FANUC Robot series

R-50*i*A/R-50*i*A Mate Controller

Python

OPERATOR'S MANUAL

B-84464EN-26/01

© FANUC CORPORATION, 2025

Important notice

Thank you very much for purchasing FANUC Robot.

Before using the Robot, be sure to read the "FANUC Robot series SAFETY HANDBOOK (B-80687EN)" and understand the content.

- No part of this manual may be reproduced in any form.
- The appearance and specifications of this product are subject to change without notice.
- Any copyright and other intellectual property rights regarding this manual shall belong to FANUC.
- The safety precautions described in this manual may no longer be accurate and up-to-date due to revisions or changes in laws and regulations. Before using our products, each user must confirm the latest laws and regulations.

The products in this manual are controlled based on Japan's “Foreign Exchange and Foreign Trade Law". The export from Japan may be subject to an export license by the government of Japan. Further, re-export to another country may be subject to the license of the government of the country from where the product is re-exported. Furthermore, the product may also be controlled by re-export regulations of the United States government. Should you wish to export or re-export these products, please contact FANUC for advice.

In this manual, we endeavor to include all pertinent matters.

There are, however, a very large number of operations that must not or cannot be performed, and if the manual contained them all, it would be enormous in volume.

It is, therefore, requested to assume that any operations that are not explicitly described as being possible are "not possible".

Safety precautions

To ensure the safety of users and prevent damage to the machine, this manual indicates each precaution on safety with "WARNING" or "CAUTION" according to its severity.

Supplementary information is indicated by "NOTE".

Read the contents of each "WARNING", "CAUTION" and "NOTE" before attempting to use the machine.

⚠WARNING
Used if a danger resulting in the death or serious injury of users are expected to occur if they fail to observe the approved procedure.

⚠CAUTION
Used if a danger resulting in the minor or moderate injury of users or equipment damage is expected to occur if they fail to observe the approved procedure.

NOTE
Used if a supplementary explanation not related to any of WARNING and CAUTION is to be indicated.

Table of contents

Important notice ii
Safety precautions iii
I OVERVIEW 1
1. Overview 3
1.1. Software & Hardware Requirements 3
1.2. Limitations 3
2. PROCEDURE TO RUN PYTHON 4
2.1. Preparing Script Files 4
2.2. Enabling Python Execution 4
2.3. Executing and Aborting a Python Script from a TP Program 4
2.4. KAREL BUILT-IN FUNCTIONS 5
2.4.1. RUN_PYTHON Built-in Function 5
2.4.2. ABORT_PYTHON Built-in Function 6
2.4.3. STATUS_PYTHON Built-in Function 6
3. REMOTE DEBUGGING 7
3.1. Setup 7
3.1.1. Controller Settings 7
3.1.2. PC Settings 8
3.2. Debug Method 9
4. ENVIRONMENT AND SPECIFICATIONS 12
4.1. Language Specifications 12
4.2. File System 12
4.3. Standard Library 12
II Reference 15
1. fanucrobot LIBRARY REFERENCE 17
1.1. Robot Position Object 17
1.1.1. Cartesian Class 17
1.1.2. Joint Class 18
1.2. robotif Package 19
1.2.1. Setup 20
1.2.2. Sample Scripts 21
1.2.2.1. Read/Write I/O 21
1.2.2.2. Read/Write Registers 22
1.2.2.3. Read/Write Position Registers 23
1.2.3. Core Class 24
1.2.3.1. Core.connect Method 24
1.2.3.2. Core.disconnect Method 24
1.2.3.3. Core.set_timeout Method 25
1.2.3.4. Core.get_timeout Method 25
1.2.3.5. Core.read_{I/O} Methods 25
1.2.3.6. Core.write_{I/O} Methods 25
1.2.3.7. Core.get_datatable Method 25
1.2.3.8. Core.get_datatable2 Method 25
1.2.3.9. Core.get_available Method 26
1.2.3.10. Core.clear_alarm Method 26
1.2.4. DataTable Class 26
1.2.4.1. DataTable.clear Method 26
1.2.4.2. DataTable.add_curpos Method 27
1.2.4.3. DataTable.add_numreg Method 27
1.2.4.4. DataTable.add_posreg Method 27
1.2.4.5. DataTable.add_variable Method 27

1.2.4.6. DataTable.add_varpos Method 28
1.2.4.7. DataTable.add_task Method 28
1.2.4.8. DataTable.add_alarm Method 29
1.2.4.9. DataTable.add_string Method 29
1.2.4.10. DataTable.add_io Method 30
1.2.4.11. DataTable.refresh Method 30
1.2.5. DataCurPos Class 30
1.2.5.1. DataCurPos.getvalue Method 31
1.2.6. DataNumReg Class 31
1.2.6.1. DataNumReg.setvalues Method 31
1.2.6.2. DataNumReg.getvalue Method 31
1.2.7. DataPosReg Class 31
1.2.7.1. DataPosReg.setvalue Method 32
1.2.7.2. DataPosReg.getvalue Method 32
1.2.8. DataVariableClass 32
1.2.8.1. DataVariable.setvalue Method 32
1.2.8.2. DataVariable.getvalue Method 32
1.2.9. DataVarPos Class 33
1.2.9.1. DataVarPos.setvalue Method 33
1.2.9.2. DataVarPos.getvalue Method 33
1.2.10. DataTask Class 33
1.2.10.1. DataTask.getvalue Method 33
1.2.11. DataAlarm Class 34
1.2.11.1. DataAlarm.getvalue Method 34
1.2.12. DataStringClass 35
1.2.12.1. DataString.setvalue Method 35
1.2.12.2. DataString.getvalue Method 35
1.2.12.3. DataString.update Method 35
1.2.12.4. DataString.reset Method 36
1.2.13. DataIO Class 36
1.2.13.1. DataIO.setvalues Method 36
1.2.13.2. DataIO.getvalue Method 36
1.2.13.3. DataIO.update Method 36
1.2.13.4. DataIO.reset Method 36
1.3. rmi package 37
1.3.1. Setup 37
1.3.2. Sample Scripts 37
1.3.2.1. Simple Single Group Motion Instructions 37
1.3.2.2. Incremental Motion Instruction 38
1.3.2.3. Motion Instruction for Multi-Group System 40
1.3.3. RMIClass : Communication Methods and Constructor 41
1.3.3.1. Constructor 41
1.3.3.2. RMI. connect Method 41
1.3.3.3. RMI.disconnect Method 41
1.3.4. RMIClass: Command Method 41
1.3.4.1. RMI.init_move Method 41
1.3.4.2. RMI.abort Method 42
1.3.4.3. RMI.pause Method 42
1.3.4.4. RMI.resume Method 42
1.3.4.5. RMI.read_alarm Method 42
1.3.4.6. RMI.get_frame_number Method 42
1.3.4.7. RMI.set_frame_number Method 43
1.3.4.8. RMI.get_status Method 43
1.3.4.9. RMI.read_uframe Method 43
1.3.4.10. RMI.write_uframe Method 44
1.3.4.11. RMI.read_utool Method 44
1.3.4.12. RMI.write_utool Method 44

1.3.4.13. RMI.read_di Method 44
1.3.4.14. RMI.write_do Method 44
1.3.4.15. RMI.curpos_cartesian Method 45
1.3.4.16. RMI.curpos_joint Method 45
1.3.4.17. RMI.set_override Method 45
1.3.4.18. RMI.read_position_register Method 45
1.3.4.19. RMI.write_position_register Method 45
1.3.4.20. RMI.reset_alarm Method 46
1.3.4.21. RMI.get_tcp_speed Method 46
1.3.5. RMIClass: Instruction Method 46
1.3.5.1. RMI.wait_di Method 46
1.3.5.2. RMI.wait_time Method 47
1.3.5.3. RMI.sel_uframe Method 47
1.3.5.4. RMI.sel_utool Method 47
1.3.5.5. RMI.sel_payload Method 47
1.3.5.6. RMI.call Method 47
1.3.5.7. RMI.linear Method 47
1.3.5.8. RMI.linear_incremental Method 49
1.3.5.9. RMI.joint Method 49
1.3.5.10. RMI.joint_incremental Method 49
1.3.5.11. RMI.circular Method 49
1.3.5.12. RMI.circular_incremental Method 49
1.3.6. RMIClass: Other API 50
1.3.6.1. RMI.wait_progress Method 50
1.3.6.2. RMI.end_move Method 50
1.3.6.3. RMI.motion_option Method 50
1.3.6.4. RMI._seqid Member Variable 51
1.3.6.5. RMI._done_id Member Variable 51
1.3.6.6. RMI.RMIError Exception 51
Revision record 52

I OVERVIEW

1. Overview

The R-50iA robot controller can use Python® for system development. In Python on R-50iA, it is possible to do following:

- Execute Python from a TP program or KAREL program
- Read or write various robot data with Modbus TCP of HMI Device Communication (R553)
- Execute robot motion with the Remote Motion Interface (R912)
- Perform remote debugging with the Visual Studio Code (VSCode) extension

This manual describes how to use Python on an R-50iA robot controller.

(Python is a registered trademark of the Python Software Foundation in the United States and other countries.)
(Visual Studio Code is a registered trademark of Microsoft Corporation in the United States and other countries.)

1.1. Software & Hardware Requirements

Hardware

A FANUC R-50iA controller is required.

Software

The software version must be V10.10P/11 (7DH1/11) or later.

(*) There may be changes to specifications and available functions in future software updates.

1.2. Limitations

- The standard output when running Python from a TP program or KAREL program cannot be displayed on the iPendant screen. If a text file named "pylog.txt" exists at the root of MC:, the standard output will be redirected to the file.
- Standard input and interactive mode are unavailable.

2. PROCEDURE TO RUN PYTHON

In this chapter, how to execute a Python script from TP program or KAREL is described.

2.1. Preparing Script Files

Python script files (.py) stored in FR:, MC:, and UD1: can be executed. When storing in FR:, the files need to be stored in FR:/PYTHON. When storing in other directories, backup is unavailable. When a script file is loaded from the FILE screen, it will be copied to FR:/PYTHON.

When using subdirectories, store the script files in either the MC: or UD1: directory.

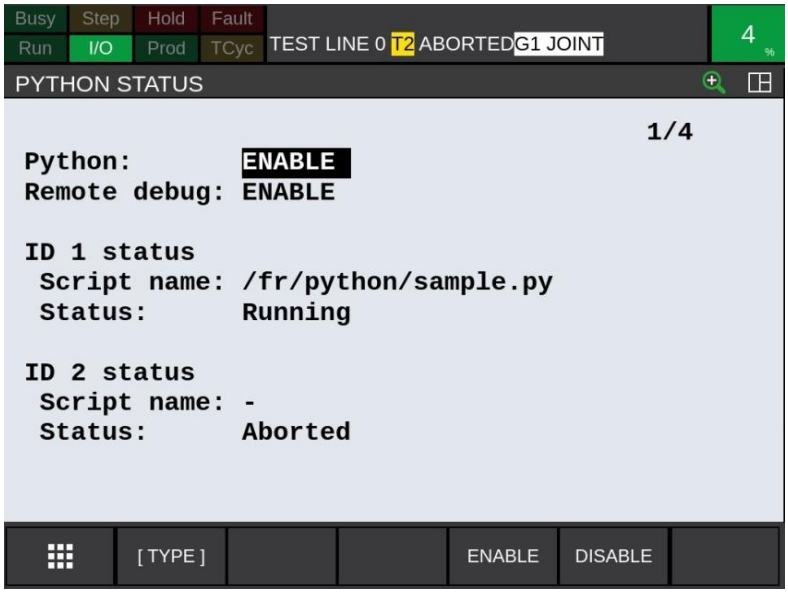
2.2. Enabling Python Execution

To enable Python execution on the controller, do the following:

1 **Press MENU.**

2 **Select "0 NEXT", select STATUS, and then select "Python". The following PYTHON STATUS screen will be displayed.**

3 **Move the cursor to Python and press F4, ENABLE, and then cycle controller power.**



Up to two Python instances can be executed from a TP program or KAREL program at the same time. The Python Status screen displays the current Python's script names and their statuses.

2.3. Executing and Aborting a Python Script from a TP Program

You can teach a TP program to execute Python using the following procedure. Note that Python execution is asynchronous with the execution of a TP program.

1 **On the program edit screen, press F1, [INST] and then select CALL.**

2 **Select "CALL program" from CALL instruction menu.**

3 **Press F3, [COLLECT] and then select KAREL Progs. Select RUNPY from the list displayed.**

```
   1:  CALL RUNPY(script name='sample.py',
   :  pythonID R[]=1)
```

- The first argument is the Python script name. If the script file is in FR:\PYTHON, the device name and path can be omitted as shown above. If the script file is in other paths, specify the absolute path like the following example: ‘ud1:\test\sample.py’.
- The second argument is the register index to output the Python ID. The Python ID is the identifier to distinguish between running Python instances. There will be no Python ID output if 0 is specified.
- The third and greater arguments are optional. These arguments will be converted into appropriate strings, and then passed to Python as command line arguments. For example, these arguments will be converted into strings like '1' or '0' for Boolean, and '100' or '3.14000E+00' for Integers. Data that cannot be converted into strings, like position data, are not allowed to be passed as command line arguments. Up to 15 command line arguments can be passed on, and any further arguments will be ignored.

To abort a running Python instance from a TP program perform the following step.

4 From the KAREL Progs list (F3, [COLLECT] > KAREL Progs), select ABORTPY.

```
   1:  CALL ABORTPY(pythonID=1)
```

5 Specify the Python ID to be aborted. All running Python instances will be aborted if 0 is specified.

2.4. KAREL BUILT-IN FUNCTIONS

KAREL built-in functions are built-in functions (or routines) pre-defined in the KAREL language.

The KAREL language can be used to execute, abort, and obtain the statuses of Python instances, using the built-in functions in this section.

Please refer to "OPERATOR’S MANUAL(Basic Operation)"(B-84464EN) for details on KAREL programs.

2.4.1. RUN_PYTHON Built-in Function

Purpose :

To run Python asynchronously.

Syntax :

python_id = RUN_PYTHON(script_name, status)

[in] script_name: STRING

[out] status: INTEGER

[return] python_id: INTEGER

Details :

- The “script_name” argument indicates the name of the Python script file. If the script is in the FR:\PYTHON directory, the device name and path can be omitted.

When adding arguments that are separated by spaces in the “script name” argument as shown below, they can be passed as command line arguments.

python_id = RUN_PYTHON(‘sample.py 1 2 3’, status) → command line arguments: ‘1’, ‘2’, ‘3’

- The “status” argument indicates the calling status of the Python script. If not 0, then an error has occurred.
- This function returns “python_id”, which is the identifier of the running Python instance.

2.4.2. ABORT_PYTHON Built-in Function

Purpose :

To abort a Python instance.

Syntax :

ABORT_PYTHON(python_id, status)

[in] python_id: INTEGER

[out] status: INTEGER

Details :

- The “python_id” argument indicates the Python ID to be aborted.
- The “status” argument indicates the aborting status of Python. If not 0, then an error has occurred.

2.4.3. STATUS_PYTHON Built-in Function

Purpose :

To obtain the execution status of the current Python instance.

Syntax :

python_status = STATUS_PYTHON(python_id, script_name, status)

[in] python_id: INTEGER

[out] script_name: STRING

[out] status: INTEGER

[return] python_status: INTEGER

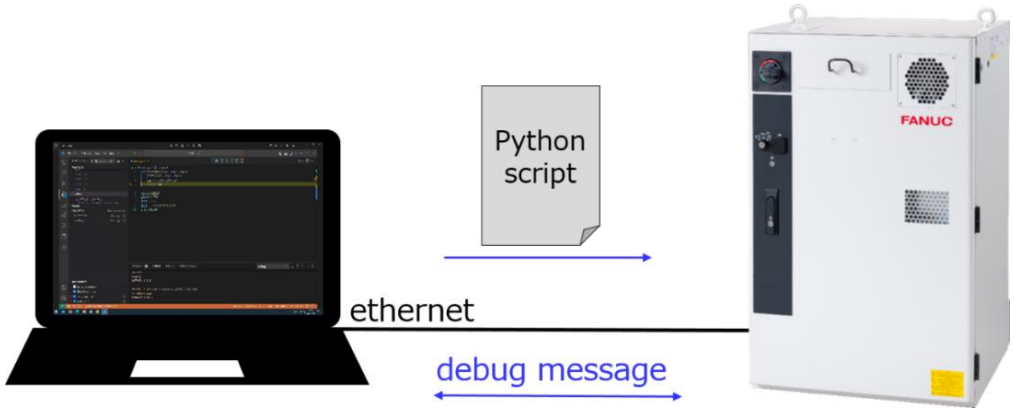
Details :

- The “python_id” argument indicates which Python ID status to obtain.
- The “script_name” argument indicates the name of the script file. This argument is filled by a NULL string if the specified Python ID is not being executed. The maximum size of the string returned is 64 bytes. This should be declared as a STRING with a length of at least 64 bytes.
- The “status” argument indicates the status of getting Python's status. If not 0, then an error has occurred.
- This function returns “python_status”, which indicates the execution status of the Python instance.

The possible return values and their meanings are as follows:

- 0 = Aborted
- 1 = Running
- 2 = Paused

3. REMOTE DEBUGGING



With Python remote debugging, it is possible to use functions like breakpoints when debugging. The script files are edited on VSCode, and transferred to the controller via LAN for execution.

This chapter describes how to debug Python on an R-50iA controller with VSCode on a PC over Ethernet (LAN).

3.1. Setup

Please follow the instructions below to setup a development environment for remote debugging.

3.1.1. Controller Settings

TCP/IP Settings

Follow the steps in "Network and Security Function OPERATOR'S MANUAL"(B-84604EN) Section "3.3.1 Defining TCP/IP Parameters" to define TCP/IP parameters of controller.

Enabling Remote Debugging

To enable remote debugging, perform the following steps:

1. **Perform Steps 1-4 in "I.2.2. Enabling Python Execution(P.4)" to display the PYTHON STATUS screen.**
2. **Move the cursor to Python and press F4, ENABLE to enable Python execution.**
3. **Move the cursor to Remote debug and press F4, ENABLE to enable remote debugging.**
4. **Cycle controller power to apply the settings.**

Creating a User for Debug Connection

To connect to controller with VSCode, the Password Function password level must be SETUP level or above. "Admin" and "Guest" users cannot be used for debug connection.

If a password user of SETUP level or above has not been created, perform the following steps. Refer to the "OPERATOR'S MANUAL(Basic Operation)"(B-84464EN)Section 8.11.4 "Password Operations by INSTALL or PWDMGR Users" for more information.

For example, if a password user "user1" of SETUP level is created, a screen like the following will be shown.

Password Users

	USERNAME	PWD	LEVEL	TIME	L
1	@Admin	-	PWDMGR	24	-
2	Guest	-	SETUP	0	-
3	user1	-	SETUP	24	-
4		*	**********	0	*
5		*	**********	0	*
6		*	**********	0	*
7		*	**********	0	*
8		*	**********	0	*
9		*	**********	0	*
10		*	**********	0	*

The authority to access the debugging connection is included in the SETUP level by default. To change this authority to another password level, refer to"OPERATOR'S MANUAL(Basic Operation)"(B-84464EN) Section 8.11.8.7 "Features tag", and set const=28 to allow or disallow the access to the feature for the specified password level.

3.1.2. PC Settings

PC Requirements

To use remote debugging, the PC needs to meet the following requirements:

- The PC must be Microsoft Windows® 10 or above.
- The PC must have VSCode installed.

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

Installing a vsix File

Install the remote debugger for Python extension in VSCode.

1 **Obtain the vsix file of the extension. This vsix file is distributed on the Customer Portal or each overseas sales branche. If the method of obtaining this file is unclear, contact FANUC customer support for more information.**

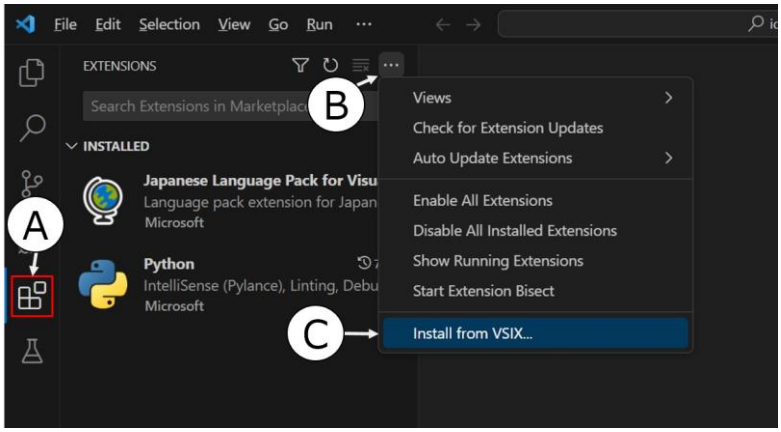
2 **Start VSCode.**

3 **Install the vsix file in VSCode as shown below. Refer to the image below.**

A: Select "Extensions" view.

B: Select "…" menu at the top right.

C: Select "Install from VSIX…", then select "Python-for-FANUC-Robot-O.O.O.vsix" obtained from 1 and install it.



4 **Restart VSCode after the installation is finished.**

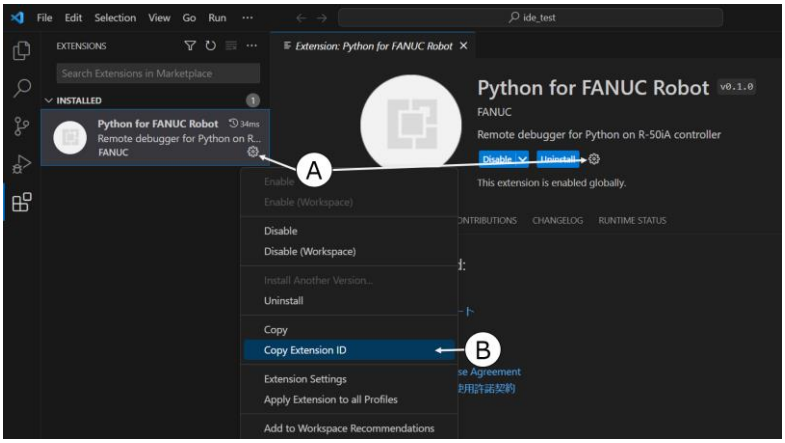
Extension Settings

Configure settings to connect VSCode to the controller using the following steps:

1 Open Extension Settings. Refer to the image below.

A: Select the gear button of the extension.

B: Select "Extension Settings".



2 Read the Software License Agreement in the Extension DETAILS page, and check the checkbox if you agree to the agreement. If the checkbox is not checked, this extension will not function. ("Host Name" in the following diagram)

3 Input the IP address of the controller. (2. in the following diagram)

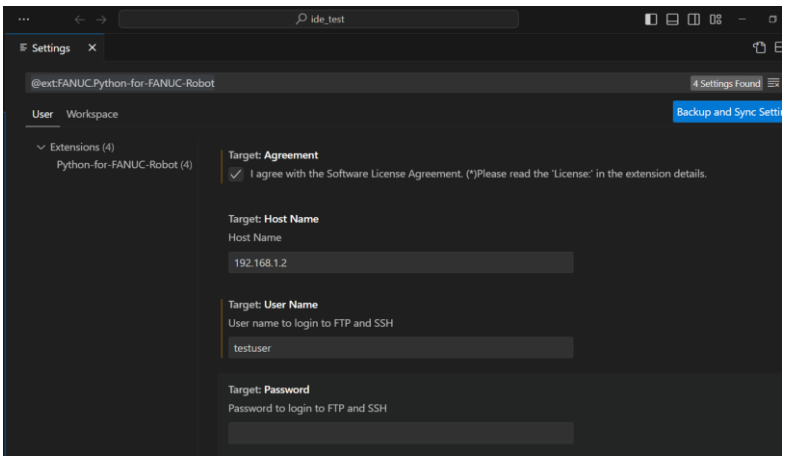
4 Set the user name and password used to login as a user for the debug connection in the controller when remote debugging on Python. ("UserName" and "Password" in the following diagram)

Refer to "Debug Connection(P.7)" in Subsection 3.1.1 for details on login user.

If the user name and password are left empty, you will be required to input them when using remote debugging.

CAUTION

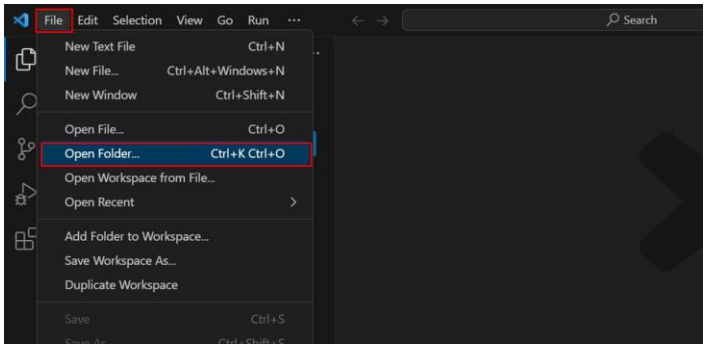
For security measures, it is recommended to leave the password column empty.



3.2. Debug Method

Take the following steps to debug Python.

1 Open your working folder on VSCode. To open the working folder, click “File” from the toolbar, then select “Open Folder…” from the dropdown menu as shown in the following diagram, and create a folder for the script files on the PC.

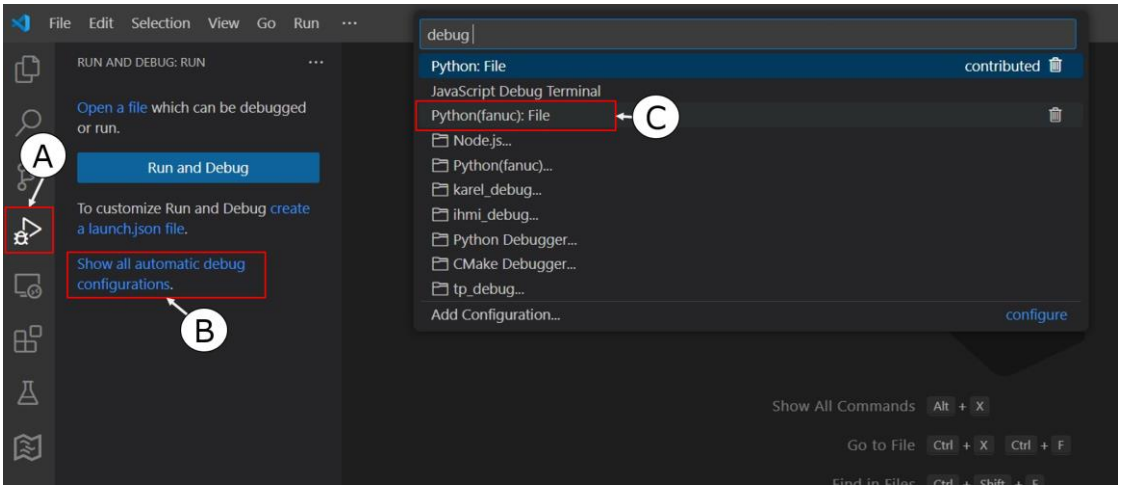


2 Select debug configurations to enable debugging with this extension (Only once). Refer to the following image.

A: Select "Run and Debug" view.

B: Select "Show all automatic debug configurations".

C: Select "Python(fanuc): File".



The added debug configuration "Python(fanuc): File" above is used when remote debugging for Python on controller. When debugging Python on PC with another extension, it is necessary to change to an appropriate debug configuration.

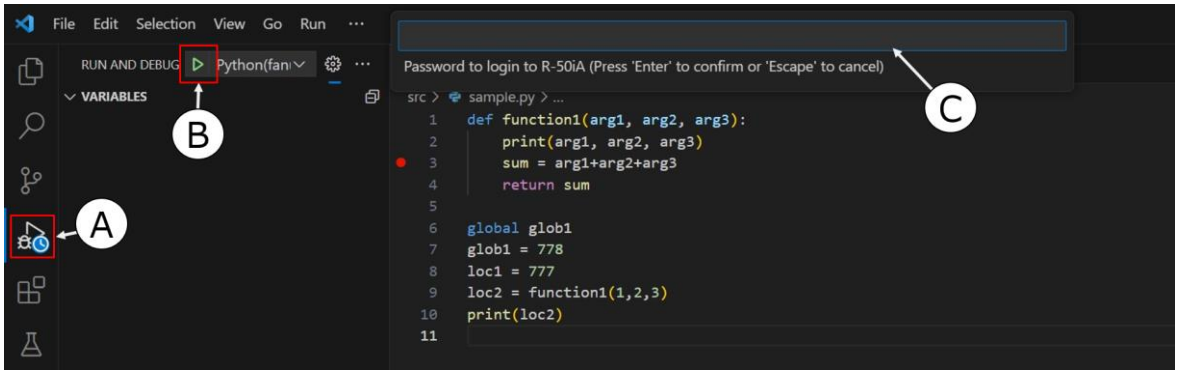
3 Create a Python script in the working folder.

4 With the script file open, press the Run and Debug button to start. Refer to the image below.

A: Select "Run and Debug" view.

B: Press the "Start Debugging" button.

C: (Only if not set in Extension Settings) Input the login user name and password at the top of the screen.



5 When the button is clicked, the script file is transferred to FR:/PYTHON/ directory of the controller and executed at the same time. Debugging can be done as shown in the following diagram.

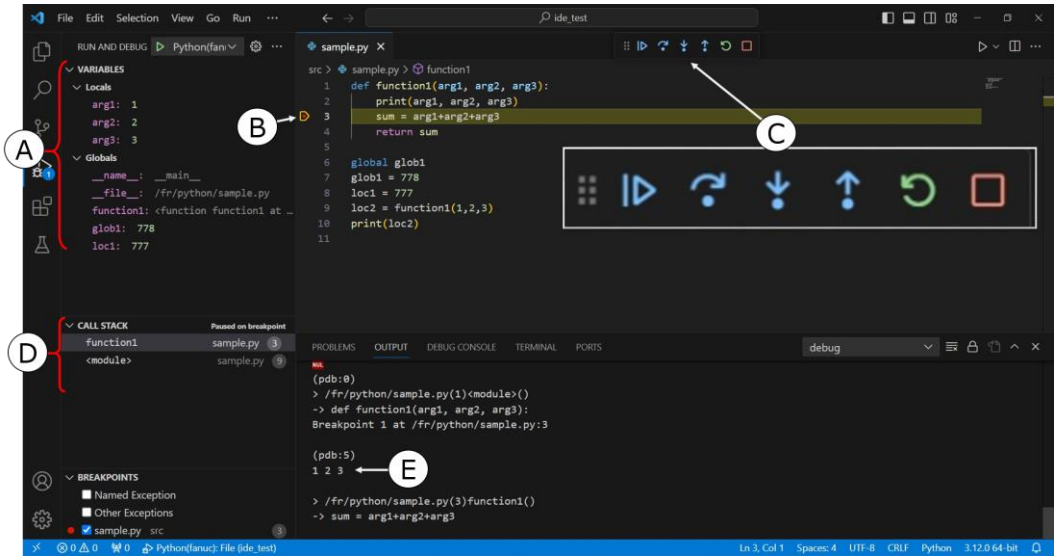
A : Variables view

B : Breakpoints

C: Debug toolbar (pause/continue, step over, step into, step out, restart, abort)

D: Call stack

E: Standard output



Limitations

- Breakpoints cannot be added when Python is running. If a breakpoint is added while Python is running, the program will be paused.
- Breakpoint view (bottom left of the above image) is not supported.

4. ENVIRONMENT AND SPECIFICATIONS

This chapter describes the basic specification of Python and the available standard libraries.

4.1. Language Specifications

Version

Python version 3.11.5

OS Environment

The availability and specifications of functions in Python are commonly dependent on the operating system. The availability and specifications of functions in Python on an R-50iA controller are mainly based on Unix. However, not all functions that work on Unix are available.

4.2. File System

File Path Format

The file path used in Python is different from that of KAREL.

The directory names are separated by a “/” (slash).

When using an absolute path, start the path name with a “/”, and the “:” (colon) after the device name is removed.

File I/O Devices

You can access the following file I/O devices in Python.

- FROM Disk (FR:)
  Specify the path starting with /fr/
- Memory Card (MC:)
  Specify the path starting with /mc/
- USB Flash Drive (UD1:)
  Specify the path starting with /ud1/

For example, the following is used when opening the test.txt file in UD1: with the built-in function open().

fp = open('/ud1/test.txt', 'r')

4.3. Standard Library

The Python standard library contains various built-in modules and libraries. The current availability of the Python standard library on the controller is shown in the following table. Do not use libraries that are not listed in the table. For details on the standard library, refer to the official Python library reference.

List of Tested Libraries/Modules

The tested Libraries / Modules are listed in the table below.

Library Name	Category
os (*1)	Generic operating system services

Library Name	Category
The available APIs: os.name, os.error, os.mkdir(), os.rename(),os.remove()	
time	
argparse	
errno	
string	Text processing services
re	
textwrap	
stringprep	
struct	Binary data services
codecs	
datetime	Basic date and time types
calendar	
collections	Data types
heapq	
bisect	
array	
weakref	
types	
copy	
enum	
math	Numeric and mathematical modules
fractions	
random	
itertools	Functional programming modules
functools	
operator	
csv	File format
configparser	
base64	Data handling
binascii	
json	
gettext	Internationalization
locale	
typing	Development tools
sys	Python runtime services
keyword	Python language services
socket	Network communication

II Reference

1. fanucrobot LIBRARY REFERENCE

The “fanucrobot” is a Python library to make programming FANUC robots easier.

This library consists of two packages, “robotif” (Robot Interface) and “rmi”, which are the wrappers for HMI Device Communication (R553) and Remote Motion Interface (R912) respectively.

For detailed information on the HMI Device Communication function and Remote Motion Interface function, refer to manuals "Optional Function OPERATOR’S MANUAL"(B-84464EN-2) and B-84184EN/02 respectively.

NOTE

If the North America Setting (R650) is selected, the HMI Device Communication(R553) option is required to use robotif.
If the Standard Setting (R651) is selected, robotif is available by default.

NOTE

The Remote Motion Interface (R912) option is required to use rmi.

This library is built into the controller, and is available when using Python on the controller.

1.1. Robot Position Object

The following classes are defined in the fanucrobot.position module to allow handling of robot position data. These classes and constant (ACTIVE_FRAME) are used in both robotif and rmi. And can also be imported from fanucrobot.robotif or fanucrobot.rmi.

1.1.1. Cartesian Class

The fanucrobot.position.Cartesian class handles robot position data with Cartesian representation. The class members in the following table are used as coordinates.

Member	Type	Description
X	float	Cartesian elements of 6-axis robot [mm] X, Y, Z : [mm] W, P, R : [deg] Default: 0.0
Y		
Z		
W		
P		
R		
Ext1		Coordinates of extended axes Default: 0.0
Ext2		
Ext3		
UToolNumber	int	Tool frame number Default : 1
UFrameNumber	int	User frame number Default : 0
Flip	bool	CONFIG True: **F**lip False: **N**oFlip Default: False
Left		CONFIG True: **L**eft False: **R**ight Default: False
Up		CONFIG True: **U**p False: **D**own Default: True
Front		CONFIG True: Fron**T** False: **B**ack Default: True
Turn4	int	Turn number of J4, J5, J6 axes Default: 0
Turn5		

Member	Type	Description
Turn6		

The UToolNumber and UFrameNumber can be ACTIVE_FRAME (= -1), which means the currently selected frame number.

It may be more efficient to edit the position data all at once like the script shown below.

```
from fanucrobot.position import Cartesian, Joint

position = Cartesian()
vars(position).update({‘X’:1500, ‘Z’:1000, ‘W’:-180})
```

1.1.2. Joint Class

The fanucrobot.position.Joint class handles robot position data with Joint representation. The class members in the following table are used as coordinates.

Member	Type	Description
J1	float	Joint elements [deg] Default: 0.0
J2		
J3		
J4		
J5		
J6		
J7		
J8		
J9		

1.2. robotif Package

The fanucrobot.robotif allows Python to access various data on the robot. The following table shows the robot data that are accessible.

Access Method	Data	Object
Read/Write	DO	Core, DataIO
	RO	Core, DataIO
	GO	Core, DataIO(*)
	AO	Core, DataIO(*)
	Flag	Core, DataIO
	Numeric register	DataNumReg
	Position register	DataPosReg
	String register	DataString
	Comment	DataString
	KAREL variable	DatarVariable
	KAREL position variable	DataVarPos
Read only	DI	Core, DataIO
	RI	Core, DataIO
	UI	Core, DataIO
	UO	Core, DataIO
	GI	Core, DataIO
	AI	Core, DataIO
	SI	Core, DataIO
	SO	Core, DataIO
	Marker	DataIO
	Current position	DataCurPos
	Program status	DataTask
	Alarm	DataAlarm

(*) Core objects are read only, DataIO objects can be read and written.

The basic flow of reading and writing robot data with robotif is shown in the following diagram.

First, create the Core object, which is used to establish a connection to the controller.

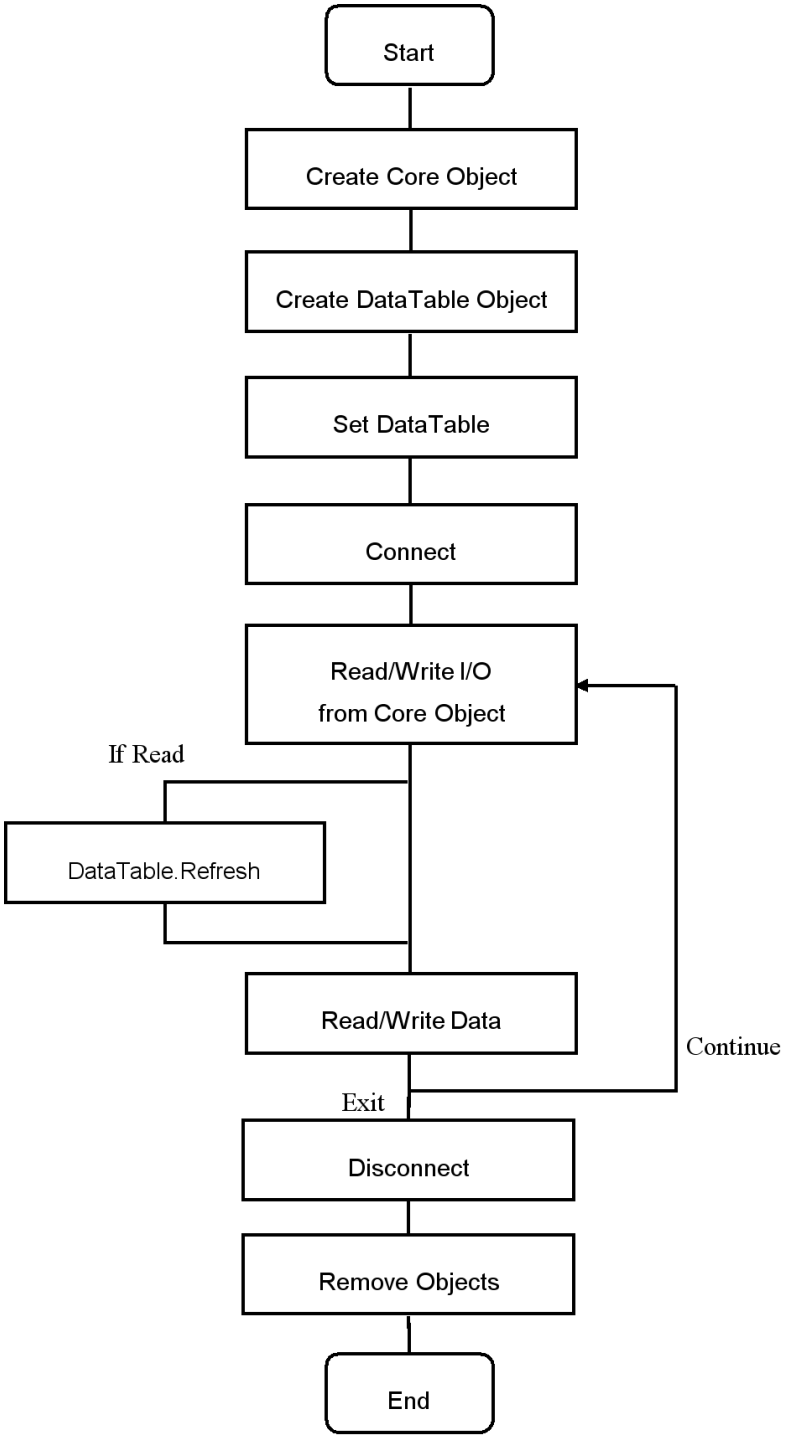
Next, create a DataTable object, which keeps the values of the robot data every time it is refreshed, from the Core object.

Using the add_[DataName] method, which will be described later, create and register the robot data objects Data[DataName] to the DataTable object.

After creating the robot data objects, connect to the controller.

It is possible to read/write various robot data with the data objects. All the data objects created by the DataTable object will be refreshed to their corresponding current values on the controller using the DataTable.refresh method.

When the desired process is finished, disconnect from the controller, and remove the Core object (removes all objects) to end.



1.2.1. Setup

Software Options

If the North America Setting (R650) is selected, the HMI Device Communication (R553) option is required to use robotif. If the Standard Setting (R651) is selected, robotif is available by default.

Controller Settings

Enable the Modbus/TCP server of the HMI Device Communication function by performing the following steps.

1. **Press MENU.**
2. **Select SETUP and then select Network Settings.**
3. **Select HMI Device settings.**
4. **Select Modbus connection.**
5. **Set the Number of connections item to 1 or more.**
6. **Cycle controller power to apply the settings.**

1.2.2. Sample Scripts

1.2.2.1. Read/Write I/O

A script sample for writing DO and reading DI using the Core object is shown below. It is possible to directly read/write DO, RO, and flags, with the Core object. Similarly, it is possible to directly read DI, RI, UI/O, SI/O, GI/O, AI/O.

References are as follows:

"II.1.2.3.5. Core.read_{I/O} Methods(P.25)"

"II.1.2.3.6. Core.write_{I/O} Methods(P.25)"

```
import fanucrobot.robotif as rbif

# Create Core object
core = rbif.Core()
#    Connect to the controller
core.connect()

# Set DO[1]~[3] to on, off, on, respectively
core.write_DO(index=1, values=[True, False, True])

# Read DI[1]~[3] to list, and print list
dilist = core.read_DI(index=1, count=3)
print(dilist) # [True, False, False]

 del core
```

A script sample for writing and reading GO using the DataIO object is shown below. This method may be more indirect than the method of directly accessing I/O using the Core object. However, this method allows access to more types of data, and the process speed is also faster due to less communication overhead.

References are as follows:

"II.1.2.4.10. DataTable.add_io Method(P.30)"

"II.1.2.13. DataIO Class(P.36)"

```
import fanucrobot.robotif as rbif

# Create Core object
core = rbif.Core()

# Create DataTable dt1
dt1 = core.get_datatable()

# Register Group Output    GO[1]~[10] to dt1
gout = dt1.add_io(datatype=rbif.IOType.GO, start_idx=1, end_idx=10)

# Connect to controller
```

```
core.connect()

# Set GO[1]~[3] to 3, 7, 15 respectively
gout.setvalues(index=1, values=[3, 7, 15])
gout.update()

# Refresh the values of registered objects in dt1
dt1.refresh()

# Read and print the values of GO[1]~[3]
gout_read = gout.getvalue(index=1, count=3)
print(gout_read) # [3, 7, 15]

del core
```

1.2.2.2. Read/Write Registers

A script sample for reading/writing Registers is shown below.
Whether register values are treated as integers or decimals is determined when the DataNumReg object is created.
References are as follows:
"II.1.2.4.3. DataTable.add_numreg Method(P.27)"
"II.1.2.6. DataNumReg Class(P.31)"

```
import fanucrobot.robotif as rbif

# Create Core object
core = rbif.Core()

# Create DataTable dt1
dt1 = core.get_datatable()

# Create DataNumReg object for R[1] to R[5] in int type
intreg = dt1.add_numreg(valuetype=int, start_idx=1, end_idx=5)
# Create DataNumReg object for R[6] to R[10] in float type
floatreg = dt1.add_numreg(valuetype=float, start_idx=6, end_idx=10)

# Connect to controller
core.connect()

# Set integer value to R[1] to R[3]
intreg.setvalues(index=1, values=[123, 456, 789])
# Set float value to R[6] to R[9]
floatreg.setvalues(index=6, values=[1.23, 45.6, 789.10])

# Refresh the values of registered objects in dt1
dt1.refresh()

# Read value of R[1]
read = intreg.getvalue(index=1)
print(read) # 123

# Read value of R[6]
read = floatreg.getvalue(index=6)
print(read) # 1.23

del core
```

1.2.2.3. Read/Write Position Registers

A script sample for reading/writing Registers is shown below.
Whether a Position Register is treated in Cartesian or Joint is determined when the DataPosReg object is created.
References are as follows:
"II.1.2.4.4. DataTable.add_posreg Method(P.27)"
"II.1.2.7. DataPosReg Class(P.31)"

```
import fanucrobot.robotif as rbif

# Create Cartesian position data to set to Position Register
cart_setpos = rbif.Cartesian()
# Initialize the position
vars(cart_setpos).update({'X':1666.666, 'Y':111.111, 'Z':1075.000,
                          'W':-179.000, 'P':1.111, 'R':2.222,
                          'Ext1':0.0, 'Ext2':0.0, 'Ext3':0.0,
                          'Flip':False, 'Left':False, 'Up':True, 'Front':True,
                          'Turn4':0, 'Turn5':0, 'Turn6':0,
                          'UFrameNumber':-1, 'UToolNumber':-1})

# Create Joint position data to set to Position Register
joint_setpos = rbif.Joint()
# Initialize the position
vars(joint_setpos).update({'J1':1.111, 'J2':2.222, 'J3':3.333,
                          'J4':4.444, 'J5':5.555, 'J6':6.666,})

# Create Core object
core = rbif.Core()

# Create DataTable dt1
dt1 = core.get_datatable()

# Create DataPosReg object for PR[1] to PR[5] in Cartesian representation
cart_posreg = dt1.add_posreg(representation=rbif.Cartesian, start_idx=1, end_idx=5)
# Create DataPosReg object for PR[1] to PR[5] in Joint representations
joint_posreg = dt1.add_posreg(representation=rbif.Joint, start_idx=1, end_idx=5)

# Connect to controller
core.connect()

# Set Cartesian position to PR[1]
cart_posreg.setvalue(index=1, position=cart_setpos)
# Set Joint position to PR[2]
joint_posreg.setvalue(index=2, position=joint_setpos)

# Refresh the values of registered objects in dt1
dt1.refresh()

# Read PR[1] in Cartesian representation
read = cart_posreg.getvalue(index=1)
print(vars(read))

# Read PR[2] in Joint representasion
read = joint_posreg.getvalue(index=2)
print(vars(read))

del core
```

1.2.3. Core Class

The Core class consists of the following methods.
From now on, the method format will be described as a Python type annotation like the one shown below.
Format: read_DI(index:int, count:int=1) -> list
In the above example, the following can be interpreted.

- The first argument "index" and the second argument "count" expect an integer (int).

(When the argument expects multiple data types, they are listed with "|".)

- The second argument “count” can be omitted, in which case 1 will be applied.
- This method returns a list.

(When “->” is not written, there is no return value.)

Method List：

- Connection Methods

connect, disconnect, set_timeout, get_timeout

- I/O Methods

read_DI, read_DO, read_RI, read_RO, read_SI, read_SO, read_UI, read_UO,
read_GI, read_GO, read_AI, read_AO, read_FLAG,
write_DO, write_RO, write_FLAG,

- DataTable Methods

get_datatable, get_datatable2, available_address

- Alarm Method

clear_alarm

1.2.3.1. Core.connect Method

Format: connect(host:str='127.0.0.1')

This method connects to the controller. Only specify the IP address of the host as an argument when using this library on a PC, or when connecting to a separate controller from the controller on which Python is to be run. If the argument is not specified, this method will connect to itself.

The various robot data registered in the DataTable created by the Core object will be assigned to Modbus Holding Registers when a connection is established. Therefore, you need to register the desired robot data to in the DataTable before connecting.

1.2.3.2. Core.disconnect Method

Format: disconnect()

This method disconnects from the controller. The DataTable object created by the Core object will be removed.

1.2.3.3. Core.set_timeout Method

Format: set_timeout(timeout:int)

This method sets a timeout value to disconnect communication with the controller. The argument is specified in msec. The default value is 10000 (10 seconds).

1.2.3.4. Core.get_timeout Method

Format: get_timeout()-> int

This method returns the timeout value (msec) specified.

1.2.3.5. Core.read_{I/O} Methods

(read_DI, read_DO, read_RI, read_RO, read_SI, read_SO, read_UI, read_UO, read_GI, read_GO, read_AI, read_AO , read_FLAG)

Format: read_{I/O}(index:int, count:int=1) -> list

These methods read all the specified I/O at once.

The “index” argument indicates the first I/O index to read. The “count” argument indicates the number of items to read.

This method returns a list of read items with length “count”. A list of bool values is returned for DI/O, RI/O, SI/O, UI/O, and FLAG. This method returns True if ON, False if OFF, for these I/O. A list of int values is returned for GI/O and AI/O.

(MARKER cannot be accessed from the Core objects. To read MARKER, create a DataIO object and use the getvalue method.)

1.2.3.6. Core.write_{I/O} Methods

（write_DO、write_RO、write_FLAG）

Format: write_{I/O}(index:int, values:list)

The “index” argument indicates the first I/O index to read.

The “values” argument indicates the I/O status represented by a list of bool values.

(GO, AO are read-only when using the Core object. To write, use a DataIO object.)

1.2.3.7. Core.get_datatable Method

Format: get_datatable() -> robotif.DataTable

This method creates the first DataTable object.

1.2.3.8. Core.get_datatable2 Method

Format: get_datatable2() -> robotif.DataTable

This method creates a second DataTable object. The functions are the same as the first DataTable object created with the get_datatable method. Data with different refresh rates can be separated into these two DataTable objects for usage.

The second DataTable is created after all the data objects in the first DataTable are created. After creating the second DataTable, it is not possible to create more data objects in the first DataTable.

1.2.3.9. Core.get_available Method

Format: get_available() -> int

The various data objects created previously in the DataTable will be assigned correspondingly to the Holding Registers when the core.connect() method is called. Refer to the "Optional Function OPERATOR'S MANUAL"(B-84464EN-2) for more details.

Holding Registers have up to 16384 addresses (32766 bytes) that they can be assigned to; any further data will not be assigned. A DataTableError exception will be raised if there are no assignable addresses.

Not to run out of Holding Registers, this method is to check the number of available addresses.

This method returns the number of assignable addresses left.

1.2.3.10. Core.clear_alarm Method

Format: clear_alarm()

This method clears the alarm log in the Alarm History screen.

(*) This is not equivalent to Alarm Reset

1.2.4. DataTable Class

The DataTable class consists of the following methods.

Method List :

- Data Register Methods

clear, add_curpos, add_numreg, add_posreg, add_variable, add_varpos, add_task, add_alarm, add_string, add_io

- Data Read Method

refresh

1.2.4.1. DataTable.clear Method

Format: clear()

This method clears the data objects registered in the DataTable object.

It is not permitted to call this clear method after connection.

1.2.4.2. DataTable.add_curpos Method

Format: add_curpos(representation:type, uf:int=-1, group:int=1) -> robotif.DataCurPos

This method creates a DataCurPos(P.30) object from DataTable.

A DataCurPos object reads the current position with the user frame specified by the “uf” argument.

The “representation” argument indicates the position representation, in which Joint or Cartesian can be specified.

The “uf” argument indicates the user frame number used to read the current position. If 1-64 is specified in “uf”, the specified user frame number will be used regardless of the selected user frame number on the controller. If robotif.ACTIVE_FRAME(=-1) is specified in “uf”, the user frame number currently selected on the controller will be used.

The “group” argument indicates the group number used to read current position. If the current position to be read is not from the robot in Group 1 of a multi-motion group system, specify the group number.

1.2.4.3. DataTable.add_numreg Method

Format: add_numreg(valuetype:type, start_idx:int, end_idx:int) -> robotif.DataNumReg

This method creates a DataNumReg(P.31) object from DataTable. A DataNumReg object reads/writes numeric register values.

The “valuetype” argument indicates the data type to be used, in which int or float can be specified.

If int is specified, the valid range of values follows that of signed 32-bit integer (-2,147,483,648 - 2,147,483,647). If float is specified, the valid range of values follows that of IEEE754 single-precision (32bits) floating-point format, but only up to 6 significant figures.

The “start_idx” argument is the first index, “end_idx” argument is the final index, of the numeric registers used for read/write.

1.2.4.4. DataTable.add_posreg Method

Format: add_posreg(representation:type, start_idx:int=-1, end_idx:int=1, group:int=1) -> robotif.DataPosReg

This method creates a DataPosReg(P.31) object from DataTable. A DataPosReg object reads/writes position register values.

The “representation” argument indicates the position representation of the POSITION type data, in which Joint or Cartesian can be specified.

The “start_idx” argument is the first index, “end_idx” argument is the final index, of the numeric registers used for read/write.

The “group” argument indicates the group number used to read/write the current position. If the current position is not from the robot in Group 1 of a multi-motion group system, specify the group number.

1.2.4.5. DataTable.add_variable Method

Format: add_variable(vartype:type, varname:str, program:str=None, stringsize:int=40, start_idx:int=None, end_idx:int=None) -> robotif.DataVariable

This method creates a DataVariable(P.32) object from DataTable. A DataVariable object reads/writes KAREL variables or Data IDs.

The “vartype” argument indicates the data type of the KAREL variable or Data ID to be used in Python. If the data type is BOOLEAN, specify bool. If the data type is INTEGER, SHORT, BYTE, specify int. If the data type is REAL, specify float. If the data type is STRING, specify str. If the data type of the KAREL variable or Data ID is POSITION, use the add_varpos method described next.

If str is specified in “vartype”, specify the address size for the string type variable in “stringsize”. 1 address is equivalent to 2 bytes. For example, if “stringsize=30” is specified, a UTF-8 string of up to 60 bytes can be used.

The “varname” argument indicates the name of KAREL variable or Data ID (include the $ at the start of the name).

The “program” argument indicates the name of the KAREL program (without extension) which contains the KAREL variable. When registering Data ID, do not specify this argument.

When registering an array, specify the variable name (without index) in “varname”, the first index for read/write in “start_idx”, and the final index in “end_idx”.

When only registering a certain element of an array, specify the variable name including index in "varname".

When registering a multidimensional array, it is possible to only register the inner-most one-dimensional array. In this case, omit the rightmost index when specifying "varname", then specify the range of array to register in "start_idx" and "end_idx". (This specification was added in V10.10P/12.)

For example, if 3dimarray is a three-dimensional array, to register 3dimarray[2,3,1]-[2,3,10], write the code as follows.

add_variable(vartype=int, varname='3dimarray[2,3]', program='sample', start_idx=1, end_idx=10)

1.2.4.6. DataTable.add_varpos Method

Format: add_varpos(representation:type, varname:str, program:str=None, start_idx:int=None, end_idx:int=None) -> robotif.DataVarPos

This method creates a DataVarPos(P.33) object from DataTable. A DataVarPos object reads/writes POSITION type KAREL variable or Data ID.

The “representation” argument indicates the position representation of the POSITION type data, in which Joint or Cartesian can be specified.

The “varname” argument indicates the name of KAREL variable or Data ID (include the $ at the start of the name).

The “program” argument indicates the name of the KAREL program (without extension) which contains the KAREL variable. When registering Data ID, this argument is not used, specify the Data ID name in “varname”.

When registering an array, specify the variable name (without index) in “varname”, the first index for read/write in “start_idx”, and the final index in “end_idx”.

When only registering a certain element of an array, specify the variable name including index in "varname".

When registering a multidimensional array, it is possible to only register the inner-most one-dimensional array. In this case, omit the rightmost index when specifying "varname", then specify the range of array to register in "start_idx" and "end_idx". (This specification was added in V10.10P/12)

For example, if 3dimarray is a three-dimensional array, to register 3dimarray[2,3,1]-[2,3,10], write the code as follows.

add_varpos(representation=robotif.Cartesian, varname='3dimpos[2,3]', program='sample', start_idx=1, end_idx=10)

1.2.4.7. DataTable.add_task Method

Format: add_task(datatype:int=0, index:int=1) -> robotif.DataTask

This method creates a DataTask(P.33) object from DataTable. A DataTask object reads program execution status.

The “datatype” argument indicates the type of task to be used, and must be selected from the table below.

The “index” argument indicates the task number in a multiple task system. When two tasks run simultaneously, task number 1 is to access the first one, task number 2 is to access the second one. The task number depends on when the program is started, and the timing of the communication, but once a task number is given to a task, the task keeps the task number until the program is aborted.

Enum name TaskType.~	Meaning	Value
TASK	Return the executing program name and line number.	0

Enum name TaskType.~	Meaning	Value
TASK_IGNORE_MACRO	If the executing program is a macro program, return the caller program name and line number instead of the macro program. If all caller programs are macro, the program name is “”(empty string) and line number is zero.	1
TASK_IGNORE_KAREL	If the executing program is a KAREL program, return the caller program name and line number instead of the KAREL program. If all caller programs are KAREL, the program name is “”(empty string) and line number is zero.	2
TASK_IGNORE_MACRO_KAREL	If the executing program is a macro or KAREL program, return the caller program name and line number instead of the macro or KAREL program. If all caller programs are macro or KAREL, the program name is “”(empty string) and the line number is zero.	3

1.2.4.8. DataTable.add_alarm Method

Format: add_alarm(datatype:int, count:int=1) -> robotif.DataAlarm

This method creates a DataAlarm(P.34) object from DataTable. A DataAlarm object reads active alarms and alarm items from the Alarm History screen.

The “datatype” argument indicates the type of alarms to register, and must be selected from the table below.

Refer to the following table for the meaning of the “count” argument.

Enum name AlarmType.~	Meaning	Value
HISTORY	The “count” argument indicates which alarm item to read counting from the top of the Alarm History screen.	0
ACTIVE	The “count” argument indicates which active alarm item to read counting from the most recent alarm.	1

1.2.4.9. DataTable.add_string Method

Format: add_string(datatype:int, start_idx:int, end_idx:int) -> robotif.DataString

This method creates a DataString(P.35) object from DataTable. A DataString object accesses string data like string registers and I/O comments.

The “start_idx” argument is the first index, “end_idx” argument is the final index, of the numeric registers used for read/write.

The “datatype” argument indicates the type of string data to register, and must be selected from the table below.

Enum name StringType.~	Type	Value
STRREG	String register	0
STRREG_COMMENT	String register comment	1
NUMREG_COMMENT	Numeric register comment	2
POSREG_COMMENT	Position register comment	3
DI_COMMENT	DI comment	4
DO_COMMENT	DO comment	5
RI_COMMENT	RI comment	6
RO_COMMENT	RO comment	7
UI_COMMENT	UI comment	8
UO_COMMENT	UO comment	9,
SI_COMMENT	SI comment	10
SO_COMMENT	SO comment	11
GI_COMMENT	GI comment	15

Enum name StringType.~	Type	Value
GO_COMMENT	GO comment	16
AI_COMMENT	AI comment	17
AO_COMMENT	AO comment	18
FLAG_COMMENT	Flag comment	19
MARKER_COMMENT	Marker comment	20

1.2.4.10. DataTable.add_io Method

Format: add_io(datatype:int, start_idx:int, end_idx:int) -> robotif.DataIO

This method creates a DataIO(P.36) object from DataTable. A DataIO object accesses various I/O data, flags, and markers.

The “start_idx” argument is the first index, “end_idx” argument is the final index, of the numeric registers used for read/write.

The “datatype” argument indicates the type of I/O to register, and must be selected from the table below.

This method has higher process speed than the Core object method when using various types of I/O, due to less communication overhead.

Enum name IOType.~	Meaning	Value
DI	Digital Input	0
DO	Digital Output	1
RI	Robot Input	2
RO	Robot Output	3
UI	Peripheral (UOP) Input	4
UO	Peripheral (UOP) Output	5
SI	Standard Operator Panel (SOP) Input	6
SO	Standard Operator Panel (SOP) Output	7
GI	Group Input	12
GO	Group Output	13
AI	Analog Input	14
AO	Analog Output	15
FLAG	Flag	16
MARKER	Marker	17

1.2.4.11. DataTable.refresh Method

Format: refresh()

When a refresh method is called, all data derived from the DataTable object will be refreshed to their current values. Read data will be kept until the next refresh. (Note that data will not change if a refresh method is not called.)

1.2.5. DataCurPos Class

Objects of DataCurPos class are created by DataTable.add_curpos method(P.27). The DataCurPos class consists of the following method.

Method List :

getvalue

1.2.5.1. DataCurPos.getvalue Method

Format: getvalue() -> Cartesian | Joint

This method reads the current position with Cartesian or Joint representation.

If the read is successful, this method returns Cartesian or Joint objects. The object type returned is specified when calling an add_curpos method.

If the read fails, or if the current position is unable to be represented as Cartesian or Joint object, this method returns None.

If read with Cartesian representation, UFrameNumber will be the frame number specified when calling the add_curpos method. UToolNumber will constantly be -1.

1.2.6. DataNumReg Class

Objects of DataNumReg class are created by DataTable.add_numreg method(P.27). The DataNumReg class consists of the following methods.

Method List :

setvalue, getvalue

1.2.6.1. DataNumReg.setvalues Method

Format: setvalues(index:int, values:list)

This method sets a series of numeric register values at one time.

The “index” argument indicates the index of the target numeric register.

The “values” argument indicates an array of set values, whose data type (int or float) is specified when calling the add_numreg method.

1.2.6.2. DataNumReg.getvalue Method

Format: getvalue(index:int) -> int | float

This method reads numeric register values.

The “index” argument indicates the index of the target numeric register. This method returns int or float type values, which are specified when calling the add_numreg method.

1.2.7. DataPosReg Class

Objects of DataPosReg class are created by DataTable.add_posreg method(P.27). The DataPosReg class consists of the following methods.

Method List :

setvalue, getvalue

1.2.7.1. DataPosReg.setvalue Method

Format: setvalue(index:int, position:Cartesian | Joint)

This method writes position data into a position register.

The “index” argument indicates the index of the target numeric register.

The “position” argument indicates the position value to be set to the target position register. The representation type of the position value is specified when calling the add_posreg method. If a different representation type is specified, an exception will be raised.

(*) UF, UT of position register cannot be overwritten with this method.

1.2.7.2. DataPosReg.getvalue Method

Format: getvalue(index:int) -> Cartesian | Joint

This method reads position register values.

The "index" argument indicates the index of the position register to be read.

If the read is successful, this method returns Cartesian or Joint objects. The object type returned is specified when calling the add_varpos method

If the read fails, or if the current position is unable to be represented as Cartesian or Joint object, this method returns None.

1.2.8. DataVariableClass

Objects of DataVariable class are created by DataTable.add_variable method(P.27). The DataVariable class consists of the following methods.

Method List :

setvalue, getvalue

1.2.8.1. DataVariable.setvalue Method

Format: setvalue(value:int | bool | float | str, index:int=None)

This method sets the value of a KAREL variable or Data ID to that of a “value” argument.

The type of “value” must be the same as specified when calling the add_variable method.

If DataVariable is registered as an array, specify the index to be set in the “index” argument.

1.2.8.2. DataVariable.getvalue Method

Format: setvalue(index:int=None) -> int | bool | float | str

This method reads a KAREL variable or Data ID.

If DataVariable is registered as an array, specify the index to be read in the “index” argument.

(*) UnicodeDecodeError may occur if the “stringsize” specified when calling the add_variable method is not large enough for the string variable.

1.2.9. DataVarPos Class

Objects of DataVarPos class are created by DataTable.add_varpos method(P.28). The DataVarPos class consists of the following methods.

Method List：

setvalue, getvalue

1.2.9.1. DataVarPos.setvalue Method

Format: setvalue(position: Cartesian | Joint, index:int=None)

This method sets the position data to KAREL POSITION type variable or Data ID.

The “position” argument indicates the position value to be set to the target position register. The representation type of the position value is specified when calling the add_posreg method. If a different representation type is specified, an exception will be raised.

If DataVarPos is registered as an array, specify the index to be set in the “index” argument.

1.2.9.2. DataVarPos.getvalue Method

Format: getvalue(index:int=None) -> Cartesian | Joint

This method reads a POSITION type KAREL variable or Data ID.

If the read is successful, this method returns Cartesian or Joint objects. The object type returned is specified when calling the add_varpos method

If DataVarPos is registered as an array, specify the index to be read in the “index” argument.

1.2.10. DataTask Class

Objects of DataTask class are created by DataTable.add_task method(P.28). The DataTask class consists of the following method.

Method List：

getvalue

1.2.10.1. DataTask.getvalue Method

Format: getvalue() -> dict

This method returns program information as a dictionary. The keys and their meanings are shown in the table below.

(*) The currently running program name and line number may differ based on the “datatype” specified when calling the add_task method.

Key	Data Type	Meaning
‘Program’	str	Returns the currently running program name.
‘Line’	int	Returns the currently running line number
‘State’	int	Returns the execution status, as shown below.

Key	Data Type	Meaning
		0: Aborted 1: Paused 2: Running
'Parent'	str	Returns the first caller program name. If no subprogram is called, this value will be the same as that of 'Program'

1.2.11. DataAlarm Class

Objects of DataAlarm class are created by DataTable.add_alarm method(P.29). The DataAlarm class consists of the following method.

Method List :

getvalue

1.2.11.1. DataAlarm.getvalue Method

Format: getvalue() -> dict

This method reads the active alarms or information in the alarm history.
This method returns alarm information as a dictionary, as shown in the following table.

Key	Data Type	Meaning
'AlarmID'	int	Returns the facility code. For example, in 'SRVO-001', AlarmID returns 11 which is the facility code that represents the 'SRVO' facility name. Refer to the Alarm Code List OPERATOR'S MANUAL for more information. If there is no active alarm, AlarmNumber returns 0 for the active alarm reference.
'AlarmNumber'	int	Returns the error code number. For example, in 'SRVO-001', AlarmNumber returns 1. If there is no active alarm, AlarmNumber returns 0 for the active alarm reference.
'CauseAlarmID'	int	Returns the cause alarm ID. Normally when an alarm is raised, it will be displayed at the top line of the TP screen. Some alarms have two alarm messages and will also be displayed on the second line of the TP screen. This second line of the alarm is the cause code. The value means cause code alarm ID. If there is no cause code, CauseAlarmID returns 0.
'CauseAlarmNumber'	int	Returns the cause code alarm number. If there is no cause code, CauseAlarmID returns 0.
'Severity'	int	Returns the alarm severity. The correspondence of alarm severity and the 'Severity' key value is shown in the following table.
'Date'	datetime	Returns the date and time of the alarm as a datetime.datetime object.
'AlarmMessage'	str	Returns the alarm message. This message is the message displayed on the top line of TP (including error codes like 'SRVO-001') when an alarm is raised.
'CauseAlarmMessage'	str	Returns the cause code of the alarm message, if available.
'SeverityMessage'	str	Returns the alarm severity string, like 'WARN'.

The values and meanings of 'Severity' key in the table above is following.

Alarm Severity	Value
NONE	128
WARN	0
PAUSE.L	2

Alarm Severity	Value
PAUSE.G	34
STOP.L	6
STOP.G	38
SERVO	54
ABORT.L	11
ABORT.G	43
SERVO2	58
SYSTEM	123

1.2.12. DataStringClass

Objects of DataString class are created by DataTable.add_string method(P.29). The DataString class consists of the following methods.

Method List :

setvalue, getvalue, write_buffer, update, reset

1.2.12.1. DataString.setvalue Method

Format: setvalue(index:int, value:str)

This method reads string data like string registers and comments.

The “index” argument indicates the index to set the string data like string registers and comments.

The “value” argument indicates the string data to write.

The data is not immediately transferred when calling this method. The data is only stored in a buffer. This method can be called multiple times to store a series of string data. When the desired string data has been stored, call the update method to transfer the data to the robot.

(*) The valid size of string data for “value” is based on R-50iA, which uses UTF-8. The valid size is 253 bytes for string registers, and 95 bytes for comments. String data cannot be set if the size exceeds the valid size.

1.2.12.2. DataString.getvalue Method

Format: getvalue(index:int) -> str

This method reads string data like string registers and comments.

The “index” argument indicates the index of the string data to read.

1.2.12.3. DataString.update Method

Format: update()

This method transfers the data stored in a buffer by the setvalue method to the robot.

The data stored by the setvalue method from the minimum index to the maximum index are sent together to the robot. (The values of string data that are not stored in the buffer by the setvalue method are empty string.)

The buffer is cleared after the data is transferred.

1.2.12.4. DataString.reset Method

Format: reset()

This method clears the buffer to set string data. Previously called setvalue will be cancelled.

1.2.13. DataIO Class

Objects of DataIO class are created by DataTable.add_io method(P.30). The DataIO class consists of the following methods.

Method List :

setvalues, getvalue, update, resest

1.2.13.1. DataIO.setvalues Method

Format: setvalues(index:int, values:list)

This method sets the values of various I/O.

Specify the starting index in "index".

If the I/O to be registered to DataIO is GI/O or AI/O, specify a list of int values in "values". Otherwise, specify a list of bool values.

The data is not immediately transferred when calling this method. The data is only stored in a buffer. When transferring the data to the robot, call the update method.

1.2.13.2. DataIO.getvalue Method

Format: getvalue (index:int, count:int=1) -> list

This method reads the values of various I/O.

Specify the starting index of referred I/O in "index", and the number of reference points in "count".

If GI/O, AI/O are registered to DataIO when calling the add_io method, this method returns int type values. For other I/O, bool is returned.

1.2.13.3. DataIO.update Method

Format: update()

This method transfers the data stored in a buffer by the setvalue method to the robot.

The data stored by the setvalue method from the minimum index to the maximum index are sent together to the robot. The I/O values that are not set in the buffer by the setvalue method will be OFF or 0.

The buffer is cleared after the data is transferred.

1.2.13.4. DataIO.reset Method

Format: reset()

This method clears the buffer to set string data. Previously called setvalue will be cancelled.

1.3. rmi package

The fanucrobot.rmi is a library for making Teach Pendant (TP) programs, and is implemented as a wrapper library for RMI (Remote Motion Interface, R912).

Keep in mind the following basic specifications for RMI when using this library.

Refer to Remote Motion Interface OPERATOR'S MANUAL (B-84184EN) for details.

- You must be outside the safety fence and ready to apply an ESTOP when using RMI.
- Communication between remote devices and robot controller is established by exchanging JSON string packets through socket communication. When running Python on controller, the controller itself is considered a remote device.
- Remote motion is done by appending instructions which are delivered through socket communication, to a temporary TP program called RMI_MOVE.
- RMI consists of a "Communication packet", "Robot Command packet", and "TP Program Instruction packet".
- The "Communication packet" is to connect to, or disconnect from the robot controller, which has to be sent when starting or stopping the use of RMI respectively.
- The "Robot Command packet" is to refer or change the current robot status. This command will be executed and the result will be returned immediately after sending this packet.
- The "TP Program Instruction packet" appends TP instructions to RMI_MOVE. The results of this packet are not returned immediately, and will only be returned after the appended TP instruction is executed. Up to eight pending TP instructions can be stored, and any further instructions will not be appended. Hence, until the first of eight pending TP instructions is executed, delivery of the next (ninth) instruction will be put on hold.

1.3.1. Setup

To use RMI, set the robot controller as follows:

1. The teach pendant is disabled, and the controller is in AUTO mode.
2. The controller is ready, i.e., there are no servo or other alarms.
3. The selected program is not RMI_MOVE.
4. Use Cold Start (SYSTEM > CONFIG > HOT/COLD Start > Use HOT START > FALSE)

1.3.2. Sample Scripts

1.3.2.1. Simple Single Group Motion Instructions

The following example is a sample script of a simple pick and place operation with linear motion.

Note that the write_do method is a kind of command packet, so when executed in Python, the method will be efficient immediately, regardless of the execution status of the motion instruction. In this example, the wait_progress method is used to delay the execution of the write_do command until after the motion instructions are completed.

```
import copy
from fanucrobot.rmi import RMI, Joint, Cartesian

# create RMI object
rmi = RMI()
# connect to controller
rmi.connect()

# create Cartesian position data
pos1 = Cartesian()
```

```
# Set pos1 to use as Start point and Place point
vars(pos1).update({'X':1650.000, 'Y':0.000, 'Z':1075.000,
                   'W':-180.000, 'P':0.000, 'R':0.000,l
                   'Ext1':0.0, 'Ext2':0.0, 'Ext3':0.0,
                   'Flip':False, 'Left':False, 'Up':True, 'Front':True,
                   'Turn4':0, 'Turn5':0, 'Turn6':0,
                   'UFrameNumber':0, 'UToolNumber':1})

# Set pos2 as a escape position from pickup point, that is X, Y away from pos1
pos2 = copy.copy(pos1)
vars(pos2).update({'X':1900.000, 'Y':300.000})

# Set pos3 as the pickup point, that is Z away from pos2
pos3 = copy.copy(pos2)
vars(pos3).update({'Z':700.000})

# Start execution of RMI_MOVE
rmi.init_move()

# Open hand ( DO[1]=ON : close, DO[1]=OFF    : open )
rmi.write_do(index=1, onoff=False)

# Linear motion instruction
rmi.linear(pos=pos1, speedtype='mmsec', speed=500, termtype='CNT', termvalue=80)
rmi.linear(pos=pos2, speedtype='mmsec', speed=300, termtype='CNT', termvalue=30)

rmi.linear(pos=pos3, speedtype='mmsec', speed=300, termtype='FINE')
# Wait until move to pos3 (pickup point)
rmi.wait_progress()
# Pickup by closing hand, DO[1]=ON
rmi.write_do(index=1, onoff=True)

rmi.linear(pos=pos2, speedtype='mmsec', speed=300, termtype='CNT', termvalue=30)
rmi.linear(pos=pos1, speedtype='mmsec', speed=500, termtype='CNT', termvalue=80,
           option={'NoBlend':True}) # NoBlend : execute this motion without waiting for next motion instruction

# End remote motion program when motion is completed
rmi.end_move()

# Open hand to place
rmi.write_do(index=1, onoff=False)

# Disconnect
rmi.disconnect()
```

The above sample will produce a motion similar to the following TP program. Here, P[1], P[2], P[3] are equivalent to pos1, pos2, pos3 respectively.

```
1: DO[1]=OFF
2: L P[1] 500mm/sec CNT80
3: L P[2] 300mm/sec CNT30
4: L P[3] 300mm/sec FINE
5: DO[1]=ON
6: L P[2] 300mm/sec CNT30
7: L P[1] 500mm/sec CNT80
8: DO[1]=OFF
```

1.3.2.2. Incremental Motion Instruction

The following sample script rotates the J6 axis by +30 degrees from its current position.

In this example, the curpos_joint method is used to get the current position, then 30 degrees is added to the J6 axis component of the current position to move the robot.

```
from fanucrobot.rmi import RMI, Joint, Cartesian

# create RMI object
rmi = RMI()

# connect to controller
rmi.connect()

# Set override to 50
rmi.set_override(override=50)

# Start execution of RMI_MOVE
rmi.init_move()

# Get the current Joint position values
jpos = rmi.curpos_joint()

if jpos != None:
    # Set J6 axis component by adding 30 degrees
    jpos.J6 += 30.0
    # Move robot to set position with joint motion and FINE termination type
    rmi.joint(pos=jpos, speedtype='percent', speed=100, termtype='FINE')

  # End remote motion program when motion is completed
    rmi.end_move()

# Disconnect
rmi.disconnect()
```

The following sample script also rotates the J6 axis by +30 degrees from its current position.

In this example, the joint_incremental method is used to append an incremental instruction, which moves the robot relative to its current position, based on the relative angular displacement specified.

```
from fanucrobot.rmi import RMI, Joint

rmi = RMI()
# connect to controller
rmi.connect()

# Set override to 50
rmi.set_override(override=50)

# Start execution of RMI_MOVE
rmi.init_move()

# Create Joint position data
jpos = Joint()

# Set angular displacement of J6 axis to +30 degrees (default value of other axes=0)
jpos.J6 = 30.0

# Move robot in Joint motion based on angular displacement set
rmi.joint(pos=jpos, speedtype='percent', speed=100, termtype='FINE')

# End remote motion program when motion is completed
rmi.end_move()

# Disconnect
```

```
rmi.disconnect()
```

1.3.2.3. Motion Instruction for Multi-Group System

Following is a sample script to move groups 1 and 3 simultaneously in a multi-group system. Here, groups 1, 2 are robots, and group 3 is a single-axis positioner.

```
from fanucrobot.rmi import RMI, Joint, Cartesian

rmi = RMI()
# connect to controller
rmi.connect()

# Set override to 50
rmi.set_override(override=50)

# Position data list of group 1
posg1 = [Cartesian(), Cartesian()]
# Position data list of group 3
posg3 = [Joint(), Joint()]

# create Cartesian position data
# Set posg1
vars(posg1[0]).update({'X':1650.000, 'Y':0.000, 'Z':1075.000,
                       'W':-180.000, 'P':0.000, 'R':0.000,
                       'Ext1':0.0, 'Ext2':0.0, 'Ext3':0.0,
                       'Flip':False, 'Left':False, 'Up':True, 'Front':True,
                       'Turn4':0, 'Turn5':0, 'Turn6':0,
                       'UFrameNumber':0, 'UToolNumber':1})

vars(posg1[1]).update({'X':1800.000, 'Y':200.000, 'Z':700.000,
                       'W':-180.000, 'P':0.000, 'R':0.000,
                       'Ext1':0.0, 'Ext2':0.0, 'Ext3':0.0,
                       'Flip':False, 'Left':False, 'Up':True, 'Front':True,
                       'Turn4':0, 'Turn5':0, 'Turn6':0,
                       'UFrameNumber':0, 'UToolNumber':1})

# Set posg3
posg3[0].J1 = 0.00
posg3[1].J1 = 180.00

# Start execution of RMI_MOVE. Specify group 1, 3 as motion group
rmi.init_move(groupmask=0b101)

# Move robot in Linear motion
rmi.linear(pos={'G1':posg1[0], 'G3':posg3[0]}, speedtype='mmsec', speed=300, termtype='FINE')
rmi.linear(pos={'G1':posg1[1], 'G3':posg3[1]}, speedtype='mmsec', speed=300, termtype='FINE')

# End remote motion program when motion is completed
rmi.end_move()

# Disconnect
rmi.disconnect()
```

1.3.3. RMIClass : Communication Methods and Constructor

The robotif.rmi.RMI class consists of various methods, so the methods will be explained based on their types. The communication methods explained in this subsection send a “Communication packet” to the controller to connect and disconnect communication with the controller.

Method List :

RMI, connect, disconnect

1.3.3.1. Constructor

Format: RMI(host:str='127.0.0.1', timeout:int=10)

The method is a constructor of an RMI object.

The “host” argument indicates the IP address of the target robot when executing Python on a remote PC. If Python is running on R-50iA, this argument can be omitted.

The “timeout” argument indicates the timeout value of socket communication between the controller and remote PC. This value is set to 10 sec by default.

1.3.3.2. RMI. connect Method

Format: connect()

This method does a handshake with the target robot and establishes a socket communication for RMI.

This method needs to be called before all other RMI methods. To start execution of the remote motion program RMI_MOVE, init_move method needs to be called after this method is done.

1.3.3.3. RMI.disconnect Method

Format: disconnect()

This method disconnects the socket communication with the controller, and ends the RMI session.

1.3.4. RMIClass: Command Method

The communication methods in this subsection sends a “Robot Command packet” to the controller, and will return a value upon receiving the return packet. If ErrorID in the return packet is not zero, an exception will be raised within the method.

Method List :

init_move, abort, pause, resume, read_alarm, get_frame_number, set_frame_number, get_status, read_uframe, write_uframe, read_utool, write_utool, read_di, write_do, curpos_cartesian, curpos_joint, set_override, read_position_register, write_position_register, reset_alarm, get_tcp_speed

1.3.4.1. RMI.init_move Method

Format: init_move(groupmask:int=1)

This method initializes remote motion. The controller becomes ready to execute RMI_MOVE and receive TP instructions.

RMI can control up to two groups at a time in multi-group systems.

The “groupmask” argument indicates an unsigned 1 byte bit-field where each bit corresponds to the group number in a multi-group system. For example, in a system with three groups, groupmask = 6 = 0b00000110 indicates that the motion group will be groups 2 and 3.

This method must be called before all other instruction methods.

1.3.4.2. RMI.abort Method

Format: abort()

Aborts the execution of RMI_MOVE. Even if the last appended TP instruction has been executed, execution of RMI_MOVE will not be aborted. This method or the disconnect method must be called at the end of an RMI session. Otherwise, other TP programs cannot be executed after the RMI session.

1.3.4.3. RMI.pause Method

Format: pause()

This method pauses the execution of RMI_MOVE.

1.3.4.4. RMI.resume Method

Format: resume()

This method resumes the execution of RMI_MOVE.

1.3.4.5. RMI.read_alarm Method

Format: read_alarm() -> list

This method gets up to five alarm codes of active alarms in the robot controller.

This method returns the alarm code strings as a list. For example, if alarms “SYST-045 TP ON in AUTO mode” and “SYST-212 Need to apply to DCS param” are currently active on the controller, the returned value will be [’SYST-212’,’SYST-045’].

1.3.4.6. RMI.get_frame_number Method

Format: get_frame_number(group:int=1) -> dict

This method gets the currently selected user frame number and tool frame number.

The “group” argument indicates the group number used to get frame numbers. If the user/tool frame is not from Group 1 of a multi-motion group system, specify the group number.

This method returns the frame numbers as a dictionary. For example, if user frame number 1 and tool frame number 2 are selected, the return value will be as follows:

{

'UFrameNumber' : 1,

'UToolNumber' : 2

}

1.3.4.7. RMI.set_frame_number Method

Format: set_frame_number(ufnum:int, utnum:int, group:int=1)

This method sets the current user frame number and tool frame number.

The “ufnum” and “utnum” arguments indicate the user frame number and tool frame number to be set respectively.

The “group” argument indicates the group number used to set frame numbers. If the user/tool frame is not from Group 1 of a multi-motion group system, specify the group number.

⚠CAUTION

Do not call this method while the robot is in motion, because changing the frame number and tool number may cause a frame mismatch error and stop the TP program.

1.3.4.8. RMI.get_status Method

Format: get_status() -> dict

This method gets the current status of the controller.

This method returns the status as a dictionary, where the keys and values are as shown below.

Key (str)	Meaning of value (int)	Example of value
‘ServoReady’	If 1, the controller is ready for motion. Otherwise, the controller is in an error state.	2
‘TPMode’	If 0, the teach pendant is disabled.	1
‘RMIMotionStatus’	If 1, RMI_MOVE is being executed.	0
‘ProgramStatus’	Shows the current status of RMI_MOVE. If 1, RMI_MOVE is aborted.	0
‘SingleStepMode’	If 1, the controller is in single-step mode.	0
‘NumberUTool’	Shows the number of tool frames available in the robot controller.	10
‘NumberUFrame’	Shows the number of user frames available in the robot controller.	9
‘NextSequenceID’	Shows the next valid sequence ID. (*) Sequence ID refers to the serial number of the instructions appended to RMI_MOVE. Sequence IDs are assigned to each TP instruction packet within the library. Sequence IDs are used as each instruction’s ID, and can be used to check for missing packets in the controller.	1
‘Override’	Shows the override value set in the controller. If the override value is set to FINE/VFINE/0%, this value will be 1.	90

1.3.4.9. RMI.read_uframe Method

Format: read_uframe(ufnum:int, group:int=1) -> Cartesian

This method reads the content of the specified user frame.

The “ufnum” argument indicates the user frame number whose data will be read.

The “group” argument indicates the group number used to read user frame data. If the user frame is not from Group 1 of a multi-motion group system, specify the group number.

This method only returns the position and angle components of the user frame data. The returned Cartesian value (UToolNumber, UFrameNumber, Front, Up, Left, Flip, Turn4~6) is always the initial value, so do not use these values before setting them.

1.3.4.10. RMI.write_uframe Method

Format: write_uframe(ufnum:int, pos:Cartesian, group:int=1)

This method sets the user frame data.

The “ufnum” argument indicates the user frame number whose data will be set to that of “pos”.

The “pos” argument uses only position and angle components (X, Y, Z, W, P, R), so the frame and configuration components will be ignored.

The “group” argument indicates the group number used to write user frame data. If the user frame is not from Group 1 of a multi-motion group system, specify the group number.

1.3.4.11. RMI.read_utool Method

Format: read_utool(utnum:int, group:int=1) -> Cartesian

This method reads the content of the specified tool frame.

The “utnum” argument indicates the user frame number whose data will be read.

The “group” argument indicates the group number used to write user frame data. If the user frame is not from Group 1 of a multi-motion group system, specify the group number.

1.3.4.12. RMI.write_utool Method

Format: write_utool(utnum:int, pos:Cartesian, group:int=1)

This method sets the user frame data.

The “utnum” argument indicates the tool frame number whose data will be set to that of “pos”.

The “pos” argument uses only position and angle components (X, Y, Z, W, P, R), so the frame and configuration components will be ignored.

The “group” argument indicates the group number used to write user frame data. If the user frame is not from Group 1 of a multi-motion group system, specify the group number.

1.3.4.13. RMI.read_di Method

Format: read_di(index:int) -> bool

This method reads the status of the digital input port, DI[index]. If the specified DI is ON, then True is returned.

1.3.4.14. RMI.write_do Method

Format: write_do(index:int, onoff:bool)

This method sets the digital output port, DO[index] to ON/OFF. The "onoff" argument should be set to True if the DO is ON, and False if the DO is OFF.

1.3.4.15. RMI.curpos_cartesian Method

Format: curpos_cartesian(group:int=1) -> Cartesian

This method reads the current Cartesian position of robot.

The “group” argument indicates the group number used to read the robot position. If the current position to be read is not from Group 1 of a multi-motion group system, specify the group number.

CAUTION

If machine lock is ON, or if you are using ROBOGUIDE, the refresh rate of the position values read by the curpos_cartesian method is about 100 msec. For example, when calling this method right after a motion instruction with “FINE” termination type is executed, a position prior to reaching the target position of the motion instruction may be read.

1.3.4.16. RMI.curpos_joint Method

Format: curpos_joint(group:int=1) -> Joint

This method reads the current Joint position of robot.

The “group” argument indicates the group number used to read the robot position. If the current position to be read is not from Group 1 of a multi-motion group system, specify the group number.

CAUTION

If machine lock is ON, or if you are using ROBOGUIDE, the refresh rate of the joint angles read by the curpos_joint method is about 100 msec. For example, when calling this method right after a motion instruction with “FINE” termination type is executed, a position prior to reaching the target position of the motion instruction may be read.

1.3.4.17. RMI.set_override Method

Format: set_override(override:int)

This method sets the override value of the controller.

The range of override value that can be set is 1-100.

1.3.4.18. RMI.read_position_register Method

Format: read_position_register(index:int, group:int=1) -> Cartesian

This method reads the contents of a position register.

The "index" argument indicates the index of the position register to be read.

The "group" argument indicates the group number used to read the position register. If the position register to be read is not from Group 1 of a multi-motion group system, specify the group number.

1.3.4.19. RMI.write_position_register Method

Format: write_position_register(index:int, pos:Cartesian, group:int=1)

This method writes position data into a position register.

The "index" argument indicates the index of the position register which the Cartesian position data specified in the "pos" argument will be written into.

The "group" argument indicates the group number used to write the position register.

If the position register to be written is not from Group 1 of a multi-motion group system, specify the group number.

1.3.4.20. RMI.reset_alarm Method

Format: reset_alarm()

This method resets alarms in the controller.

When the controller posts an alarm after a connect method is called to connect to the controller, and an alarm post notification packet is received, the RMI connection will be disconnected within the RMI class and Python will be aborted unconditionally.

For this reason, use this method to reset the alarms remaining in the controller, after calling a connect method and before calling an init_move method.

1.3.4.21. RMI.get_tcp_speed Method

Format: get_tcp_speed() -> dict

This method gets the current speed of the tool center point (TCP).

This method can only be called after calling an init_move method.

This method returns the status as a dictionary, where the keys and their meanings are as shown below.

{

'TimeTag' : 475164.

'Speed' : 705.528

}

1.3.5. RMIClass: Instruction Method

The instruction methods in this subsection send a "TP Program Instruction packet" to the controller to append a TP instruction to RMI_MOVE. These methods are only available after calling the init_move method.

All instruction methods will return the sequence ID of the appended TP instruction.

The execution result of the appended TP instruction is returned as a packet from the controller when the TP instruction is executed. However, this packet is not received within the instruction method, but is instead received by the receiver thread of the RMI object. If the receiver thread receives a packet with an error, the RMI object will go into an error state, and an exception will be raised when the next method is called.

If there are eight or more appended TP instructions pending to be executed, any further TP instructions will be kept on hold within the method until execution of RMI_MOVE proceeds and the pending TP instructions is reduced to seven or fewer.

Method List :

wait_di, wait_time, sel_uframe, sel_utool, sel_payload, call, linear, joint, linear_incremental, joint_incremental, circular, circular_incremental

1.3.5.1. RMI.wait_di Method

Format: wait_di(index:int, onoff:bool=True) -> int

This method appends a "WAIT DI[index] = onoff" instruction.

The "onoff" argument should only be set to False when waiting for DI is OFF.

1.3.5.2. RMI.wait_time Method

Format: wait_time(sec:float) -> int

This method appends a “WAIT sec(sec)” instruction.

1.3.5.3. RMI.sel_uframe Method

Format: sel_uframe(ufnum:int) -> int

This method appends a “UFRAME_NUM = ufnum” instruction to set the user frame number.

1.3.5.4. RMI.sel_utool Method

Format: set_utool(utnum) -> int

This method appends a “UTOOL_NUM = utnum” instruction to set tool frame number.

1.3.5.5. RMI.sel_payload Method

Format: sel_payload(snum:int) -> int

This method appends a “PAYLOAD = snum” instruction.
The “snum” argument indicates the schedule number to be set.

1.3.5.6. RMI.call Method

Format: call(program:str) -> int

This method appends a “CALL program” instruction. The “program” argument indicates the name of the TP or KAREL program, without its file extension.

1.3.5.7. RMI.linear Method

Format: linear(pos:Cartesian | Joint | dict, speedtype:str, speed:int,
termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends a “L P[n] speed(speedtype) [termtype] termvalue” linear motion instruction.
The “pos” argument indicates the target position with Cartesian or Joint representation. In a multi-motion group system, dict can be specified as shown below. Here, pos1 and pos2 can each be specified as Cartesian or Joint respectively.
rmi.linear(pos={‘G1’:pos1, ‘G2’:pos2}, …
The “speedtype” argument indicates the motion speed units, selected from below (case-insensitive).

- mmSec' (mm/sec)
- InchMin' (0.1 inch/min)
- Time' (0.1 sec)

- mSec' (msec)

The “speed” argument indicates the motion speed value, in units set by “speedtype”.

The “termtype” argument indicates the motion termination type, selected from below (case-insensitive). (case-insensitive).

- ‘CNT’
- ‘FINE’

‘CR’ can be specified if the Advanced Constant Path option is ordered.

When specifying 'CNT' in 'termtype', the robot will not move until the next motion instruction is added. To execute the next motion without waiting, the 'NoBlend' option which will be explained later, can be used.

When 'CNT' is specified in 'termtype', the CNT value can be specified from 1-100 in 'termvalue'. If not specified, the value will be set to 50 by default. If “termtype” is ‘FINE’, this argument can be omitted.

The “option” argument is for additional motion modifiers, and can be set by specifying the necessary items from the following, as a dictionary.

Key (str)	Type of the value	Meaning
‘ACC’	int	Adds an acceleration option, where the valid range of values is 1-100.
‘OffsetPRNumber’	int	Adds a position register offset. For example, when 2 is specified, “Offset, PR[2]” will be added.
‘ToolOffsetPRNumber’	int	Adds a tool position register offset. For example, when 2 is specified, “Tool_Offset, PR[2]” will be added.
‘VisionPRNumber’	int	Adds a vision offset to the linear motion instruction.
‘WristJoint’	bool	Adds a Wrist Joint motion (“Wjnt”) option if ‘ON’ or True. This is valid only for Linear motion, and will be ignored with Joint motion.
‘MROT’	bool	Adds a MROT motion option if ‘ON’ or True. This is available only if R640, MROT Instruction is installed.
‘LCBType’	str	Adds a Local Condition Block, selected from below (case-insensitive). • ‘TB’ : Time Before • ‘TA’ : Time After • ‘DB’ : Distance Before
‘LCBValue’	int	Values for ‘LCBType’. Units are msec for TB/TA, 0.01mm for DB. The motion modifiers LCBType and LCBValue are used for operating output signals, when the robot reaches a certain position or distance relative to the target position.
‘PortType’	int	DO if 1, RO if 2 is specified. GO and AO are not supported.
‘PortNumber’	int	The port number for I/O specified in ‘PortType’.
‘PortValue’	bool	‘ON’ or True to turn on the I/O specified in ‘PortType’, ‘OFF’ or False to turn off.
‘ALIM’	int	Adds an Acceleration Limit motion modifier, with units [mm/sec2] and a max value of 10000. This is only available if R921, Acceleration Limit Motion Option is installed.
‘ALIMREG’	int	Adds an Acceleration Limit with Register motion modifier to the motion instruction, where the value is a register number.
‘NoBlend’	bool	Motion modifier unique to RMI. ‘ON’ or True allows a motion with CNT termination type to be executed without waiting for the next motion. ‘NoBlend’ is always OFF by default.
‘Coord’	bool	Only available if using Coordinated Motion Function in a multi-group system. ‘ON’ or True allows 2 groups of robots to be moved in coordinated motion.

For example, the following “option” will result in additional motion modifier “TA 0.10 sec DO[3]=ON”.

```
{
'LCBType': 'TA',
'LCBValue': 100,
'PortType': 1,
'PortNumber': 3,
'PortValue': 'ON',
}
```

1.3.5.8. RMI.linear_incremental Method

Format: linear_incremental(pos:Cartesian | Joint | dict, speedtype:str, speed:int, termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends an incremental linear motion instruction.

The “pos” argument indicates the relative position from current position, instead of the target position. Other than this point, all arguments are similar to that of the linear method.

1.3.5.9. RMI.joint Method

Format: joint(pos:Cartesian | Joint | dict, speedtype:str, speed:int, termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends a joint motion instruction.

Arguments other than “speedtype” are similar to that of the linear method.

The “speedtype” argument indicates the motion speed units, selected from below. (case-insensitive).

- 'Percent' (%)
- 'Time' (0.1 sec)
- 'mSec' (msec)

1.3.5.10. RMI.joint_incremental Method

Format: joint_incremental(pos:Cartesian | Joint | dict, speedtype:str, speed:int, termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends an incremental joint motion instruction.

The “pos” argument indicates the relative position from current position, instead of the target position. Other than this point, all arguments are similar to that of the joint method.

1.3.5.11. RMI.circular Method

Format: circular(pos:Cartesian | dict, viapos:Cartesian | dict, speedtype:str, speed:int, termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends a circular motion instruction.

A circular motion instruction creates a circular path motion with current position as the starting point, “viapos” as the passing point, and “pos” as the end point. The “pos”, “viapos” arguments must be Cartesian objects. Joint objects cannot be used.

In a multi-motion group system, “pos”, “viapos” must be specified as a dictionary as shown below.

rmi.cicular(pos={‘G1’:pos1, ‘G2’:pos2}, viapos={‘G1’:viapos1, ‘G2’:viapos2}, …

All other arguments are similar to those of the linear method.

1.3.5.12. RMI.circular_incremental Method

Format: circular_incremental(pos:Cartesian | dict, viapos:Cartesian | dict, speedtype:str, speed:int, termtype:str, termvalue:int=50, option:dict=None) -> int

This method appends an incremental circular motion instruction.

The “pos”, “viapos” arguments indicate the relative position from the current position, instead of the target position. Other than this point, all arguments are similar to those of the circular method.

1.3.6. RMIClass: Other API

This subsection describes other miscellaneous API that have not been explained in the previous subsections.

Method List：

wait_progress, end_move, motion_option

Member List：

_seqid, _done_id

Exception List：

RMIError

1.3.6.1. RMI.wait_progress Method

Format: wait_progress()

This method pauses the execution of the remote motion program RMI_MOVE.
The Python process is blocked until the TP program instruction last appended is done.

1.3.6.2. RMI.end_move Method

Format: end_move()

This method aborts the remote motion program RMI_MOVE after receiving the return packet of the last appended TP instruction. This method is equivalent to calling an abort method after wait_progress method.
If the last appended instruction is of CNT termination type, or is a wait instruction, since the return packet will only be delivered after executing the next instruction, this method will abort remote motion program after receiving the return packet of the second last appended TP instruction.

1.3.6.3. RMI.motion_option Method

Format: motion_option([refer to explanation]) -> dict

This method creates a dictionary to be used in the “option” arguments of the motion instruction methods.
The arguments available are as shown below.
acc: int=None, Value of ‘ACC’
offsetprnumber: int=None, Value of ‘OffsetPRNumber’
tooloffsetprnumber: int=None, Value of ‘ToolOffsetPRNumber’
visionprnumber: int=None, Value of ‘VisionPRNumber’
wristjoint: bool=None, Value of ‘WristJoint’
mrot: bool=None, Value of ‘MROT’

lcbtype: str=None, Value of ‘LCBType’
lcbvalue: str=None, Value of ‘LCBValue’
porttype: int=None, Value of ‘PortType’
portnumber: int=None, Value of ‘PortNumber’
alim: int=None, Value of ‘ALIM’
alimreg: int=None, Value of ‘ALIMREG’
noblend: bool=None, Value of ‘NoBlend’
coord: bool=None, Value of ‘COORD’

1.3.6.4. RMI._seqid Member Variable

This member stores the sequence ID of the appended TP instruction to be executed next. This member is only for reading, and must not be written to.

1.3.6.5. RMI._done_id Member Variable

This member stores the sequence ID of the previously executed TP instruction. This member is read to check the execution progress of RMI_MOVE, and must not be written to.

1.3.6.6. RMI.RMIError Exception

If an alarm post notification packet is received, or if an error is returned by a return packet, while using an RMI object, this exception will be raised in Python. The exception message will show an error message that corresponds to the ErrorID in the return packet.

Revision record

Edition	Date	Contents
01	Jan., 2025	-

INDEX

<A>

ABORT_PYTHON Built-in Function....6

<C>

Cartesian Class....17
Constructor....41
Controller Settings....7
Core Class....24
Core.clear_alarm Method....26
Core.connect Method....24
Core.disconnect Method....24
Core.get_available Method....26
Core.get_datatable Method....25
Core.get_datatable2 Method....25
Core.get_timeout Method....25
Core.read_{I/O} Methods....25
Core.set_timeout Method....25
Core.write_{I/O} Methods....25

<D>

DataAlarm Class....34
DataAlarm.getvalue Method....34
DataCurPos Class....30
DataCurPos.getvalue Method....31
DataIO Class....36
DataIO.getvalue Method....36
DataIO.reset Method....36
DataIO.setvalues Method....36
DataIO.update Method....36
DataNumReg Class....31
DataNumReg.getvalue Method....31
DataNumReg.setvalues Method....31
DataPosReg Class....31
DataPosReg.getvalue Method....32
DataPosReg.setvalue Method....32
DataString.getvalue Method....35
DataString.reset Method....36
DataString.setvalue Method....35
DataString.update Method....35
DataStringClass....35
DataTable Class....26
DataTable.add_alarm Method....29
DataTable.add_curpos Method....27
DataTable.add_io Method....30
DataTable.add_numreg Method....27
DataTable.add_posreg Method....27
DataTable.add_string Method....29
DataTable.add_task Method....28
DataTable.add_variable Method....27
DataTable.add_varpos Method....28
DataTable.clear Method....26
DataTable.refresh Method....30
DataTask Class....33
DataTask.getvalue Method....33
DataVariable.getvalue Method....32
DataVariable.setvalue Method....32
DataVariableClass....32
DataVarPos Class....33
DataVarPos.getvalue Method....33
DataVarPos.setvalue Method....33
Debug Method....9

<E>

Enabling Python Execution....4
ENVIRONMENT AND SPECIFICATIONS....12
Executing and Aborting a Python Script from a TP Program....4

<F>

fanucrobot LIBRARY REFERENCE....17
File System....12

<I>

Important notice....ii
Incremental Motion Instruction....38

<J>

Joint Class....18

<K>

KAREL BUILT-IN FUNCTIONS....5

<L>

Language Specifications....12
Limitations....3

<M>

Motion Instruction for Multi-Group System....40

<O>

Overview....3

<P>

PC Settings....8
Preparing Script Files....4
PROCEDURE TO RUN PYTHON....4

<R>

Read/Write I/O....21
Read/Write Position Registers....23
Read/Write Registers....22
REMOTE DEBUGGING....7
Revision record....52
rmi package....37
RMI. connect Method....41
RMI._done_id Member Variable....51
RMI._seqid Member Variable....51
RMI.abort Method....42
RMI.call Method....47
RMI.circular Method....49

RMI.circular_incremental Method 49
RMI.curpos_cartesian Method 45
RMI.curpos_joint Method 45
RMI.disconnect Method 41
RMI.end_move Method 50
RMI.get_frame_number Method 42
RMI.get_status Method 43
RMI.get_tcp_speed Method 46
RMI.init_move Method 41
RMI.joint Method 49
RMI.joint_incremental Method 49
RMI.linear Method 47
RMI.linear_incremental Method 49
RMI.motion_option Method 50
RMI.pause Method 42
RMI.read_alarm Method 42
RMI.read_di Method 44
RMI.read_position_register Method 45
RMI.read_uframe Method 43
RMI.read_utool Method 44
RMI.reset_alarm Method 46
RMI.resume Method 42
RMI.RMIError Exception 51
RMI.sel_payload Method 47
RMI.sel_uframe Method 47
RMI.sel_utool Method 47
RMI.set_frame_number Method 43
RMI.set_override Method 45
RMI.wait_di Method 46
RMI.wait_progress Method 50
RMI.wait_time Method 47
RMI.write_do Method 44
RMI.write_position_register Method 45
RMI.write_uframe Method 44
RMI.write_utool Method 44
RMIClass : Communication Methods and Constructor .41
RMIClass: Command Method 41
RMIClass: Instruction Method 46
RMIClass: Other API 50
Robot Position Object 17
robotif Package 19
RUN_PYTHON Built-in Function 5

<S>

Safety precautions iii
Sample Scripts 21,37
Setup 7,20,37
Simple Single Group Motion Instructions 37
Software & Hardware Requirements 3
Standard Library 12
STATUS_PYTHON Built-in Function 6

B-84464EN-26/01

