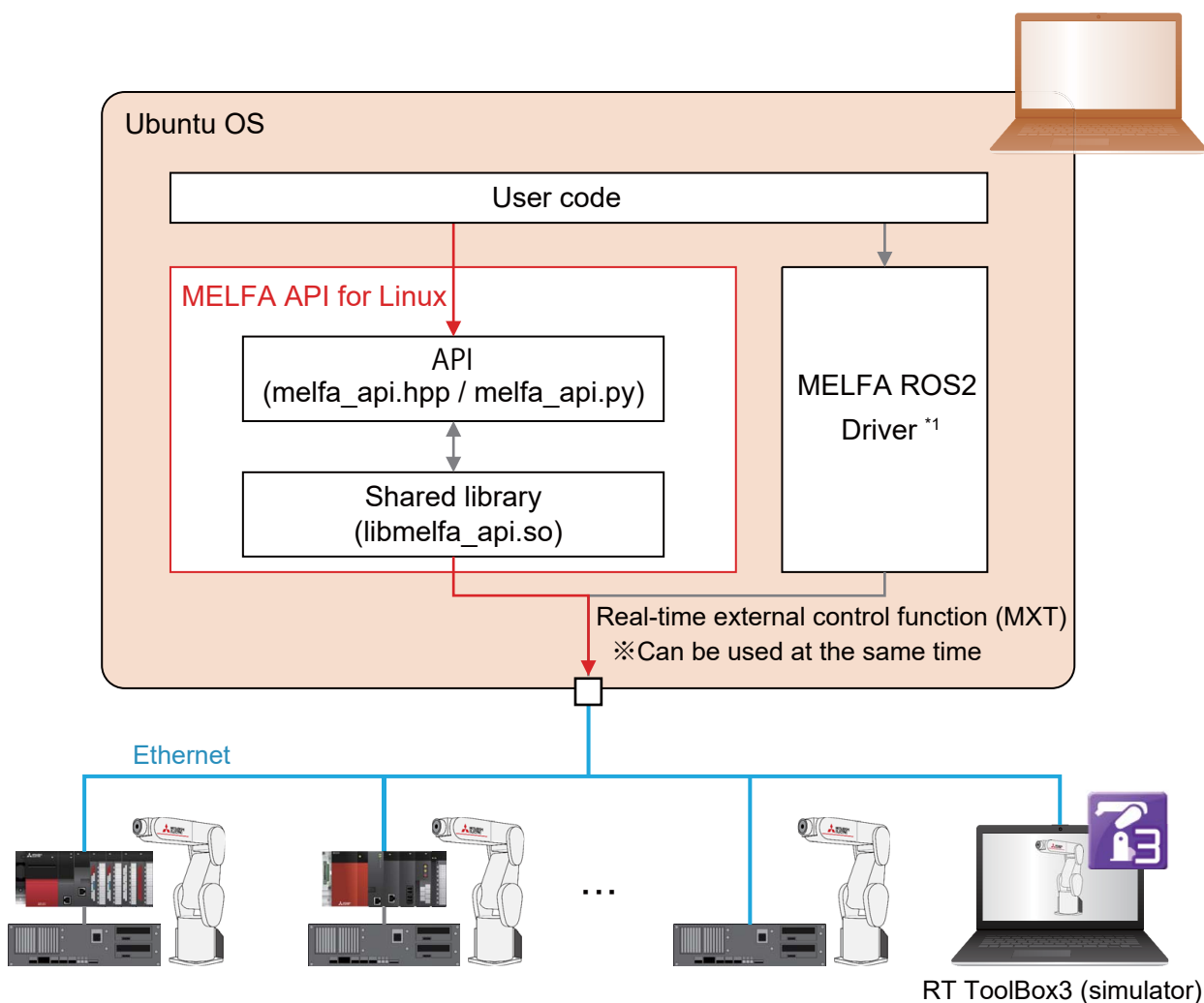
Series	Configuration	Connection	Supported robot controller version
CR8xx series	CR8xx-D	USB	NA
		Ethernet (TCP/IP)	A1 or later
		GOT communication	NA
	CR8xx-R	CRnQ_R communication	NA
		Ethernet (TCP/IP)	A1 or later
	CR8xx-Q	CRnQ_R communication	NA
		Ethernet (TCP/IP)	A1 or later
	Other than the above	—	NA
Other than CR8xx series	—	—	NA

3 HOW TO USE

The use case described in this chapter is an example. Change the system configuration to suit your environment.
The following procedure assumes that this API is stored in the home directory of a computer with Ubuntu 22.04 LTS installed.

3.1 System configuration

3

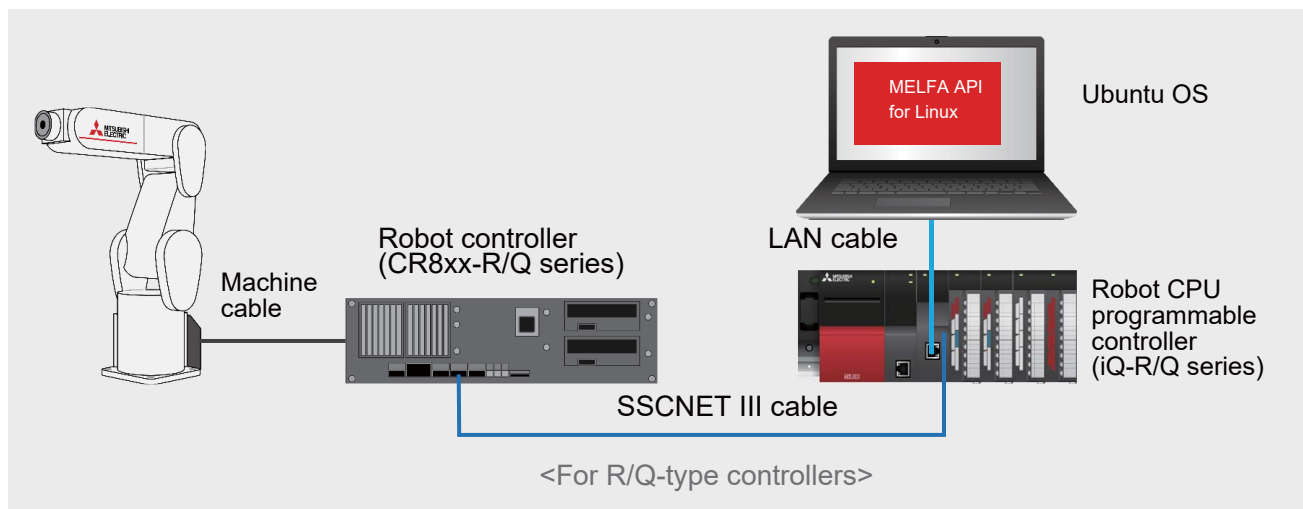
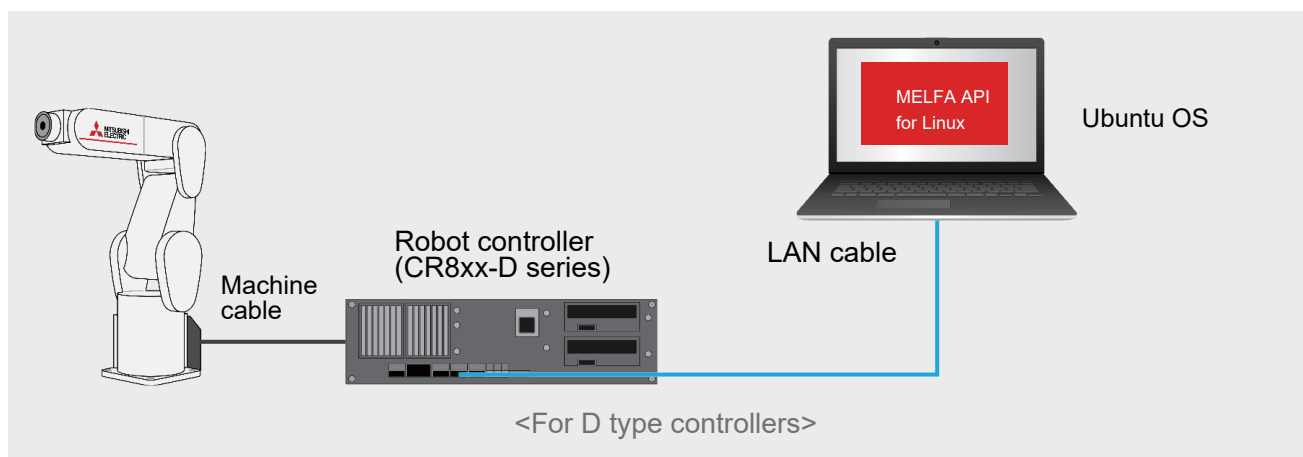
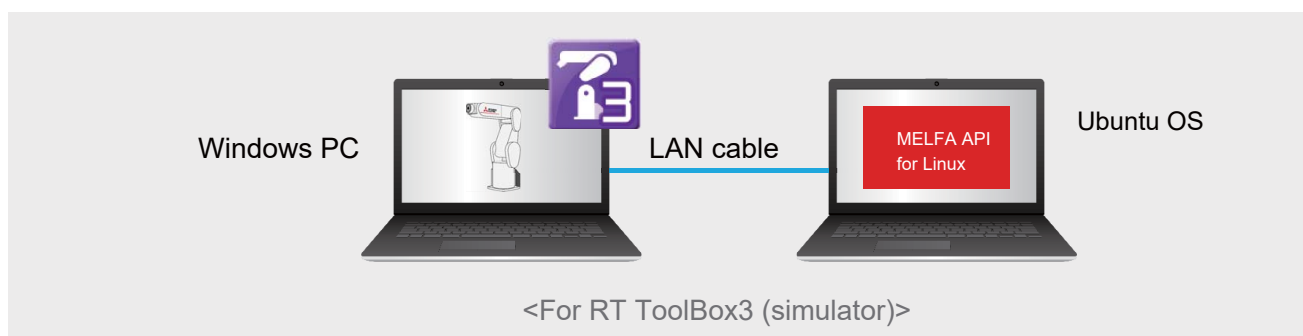


*1 MELFA ROS2 Driver can perform motion control and signal control for Mitsubishi Electric industrial and collaborative robots.
MELFA ROS2 Driver is not included in this API and is not required.

3.2 Connecting hardware

Examples of hardware connections are shown in the figures below.

For information on the connection methods, refer to the instruction manuals included with the products.



3.3 Installation

Decompress the API before using it.

```
$ tar -zxvf melfa_api_for_linux_X_Y_Z.tar.gz
```

* Substitute X_Y_Z with the version name of the MELFA API for Linux you are using.

Place this API library in a directory that can be referenced.

For now, copy it to /usr/local/lib/.

```
$ sudo cp ~/melfa_api_for_linux/lib/libmelfa_api.so /usr/local/lib/  
$ sudo ldconfig
```

To uninstall the API, delete the file above.

3.4 C++ (CMake)

This section explains how to use this API from C++ (CMake).

MELFA API for Linux must be installed.

1. Create a work directory.

```
$ mkdir my_project_cpp
$ cd my_project_cpp
```

2. Copy the library header from this API.

```
$ cp ~/melfa_api_for_linux/api/cpp/melfa_api.hpp ./
```

3. Create a program.

```
$ touch main.cpp
```

In this example, the program will display the robot model name after the robot controller is connected.

Refer to the table (◇Parameters for checking the IP address and port number) and configure the IP address and port number.

* Do not connect multiple devices to the same port.

◇Parameters for checking the IP address and port number

Setting	Robot	RT ToolBox3 (simulation)
IP address	NETIP	Computer IP address
Port	One of the NETPORT elements from the second through to the tenth elements	

* Error handling is omitted in this sample for simplicity; however, be sure to implement appropriate error handling in actual operation.

```
#include <iostream>
#include "melfa_api.hpp"
using namespace MELFA;

int main()
{
    RobotController rc("192.168.0.20", 10002);

    if (rc.connect() != MR_OK)
    {
        std::cerr << "Failed to connect." << std::endl;
        return -1;
    }

    BasicInfo basic_info;
    rc.get_basic_info(basic_info);
    std::cout << basic_info.rb_name << std::endl;
    rc.disconnect();

    return 0;
}
```

4. Create CMakeLists.txt.

```
$ touch CMakeLists.txt
```

Enter the following code.

```
cmake_minimum_required(VERSION 3.10)
project(my_project_cpp)

add_executable(my_project_cpp main.cpp)
target_link_libraries(my_project_cpp melfa_api)
```

5. Build the program.

```
$ mkdir build && cd build
$ cmake .. && make
```

6. Execute the program to display the robot model name.

```
$ ./my_project_cpp
RV-7FRL-D
```

3.5 Python

This section explains how to use this API from Python.
MELFA API for Linux must be installed.

1. Create a work directory.

```
$ mkdir my_project_py  
$ cd my_project_py
```

2. Copy the module from this API.

```
$ cp ~/melfa_api_for_linux/api/python/melfa_api.py ./
```

3. Create a program.

```
$ touch main.py
```

In this example, the program will display the robot model name after the robot controller is connected.

Refer to the table (◇Parameters for checking the IP address and port number) for the IP address and port number.

* Error handling is omitted in this sample for simplicity; however, be sure to implement appropriate error handling in actual operation.

```
from melfa_api import RobotController, RcResultCode  
  
def main() -> None:  
    rc: RobotController = RobotController("192.168.0.20", 10002)  
  
    ret = rc.connect()  
    if ret != RcResultCode.MR_OK:  
        print("Failed to connect.")  
        return  
  
    _ret, basic_info = rc.get_basic_info()  
    print(f"{basic_info.rb_name}")  
  
    rc.disconnect()  
  
if __name__ == "__main__":  
    main()
```

4. Execute the program to display the robot model name.

```
$ python3 main.py  
RV-7FRL-D
```

3.6 ROS 2

This section explains how to use this API from ROS 2.

The ROS 2 development environment and this API are assumed to have been installed.

In this example, we will create a package to execute and stop robot programs using Services, which is a form of ROS 2 communication.

1. Set the ROS 2 environment variables.

```
$ source /opt/ros/humble/setup.bash
```

2. Create a workspace.

```
$ mkdir -p ~/ros2_ws_melfa/src
```

3. Create a package.

Name the package "my_melfa_package" and the node "my_melfa_service". Use CMake.

```
$ cd ~/ros2_ws_melfa/src
$ ros2 pkg create --build-type ament_cmake --node-name my_melfa_service my_melfa_package
```

The package will have the following structure:

```
├── my_melfa_package
│   ├── CMakeLists.txt
│   ├── include
│   │   └── my_melfa_package
│   ├── package.xml
│   └── src
│       └── my_melfa_service.cpp
```

4. Add a dependency on std_srvs.

Add the following code to "package.xml".

```
<depend>std_srvs</depend>
```

Add the following code to "CMakeLists.txt".

```
find_package(std_srvs REQUIRED)
find_package(rclcpp REQUIRED)
ament_target_dependencies(my_melfa_service rclcpp std_srvs)
```

5. Add a dependency on this API.

Copy the library header.

```
$ cp ~/melfa_api_for_linux/api/cpp/melfa_api.hpp ./my_melfa_package/include/my_melfa_package
```

Add the following code to "CMakeLists.txt".

```
target_include_directories(my_melfa_service PRIVATE include)
target_link_libraries(my_melfa_service melfa_api)
```

6. Create a source file.

Enter the following code.

Refer to the table (◇Parameters for checking the IP address and port number) for the IP address and port number.

* Error handling is omitted in this sample for simplicity; however, be sure to implement appropriate error handling in actual operation.

```
#include "rclcpp/rclcpp.hpp"
#include "std_srvs/srv/empty.hpp"
#include "my_melfa_package/melfa_api.hpp"

class MelfaService : public rclcpp::Node
{
public:
    MelfaService() : Node("melfa_service")
    {
        // Create a ROS service named "melfa_start"
        service_start_ = this->create_service<std_srvs::srv::Empty>(
            "melfa_start",
            std::bind(
                &MelfaService::handle_service_start,
                this,
                std::placeholders::_1,
                std::placeholders::_2));

        // Create a ROS service named "melfa_stop"
        service_stop_ = this->create_service<std_srvs::srv::Empty>(
            "melfa_stop",
            std::bind(
                &MelfaService::handle_service_stop,
                this,
                std::placeholders::_1,
                std::placeholders::_2));

        // Connect to the MELFA robot controller
        melfa_rc_.connect();
    }

    ~MelfaService()
    {
        // Disconnect from the robot controller
        melfa_rc_.disconnect();
    }

private:
    rclcpp::Service<std_srvs::srv::Empty>::SharedPtr service_start_;
    rclcpp::Service<std_srvs::srv::Empty>::SharedPtr service_stop_;

    // IP address and Port number of robot controller
    MELFA::RobotController melfa_rc_ = {"192.168.0.20", 10002};

    // Callback function to handle the "melfa_start" service request
    void handle_service_start(
        [[maybe_unused]]const std::shared_ptr<std_srvs::srv::Empty::Request> request,
        [[maybe_unused]]std::shared_ptr<std_srvs::srv::Empty::Response> response)
    {
        RCLCPP_INFO(this->get_logger(), "Received start");
        melfa_rc_.start();
    }

    // Callback function to handle the "melfa_stop" service request
    void handle_service_stop(
        [[maybe_unused]]const std::shared_ptr<std_srvs::srv::Empty::Request> request,
        [[maybe_unused]]std::shared_ptr<std_srvs::srv::Empty::Response> response)
    {
        RCLCPP_INFO(this->get_logger(), "Received stop");
        melfa_rc_.stop();
    }
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<MelfaService>());
    rclcpp::shutdown();
    return 0;
}
```

7. Build the program.

```
$ cd ~/ros2_ws_melfa
$ colcon build
```

8. Execute the program.

```
$ source install/setup.bash
$ ros2 run my_melfa_package my_melfa_service
```

9. Start rqt in a different terminal.

```
$ source /opt/ros/humble/setup.bash
$ cd ~/ros2_ws_melfa
$ source install/setup.bash
$ rqt
```

10. Open Plugins/Services/Service Caller.



You can run the robot program by selecting "/melfa_start" from the Service list and pressing the call button.

You can stop the robot program by selecting "/melfa_stop" from the Service list and pressing the call button.

4 PROGRAMMING LIBRARY

4.1 Precautions for use

When executing API commands while the robot program is running, it is recommended to wait 100 ms or more before executing the next command.

Do not use RT ToolBox3 online mode when using this API.

4.2 Error No.

The meanings of the error numbers are shown in the table below. Values less than or equal to 0 are defined as constants.

Error No. RcResult	Description
1 or more	The request failed due to an error with the robot controller. The error number issued by the robot controller is stored. For further information on error numbers, refer to the "CR800 Series Controller INSTRUCTION MANUAL Troubleshooting (BFP-A3480)".
MR_OK (0)	The request to the robot controller was successful.
MR_ER_SEND (-1)	The request to the robot controller failed due to an error during transmission. Check the connection with the robot controller. This error will occur if it is not connected.
MR_ER_ARGS (-2)	The request to the robot controller failed due to an invalid argument in the request. Check the argument when sending the request.
MR_ER_RECV (-3)	The request to the robot controller failed due to an error during reception. Check the connection with the robot controller.
MR_ER_DATA (-4)	The request to the robot controller failed due to an error with the received data.
MR_ER_TOUT (-5)	The communication timed out. Prolong the timeout time or check the connection with the robot controller.

4.3 Namespace list (C++ only)

Namespace	Description
MELFA	All C++ APIs are included in MELFA namespace.

4.4 Class list

Class name	Description	Reference
RobotController	Can perform various robot controller operations.	 Page 23 RobotController class
BasicInfo	Displays basic information.	 Page 47 BasicInfo class
SlotState	Displays the slot state.	 Page 48 SlotState class
Joint	Displays joint type.	 Page 49 Joint class
Position	Displays position type.	 Page 50 Position class

4.5 RobotController class

Various robot controller operations can be performed.

Method list

No.	Method name	Function	Compatible API Ver.	Compatible R/C Ver.	Reference
1.	RobotController	Constructor	1.0 or later	A1 or later	 Page 24 Constructor
2.	connect	Connection	1.0 or later	A1 or later	 Page 25 Connection
3.	disconnect	Disconnection	1.0 or later	A1 or later	 Page 26 Disconnection
4.	set_timeout	Receive timeout setting	1.0 or later	A1 or later	 Page 27 Timeout setting
5.	get_timeout	Receive timeout acquisition	1.0 or later	A1 or later	 Page 28 Timeout acquisition
6.	get_basic_info	Basic information acquisition	1.0 or later	A1 or later	 Page 29 Basic information acquisition
7.	turn_servo	Servo ON/OFF	1.0 or later	A1 or later	 Page 30 Servo ON/OFF
8.	set_override	Override setting	1.0 or later	A1 or later	 Page 31 Override settings
9.	get_override	Override acquisition	1.0 or later	A1 or later	 Page 32 Override acquisition
10.	get_slot_state	Slot state acquisition	1.0 or later	A1 or later	 Page 33 Slot state acquisition
11.	start	Program start	1.0 or later	A1 or later	 Page 34 Program start
12.	stop	Program stop	1.0 or later	A1 or later	 Page 35 Program stop
13.	reset	Program reset	1.0 or later	A1 or later	 Page 36 Program reset
14.	init_slot	Program slot initialization	1.0 or later	A1 or later	 Page 37 Program slot initialization
15.	direct	Direct execution	1.0 or later	A1 or later	 Page 38 Direct execution
16.	get_val_integer	Program variable contents acquisition (integer type)	1.0 or later	A1 or later	 Page 39 Read variable contents (integer type)
17.	get_val_float	Program variable contents acquisition (real number type)	1.0 or later	A1 or later	 Page 40 Read variable contents (real number type)
18.	get_val_chars	Acquire program variable contents (string type)	1.0 or later	A1 or later	 Page 41 Read variable contents (string type)
19.	get_val_pos	Acquire program variable contents (position type)	1.0 or later	A1 or later	 Page 42 Read variable contents (position type)
20.	get_val_joint	Acquire program variable contents (joint type)	1.0 or later	A1 or later	 Page 43 Read variable contents (joint type)
21.	reset_error	Error reset	1.0 or later	A1 or later	 Page 44 Error reset
22.	get_curpos_joint	Acquire current position (joint)	1.0 or later	A1 or later	 Page 45 Current position acquisition (joint)
23.	get_curpos_xyz	Acquire current position (XYZ)	1.0 or later	A1 or later	 Page 46 Current position acquisition (XYZ)

Field list

No.	Field name	Type	Description	Compatible version
1.	—	—	—	—

Constructor

Outline

Connects to the robot controller.

C++

Signature	RobotController(const std::string& ip_addr, int port)		
Parameter	[in]	ip_addr	IP address of connected controller
	[in]	port	Port number of connected controller
Return value	—		
Example code	RobotController rc("192.168.0.20", 10002);		

Python

Signature	def __init__(self, ip_addr: str, port: int)		
Parameter	Same as C++		
Return value	Same as C++		
Example code	rc: RobotController = RobotController("192.168.0.20", 10002)		

Notes

At instantiation, you only specify the destination controller. The connection is not established.
Use the CONNECT method to establish connection.

Connection

Outline			
Connects to the robot controller.			
C++			
Signature	auto connect(const std::string& ip_addr = "", int port = 0) -> RcResult		
Parameter	[in]	ip_addr	IP address of the connected controller
	[in]	port	Port number of the connected controller
Return value	Error No.		
Example code	<pre>// Connect to the specified controller auto ret = rc.connect(); // Respecify the destination controller and connect auto ret = rc.connect("192.168.0.20", 10002);</pre>		
Python			
Signature	def connect(self, ip_addr: str = "", port: int= 0) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	<pre># Connect to the specified controller ret = rc.connect() # Respecify the destination controller and connect ret = rc.connect("192.168.0.20", 10002)</pre>		
Notes			
If a connection is already established, disconnect it then reestablish the connection.			

Disconnection

Outline

Disconnects the robot controller.

C++

Signature	auto disconnect() -> RcResult
Parameter	—
Return value	Error No.
Example code	auto ret = rc.disconnect();

Python

Signature	def disconnect(self) -> RcResult
Parameter	Same as C++
Return value	Same as C++
Example code	ret = rc.disconnect()

Notes

Returns normally even if disconnected.

Timeout setting

Outline			
Sets the response timeout time from the robot controller.			
C++			
Signature	auto set_timeout(unsigned int msec) -> void		
Parameter	[in]	msec	Timeout time (Unit: milliseconds, Range: 1 or more)
Return value	—		
Example code	rc.set_timeout(1000);		
Python			
Signature	def set_timeout(self, msec: int) -> None		
Parameter	Same as C++		
Return value	Same as C++		
Example code	rc.set_timeout(1000)		
Notes			
The default is 30000 [ms] (30 seconds).			

Timeout acquisition

Outline

Acquires the response timeout time from the robot controller.

C++

Signature	auto get_timeout() -> unsigned int
Parameter	—
Return value	Timeout time (milliseconds)
Example code	auto msec = rc.get_timeout();

Python

Signature	def get_timeout(self) -> int
Parameter	Same as C++
Return value	Same as C++
Example code	msec = rc.get_timeout()

Notes

—

Basic information acquisition

Outline			
Acquires basic information from the robot controller.			
C++			
Signature	auto get_basic_info(BasicInfo& basic_info) -> RcResult		
Parameter	[out]	basic_info	Basic information
Return value	Error No.		
Example code	BasicInfo basic_info; auto ret = rc.get_basic_info(basic_info);		
Python			
Signature	def get_basic_info(self) -> tuple[RcResult, BasicInfo]		
Parameter	—		
Return value	1. Error No. 2. Basic information		
Example code	ret, basic_info = rc.get_basic_info()		
Notes			
—			

Servo ON/OFF

Outline

Turns on/off the servo power.

C++

Signature	auto turn_servo(bool onoff) -> RcResult		
Parameter	[in]	onoff	Servo power state (true: On, false: Off)
Return value	Error No.		
Example code	<pre>// Servo ON auto ret = rc.turn_servo(true); // Servo OFF auto ret = rc.turn_servo(false);</pre>		

Python

Signature	def turn_servo(self, onoff: bool) -> RcResult
Parameter	Same as C++
Return value	Same as C++
Example code	<pre># Servo ON ret = rc.turn_servo(True) # Servo OFF ret = rc.turn_servo(False)</pre>

Notes

—

Override settings

Outline			
Sets the override.			
C++			
Signature	auto set_override(unsigned int ovrd) -> RcResult		
Parameter	[in]	ovrd	Override value (Unit: %, Range: 1-100)
Return value	Error No.		
Example code	auto ret = rc.set_override(100);		
Python			
Signature	def set_override(self, ovrd: int) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	ret = rc.set_override(100)		
Notes			
—			

Override acquisition

Outline			
Acquires the override.			
C++			
Signature	auto get_override(unsigned int& ovrd) -> RcResult		
Parameter	[out]	ovrd	Override value (Unit: %)
Return value	Error No.		
Example code	unsigned int override{0}; auto ret = rc.get_override(override);		
Python			
Signature	def get_override(self) -> tuple[RcResult, int]		
Parameter	—		
Return value	1. Error No. 2. Override value (Unit: %)		
Example code	ret, ovrd = rc.get_override()		
Notes			
—			

Slot state acquisition

Outline			
Acquires the state of the program slot.			
C++			
Signature	auto get_slot_state(SlotState& state, int slot_no = 1) -> RcResult		
Parameter	[out]	state	State
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	SlotState state; auto ret = rc.get_slot_state(state);		
Python			
Signature	def get_slot_state(self, slot_no: int = 1) -> tuple[RcResult, SlotState]		
Parameter	[in]	slot_no	Slot No.
Return value	1. Error No. 2. Slot state		
Example code	ret, state = rc.get_slot_state()		
Notes			
—			

Program start

Outline			
Starts the program.			
C++			
Signature	auto start (int slot_no = 0 const std::string& prg_name = "", int mode = 0) -> RcResult		
Parameter	[in]	slot	Slot No.: (0: All slots, 1 or more: Specified slots, Range: 0 -maximum number of slots)
	[in]	prg_name	Program name (Empty: Selected program, Occupied: Specified program)
	[in]	mode	Operation mode (0: Continuous operation, 1: Cycle operation)
Return value	Error No.		
Example code	// Start all slots auto ret = rc.start(); // Start "PRG1" in Slot 1 auto ret = rc.start (1, "PRG1");		
Python			
Signature	def start(self, slot_no: int = 0, prg_name: str = "", mode: int = 0) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	# Start all slots ret = rc.start() # Start "PRG1" in Slot 1 ret = rc.start (1, "PRG1")		
Notes			
When slot = 0 (all slots), the program name and operation mode settings are ignored.			

Program stop

Outline

Stops the program.

C++

Signature	auto stop(int slot_no = 0) -> RcResult()		
Parameter	[in]	slot_no	Slot No.: (0: All slots, 1 or more: Specified slots, Range: 0 -maximum number of slots)
Return value	Error No.		
Example code	<pre>// Stop all slots rc.stop(); // Stop Slot 1 only rc.stop(1);</pre>		

Python

Signature	def stop(self, slot_no: int = 0) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	<pre># Stop all slots rc.stop() # Stop Slot 1 only rc.stop(1)</pre>		

Notes

Program reset

Outline			
Resets the program.			
C++			
Signature	auto reset (int slot_no = 0) -> RcResult		
Parameter	[in]	slot_no	Slot No.: (0: All slots, 1 or more: Specified slots, Range: 0 -maximum number of slots)
Return value	Error No.		
Example code	<pre>// Reset all slots rc.reset(); // Reset Slot 1 only rc.reset(1);</pre>		
Python			
Signature	def reset(self, slot_no: int = 0) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	<pre># Reset all slots rc.reset() # Reset Slot 1 only rc.reset(1)</pre>		
Notes			

Program slot initialization

Outline	
Initializes all program slots.	
C++	
Signature	auto init_slot() -> RcResult
Parameter	—
Return value	Error No.
Example code	auto ret = rc.init_slot();
Python	
Signature	def init_slot(self) -> RcResult
Parameter	Same as C++
Return value	Same as C++
Example code	ret = rc.init_slot()
Notes	
Resets the slot state at controller startup.	

Direct execution

Outline			
Executes programs directly.			
C++			
Signature	auto direct (const std::string& code, int timeout = 0, int slot_no = 1) -> RcResult		
Parameter	[in]	code	Execution information
	[in]	timeout	Timeout time (Unit: ms)
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	auto ret = rc.direct("MOV P1"); // Wait for execution to complete auto ret = rc.direct("MOV P1", -1); // Do not wait for execution to complete auto ret = rc.direct("MOV P1", 60000); // Wait up to 60 seconds		
Python			
Signature	def direct(self, code: str, timeout: int = 0, slot_no: int = 1) -> RcResult		
Parameter	Same as C++		
Return value	Same as C++		
Example code	ret = rc.direct("MOV P1") # Wait for execution to complete ret = rc.direct("MOV P1", -1) # Do not wait for execution to complete ret = rc.direct("MOV P1", 60000) # Wait up to 60 seconds		
Notes			
The timeout time settings are as follows.			
-1 or lower	Do not wait for execution to complete.		
0	Wait for execution to complete. The value specified by set_timeout is ignored.		
+1 or more	Waits for the specified time. The value specified by set_timeout is ignored. A timeout error is issued if execution is not completed after the specified time.		

Read variable contents (integer type)

Outline

Reads the values of integer-type program variables.

C++

Signature	<pre>auto get_val_integer(int& val, const std::string& val_name, int slot_no = 1) -> RcResult</pre>		
Parameter	[out]	val	Variable value
	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	<pre>int m1{0}; auto ret = rc.get_val_integer(m1, "M1%");</pre>		

Python

Signature	<pre>def get_val_integer(self, val_name: str, slot_no: int = 1) -> tuple[RcResult, int]</pre>		
Parameter	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	1. Error No. 2. Variable value		
Example code	<pre>ret, m1 = rc.get_val_integer("M1%")</pre>		

Notes

If the target variable is not an integer, it will not be read correctly.

If the numeric postfix operator is omitted, the variable becomes a single-precision real number.

(Example: In MELFA BASIC, if "M1=1" is written, the M1 variable becomes a single-precision real number and cannot be read by this function. Use "M1%=1" or "M1&=1".)

Read variable contents (real number type)

Outline			
Reads the values of real number-type program variables.			
C++			
Signature	auto get_val_float(double& val, const std::string& val_name, int slot_no = 1) -> RcResult		
Parameter	[out]	val	Variable value
	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	double m1{0.0}; auto ret = rc.get_val_float(m1, "M1#");		
Python			
Signature	def get_val_float(self, val_name: str, slot_no: int = 1) -> tuple[RcResult, float]		
Parameter	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	1. Error No. 2. Variable value		
Example code	ret, m1 = rc.get_val_float("M1#")		
Notes			
If the target variable is not a real number, it will not be read correctly.			

Read variable contents (string type)

Outline			
Reads string-type program variables.			
C++			
Signature	auto get_val_chars(std::string& val, const std::string& val_name, int slot_no = 1) -> RcResult		
Parameter	[out]	val	Variable value
	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	std::string c1; auto ret = rc.get_val_chars(c1, "C1");		
Python			
Signature	def get_val_chars(self, val_name: str, slot_no: int = 1) -> tuple[RcResult, str]		
Parameter	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	1. Error No. 2. Variable value		
Example code [B001]	ret, c1 = rc.get_val_chars("C1")		
Notes			
The string variables can only be alphanumeric.			

Read variable contents (position type)

Outline			
Reads the values of position-type program variables.			
C++			
Signature	auto get_val_pos(Position& val, const std::string& val_name, int slot_no = 1) -> RcResult		
Parameter	[out]	val	Variable value
	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	Position p1; auto ret = rc.get_val_pos(p1, "P1");		
Python			
Signature	def get_val_pos(self, val_name: str, slot_no: int = 1) -> tuple[RcResult, Position]		
Parameter	[in]	val_name	Variable name
	[in]	slot_no	Slot No.
Return value	1. Error No. 2. Variable value		
Example code	ret, p1 = rc.get_val_pos("P1")		
Notes			
—			

Read variable contents (joint type)

Outline			
Reads the values of joint-type program variables.			
C++			
Signature	auto get_val_joint(Joint& val, const std::string& val_name, int slot_no = 1) -> RcResult		
Parameter	[out]	val	Variable value
	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	Error No.		
Example code	Joint j1; auto ret = rc.get_val_joint(j1, "J1");		
Python			
Signature	def get_val_joint(self, val_name: str, slot_no: int = 1) -> tuple[RcResult, Joint]		
Parameter	[in]	val_name	Variable name
	[in]	slot_no	Slot No. (Range: 1-maximum number of slots)
Return value	1. Error No. 2. Variable value		
Example code	ret, j1 = rc.get_val_joint("J1")		
Notes			
—			

Error reset

Outline	
Resets the current error.	
C++	
Signature	auto reset_error() -> RcResult
Parameter	—
Return value	Error No.
Example code	auto ret = rc.reset_error();
Python	
Signature	def reset_error(self) -> RcResult
Parameter	Same as C++
Return value	Same as C++
Example code	ret = rc.reset_error()
Notes	
—	

Current position acquisition (joint)

Outline

Acquires the current position of the robot arm in the joint coordinate system.

C++

Signature	auto get_curpos_joint(Joint& joint) -> RcResult		
Parameter	[out]	joint	Current position (joint)
Return value	Error No.		
Example code	Joint joint; auto ret = rc.get_curpos_joint(joint);		

Python

Signature	def get_curpos_joint(self) -> tuple[RcResult, Joint]		
Parameter	—		
Return value	1. Error No. 2. Current position (joint)		
Example code	ret, joint = rc.get_curpos_joint()		

Notes

—

Current position acquisition (XYZ)

Outline

Acquires the current position of the robot arm in the XYZ coordinate system.

C++

Signature	auto get_curpos_xyz(Position& position) -> RcResult		
Parameter	[out]	position	Current position (XYZ)
Return value	Error No.		
Example code	Position position; auto ret = rc.get_curpos_xyz(position);		

Python

Signature	def get_curpos_xyz(self) -> tuple[RcResult, Position]
Parameter	—
Return value	1. Error No. 2. Current position (XYZ)
Example code	ret, position = rc.get_curpos_xyz()

Notes

—

4.6 BasicInfo class

Method list

No.	Method name	Function	Compatible version
1.	—	—	—

Field list

No.	Field name	Type (C++/Python)	Description	Compatible version
1.	rb_name	std::string / str	Robot model name (Example: RV-2FR-D)	1.0 or later
2.	rc_name	std::string / str	Robot controller name (Example: CR800-D)	1.0 or later
3.	fw_version	std::string / str	Firmware version (Example: A1)	1.0 or later
4.	slot_max	int / int	Maximum number of slots	1.0 or later

4.7 SlotState class

Method list

No.	Method name	Function	Compatible version
1.	—	—	—

Field list

No.	Field name	Type (C++/Python)	Description	Compatible version
1.	servo	bool / bool	Servo ON/OFF state	1.0 or later
2.	err_no	int / int	Current error number	1.0 or later
3.	run_state	bool / bool	Program run state	1.0 or later
4.	step_no	int / int	Program execution line	1.0 or later

4.8 Joint class

Method list

No.	Method name	Function	Compatible version
1.	—	—	—

Field list

No.	Field name	Type (C++/Python)	Description	Compatible version
1.	j1	double / float	Axis 1 position	1.0 or later
2.	j2	double / float	Axis 2 position	1.0 or later
3.	j3	double / float	Axis 3 position	1.0 or later
4.	j4	double / float	Axis 4 position	1.0 or later
5.	j5	double / float	Axis 5 position	1.0 or later
6.	j6	double / float	Axis 6 position	1.0 or later
7.	j7	double / float	Axis 7 position	1.0 or later
8.	j8	double / float	Axis 8 position	1.0 or later

4.9 Position class

Method list

No.	Method name	Function	Compatible version
1.	—	—	—

Field list

No.	Field name	Type (C++/Python)	Description	Compatible version
9.	x	double / float	X axis position	1.0 or later
10.	y	double / float	Y axis position	1.0 or later
11.	z	double / float	Z axis position	1.0 or later
12.	a	double / float	X axis rotation	1.0 or later
13.	b	double / float	Y axis rotation	1.0 or later
14.	c	double / float	Z axis rotation	1.0 or later
15.	fl1	unsigned int / int	Posture flag	1.0 or later
16.	fl2	unsigned int / int	Multi-rotation data	1.0 or later
17.	l1	double / float	Additional axis position 1	1.0 or later
18.	l2	double / float	Additional axis position 2	1.0 or later

5 SAMPLE CODE

This is an example of performing different kinds of reset operations and then executing the program. Error handling and other details are omitted. Add them to the program if required.

main.cpp

```
#include <iostream>
#include "melfa_api.hpp"

using namespace MELFA;

int main()
{
    // Create an instance (Specify the IP address and port number appropriately.)
    RobotController rc("192.168.0.20", 10002);

    // Connect to robot controller.
    if (rc.connect() != MR_OK)
    {
        std::cerr << "Failed to connect." << std::endl;
        return -1;
    }

    // Check robot information.
    BasicInfo basic_info;
    rc.get_basic_info(basic_info); // Acquire basic information.
    std::cout << basic_info << std::endl;

    // Check current position.
    Position position;
    rc.get_curpos_xyz(position); // Acquire current position.
    std::cout << position << std::endl;

    // Reset state.
    rc.stop(); // Stop program.
    rc.reset_error(); // Reset error.
    rc.reset(); // Reset program.

    // Servo ON.
    rc.turn_servo(true);

    // Execute program "MXT".
    rc.start(1, "MXT");

    // Disconnect robot controller.
    rc.disconnect();

    return 0;
}
```

```

from melfa_api import RobotController, RcResultCode

def main() -> None:
    # Create an instance (Specify the IP address and port number appropriately.)
    rc = RobotController("192.168.0.20", 10002)

    #Connect to robot controller.
    ret = rc.connect()
    if ret != RcResultCode.MR_OK:
        print("Failed to connect.")
        return

    # Check robot information.
    ret, basic_info = rc.get_basic_info()
    print(basic_info)

    # Check current position.
    ret, position = rc.get_curpos_xyz()
    print(position)

    # Reset state.
    rc.stop() # Stop program.
    rc.reset_error() # Reset error.
    rc.reset() # Reset program.

    # Servo ON.
    rc.turn_servo(True)

    # Execute program "MXT".
    rc.start(1, "MXT")

    # Disconnect robot controller.
    rc.disconnect()

if __name__ == "__main__":
    main()

```

MEMO

REVISIONS

*The manual number is on the bottom left of the back cover.

Revision date	*Manual No.	Description
January.2026	BFP-A3938-A	• First edition

This manual confers no industrial property rights or any rights of any other kind, nor does it confer any patent licenses. Mitsubishi Electric Corporation cannot be held responsible for any problems involving industrial property rights which may occur as a result of using the contents noted in this manual.

© 2026 MITSUBISHI ELECTRIC CORPORATION

TRADEMARKS

No part of this manual may be reproduced by any means or in any form, without prior consent from Mitsubishi Electric. Information contained in this document is subject to change without notice.

The company names, system names, and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.

In some cases, trademark symbols such as TM or ® are not specified in this manual.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

The official name of Windows® is Microsoft® Windows® Operating System.

ETHERNET is a registered trademark of FUJIFILM Business Innovation Corp. Ethernet is a registered trademark of Xerox Corporation in the United States.

The other company names and product names mentioned in this document are either registered trademarks or trademarks of their respective companies.

COPYRIGHT INFORMATION

Screenshots are used in accordance with Microsoft Corporation guidelines.

BFP-A3938-A(2601)MEE

mitsubishi electric corporation

HEAD OFFICE: TOKYO BLDG., 2-7-3, MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS: 1-14, YADA-MINAMI 5-CHOME, HIGASHI-KU, NAGOYA 461-8670, JAPAN

Specifications subject to change without notice.