Software version : V1.31
Document version : V1.31
Original instructions(English)

API manual(V1.31)

M0609 | M0617 | M1013 | M1509 | H2017 |
H2515 | A0509 | A0509S | A0912 | A0912S

Table of Contents

1 Preface

The Doosan Robotics API was developed with the programming language C/C++, and can be used to directly control the Doosan robot controller from a user application, not from a teach pendant. This API manual describes the API functions.

The contents of this manual are current as of the date this manual was written, and product-related information may be modified without prior notification to the user.

For more information on the revised manual, please visit our Robot LAB website. (https://robotlab.doosanrobotics.com/)

1.1 Copyright

1.2 History of Document Creation/Revision

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.0	Initial Creation and Distribution	2018-06-29
1.1	Update new commands	2020-05-20
1.11	Update function name(DRL style)	2020-05-28
1.12	Split the header(add DRCFEx class)	2020-06-08
1.13	Add flange_serial function, etc.	2020-10-19
1.14	Correction of typos / Correction of some functions (movesj/set_safe_stop_reset_type/ close_connection, etc.)	2020-10-19

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.15	Add missing function set_user_home / Flange_serial_read parameter added (timeout)	2021-03-19
1.16	Add missing function servo_off function	2021-03-24
1.17	Modify version information : GL010110	2021-06-09
1.18	Modify version information : GL010111 Add new function : get_override_speed	2021-09-15
1.19	Modify version information : GL010112 Add new function . ikin(extension) . set_monitoring_robot_system . change_collision_detection . add_sw_module . del_sw_module . update_sw_module . release_protective_stop . set_on_monitoring_update_module	2021-11-22
1.2	Modifyversion information : GL010112 Add new function . add new function for real-time control(refer to the other manual). . set_safety_mode . set_on_monitoring_state	2021-12-16

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.23	3.3.21: Change the argument part TOnMonitoringSafetyState -> TOnMonitoringSafetyStateCB Add strcut.MESSAGE_PROGRESS 3.7.14: Argument part TYPE_TYPE -> change to DATA_TYPE Added enum constants SAFETY_MODE / SAFETY_EVENT 3.8.11 Modify the argument to enum -> enum.MODBUS_REGISTER_TYPE Add enum constant COORDINATE_SYSTEM Resolve enum constant ROBOT_STATE notation error delete get_override_speed add realtime control(chapter 4)	2022-03-10
1.24	Edited content related to enum.SAFETY_MODE_EVENT Fixed content related to SAFETY_MODE_EVENT parameter in set_safety_mode example Add port parameter to flange_serial related command (specify available for new flange) Change version information -> GL010115 Add new function - set_workpiece_weight - set_mode_tool_analog_input - set_tool_digital_output_type - set_tool_digital_output_level - get_tool_analog_input Add enum.COG_REFERNCE Add enum.ADD_UP Add enum.OUTPUT_TYPE	2022-04-29

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.25	delete port parameter in flange_serial_open / close function add cause sentence for set_tool / set_workpiece_weight function	2022-05-04
1.26	Fixed enum.CONTROL_SPACE fixed set_on_monitoring_safety_state fixed enum.MOVE_REFERENCE fixed change_operation_speed fixed TOnRobotSystemCB add new property function get_current_posj get_control_space get_current_velj get_desired_posj get_current_tool_flange_posx get_current_velx get_desired_velx get_joint_torque get_external_torque get_tool_force add enum.ROBOT_SPACE add struct.ROBOT_VEL / struct.ROBOT_FORCE	2022-06-02
1.27	fixed drl_stop function parameter : enum.STOP_TYPE -> unsigned char	2022-07-04
1.28	Fix enum.ROBOT_STATE description Add contents of enum.ROBOT_MODE delete enum.ROBOT_SPACE duplicates Add set_palletizing_mode command add move_home notice	2023-02-20
1.29	Add query_modbus_data_list	2023-03-24

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.30	Update Features • Add Arm64 Drfl Binary • Support for Ubuntu 22.04 Version • Improve Realtime Performance • Fix Blending Motion Related Issues • Change return value of flange_serial_read Update Definition • Add enum: MOVE_ORIENTATION, SPIRAL_DIR, ROT_DIR, DR_SERVOJ_TYPE • Add struct: SAFETY_CONFIGURATION_EX2, CONFIG_SAFETY_IO_OP • Add parameters for adjusting speed and acceleration of each axis in movej • Add extended movejx parameters • Add extended amovej parameters • Add extended amovejx parameters • Add extended servoj parameters Update Function • Add get_safety_configuration(Extension) • Update ikin(Add_iter_threshold) • Update ikin_norm • Update movec(Extension) • Update amovec(Extension) • Update move_spiral(Extension) • Update amove_spiral(Extension) • Update servoj(Extension)	2024-11-27

Revision No.	Creation/Revision Pages and Contents	Revision Date
1.31	Update Features • Add option of Controller(DRCF) version(#define DRCF_VERSION) Update Definition • Add MONITORING_CTRLIO_EX2 • Add READ_CTRLIO_INPUT_EX2 • Add READ_CTRLIO_OUTPUT_EX2 • Update GPIO_CTRLBOX_DIGITAL_INDEX for v3 controller • Add SAFETY_CONFIGURATION_EX2_V3 Update Function • Update get_safety_configuration_ex • Update get_safety_configuration with new feature • Remove SetOnMonitoringData (due to overlap with set_on_monitoring_data) • Remove SetOnMonitoringCtrlIO (due to overlap with set_on_monitoring_ctrl_io)	2024-12-03

2 Introduction

This API is composed of functions for direct control of the Doosan robot controller in a separate user application, not a T/P application (user GUI program loaded in robot controller), and has been developed with the C/C++ programming language.

2.1 Installation Guide

2.1.1 Composition of Library

This API is composed of three C/C++ header files, a library file related to this, and other support (third party) library files.

Type	File Name	Description	Remarks
Header File	DRFL.h	Library Function Definition File	
	DRFS.h	Library-related Structure Definition File	
	DRFC.h	Library-related Constant Definition File	
Library File	libDRFL.a	Linux Library File	
	DRFLWin32.lib DRFLWin32.dll	Windows Library File	
Other Support Files	libPocoFoundation.so libPocoFoundation.so.16 libPocoNet.so libPocoNet.so.16	Linux Library Dependency File	Registering library using ldConfig function
	PocoFoundation.lib PocoFoundation.dll PocoNet.lib PocoNet.dll	Windows Dependency Library File	Microsoft Visual C++ 2010 (x86) Redistributable package is needed

2.1.2 Linking Library

Linux Library

A file that has a .conf extension (e.g., DRFLib.conf) is generated in the /etc/ld.so.conf.d/ directory and the *.so file path is added in the file and the corresponding library is set by executing the ldconfig command.

Windows Library

The Windows library may not operate normally unless the Microsoft Visual C++ 2010 (x86) redistributable package is installed.

The project setting to use this API in the C++ project is as follows.

1. Setting of header file (include)
   [Composition Attribute]->[C/C++]->[General]->[Additional Include Directories]
2. Setting of library (lib) path
   [Composition Attribute]->[Linker]->[General]->[Additional Library Directories]

2.1.3 Using Header File

When there is a #include library function-related header file (DRFL.h), when the compile option is set in C++ language, programming is possible using CDRFLEx class, but when set in C language, a "_function name" type global function should be used for programming.

2.1.4 Recommended Operational Specification

The recommended operational specification supported by this library is as follows.

Classification	**Name**	**Recommended Specification**	**Remarks**
Windows	Operating System	Windows 7 / Windows 10	
	CPU Acrchietecture	X86(64-bit & 32-bit)	
	Compliler	Mircrosoft Visual C++ 2010	
Linux	Distros	Ubuntu	
	Kernel and Standard Libraries	Linux 3.x kernel Any GLIBC since 2.0	
	CPU Acrchietecture	X86(64bi & 32-bit)	

Classification	Name	Recommended Specification	Rem arks
	Compliler	GNU GCC 4.x or higher	

2.2 Programming Instructions

2.2.1 Robot Connection/Release

As the robot controller and this API are connected through TCP/IP communication, a connection establishment process is necessary. As normal connection is not possible when connection is tried with other APIs or socket-related functions because the certification process is included in the internal connection process, the connection-related functions of this API must be used.

Also, when two or more robot controllers are used for one network, the IP address of each robot controller shall be changed so that it does not overlap at the T/P application and the connection process shall be executed in each robot controller for normal control.

And as this API uses TCP/IIP communication, performance decline or functional error of the user application can occur depending on the computer performance or the network load.

2.2.2 Robot Initialization

As the robot controller receives and processes various information needed for robot operation through the initialization of the T/P application, the user application composed by using this API must start robot control after checking whether initialization has been completed in the information on robot operation state. The completion of initialization can be checked through TOnMonitoringStateCB, which is a callback function, or get_robot_state, which is a user call function, when connecting normally by using robot connection-related functions.

2.2.3 Management of Control Right

As the robot controller is configured to be controlled only in one application, the logic related to the acquisition and transfer of control right should be realized in the user application using the ManageAccessControl(=manage_access_control) function and TOnChangingAccessControlCB callback function, which are related to control right, after the completion of robot initialization, and control commands should be conveyed only when the control right is possessed. When control commands are conveyed without control right, all control commands are ignored and not processed.

2.2.4 Robot Operation Mode

The robot controller operation mode supports automatic mode and manual mode, and the Set/Getrobotmode function allows you to check the settings and modes.. Automatic mode is used for automatically executing the program composed in DRL, a robot programming language our company provides, and manual mode is for executing a single action (e.g., jog action) for which the TCP velocity of the edge of the robot is restricted to 250 mm/sec for safety. Regarding this, when the movej command, which is a robot motion control function for joint space, needs to be set by manual mode and controlled at maximum speed, overspeed can occur and robot operation can stop. Therefore, caution should be paid when setting the operation mode.

2.2.5 Robot Operation State

The operation state information of the robot has a total of 15 states as follows, and all states excluding reservation use (Nos. 11 ~ 14) can be checked through the OnMonitoringState callback function or get_robot_state, which is a user call function.

Rank	Robot Operation State	Description
0	STATE_INITIALIZING	This is a state of automatic entrance of T/P application, and is an initialization condition for the setting of various parameters. Once initialization is completed, the state is automatically converted into a command standby state.
1	STATE_STANDBY	This is an operable basis state, and is a command standby state.
2	STATE_MOVING	A command operation state that is automatically converted while the robot is moving after the receipt of commands. Once moving is done, the state is automatically converted into a command standby state.
3	STATE_SAFE_OFF	This is a robot stop mode due to a general Servo Off function and operation error, and is in the Servo Off state (the state where the motor and brake power are cut off after control stop).
4	STATE_TEACHING	Direct teaching state
5	STATE_SAFE_STOP	This is a robot pause mode caused by functional and operational error, and is a safety stop state (a state in which only control pause was executed, and a temporary program pause state in the case of automatic mode) - Displayed as STATE_STANBY or STATE_MOVING depending on the robot status
6	STATE_EMERGENCY_STOP:	Emergency stop state
7	STATE_HOMMING	Homing mode state (hardware-based array state of robot)
8	STATE_RECOVERY	Recovery mode state for moving robot into the operation range when that robot has stopped due to errors such as getting out of robot operation range

Rank	Robot Operation State	Description
9	eSTATE_SAFE_STOP2	A state in which conversion into recovery mode is needed due to getting out of robot operation range, although it is the same as the eSTATE_SAFE_STOP state
10	STATE_SAFE_OFF2	A state in which conversion into recovery mode is needed due to getting out of robot operation range, although it is the same as the eSTATE_SAFE_OFF state
11	STATE_RESERVED1	Reservation used
12	STATE_RESERVED2	Reservation used
13	STATE_RESERVED3	Reservation used
14	STATE_RESERVED4	Reservation used
15	STATE_NOT_READY	State for initialization after boot-up of robot controller It is converted into the initialization state by the T/P application.

2.2.6 Robot state transition

The command standby state (STATE_STANDBY) is a basic robot control preparation state, and it carries out motions by automatically converting into STATE_HOMING, STATE_MOVING, or STATE_TEACHING when a control command is received from the user, and if the motion is done without error, it converts again into STATE_STANDBY and waits for user commands.

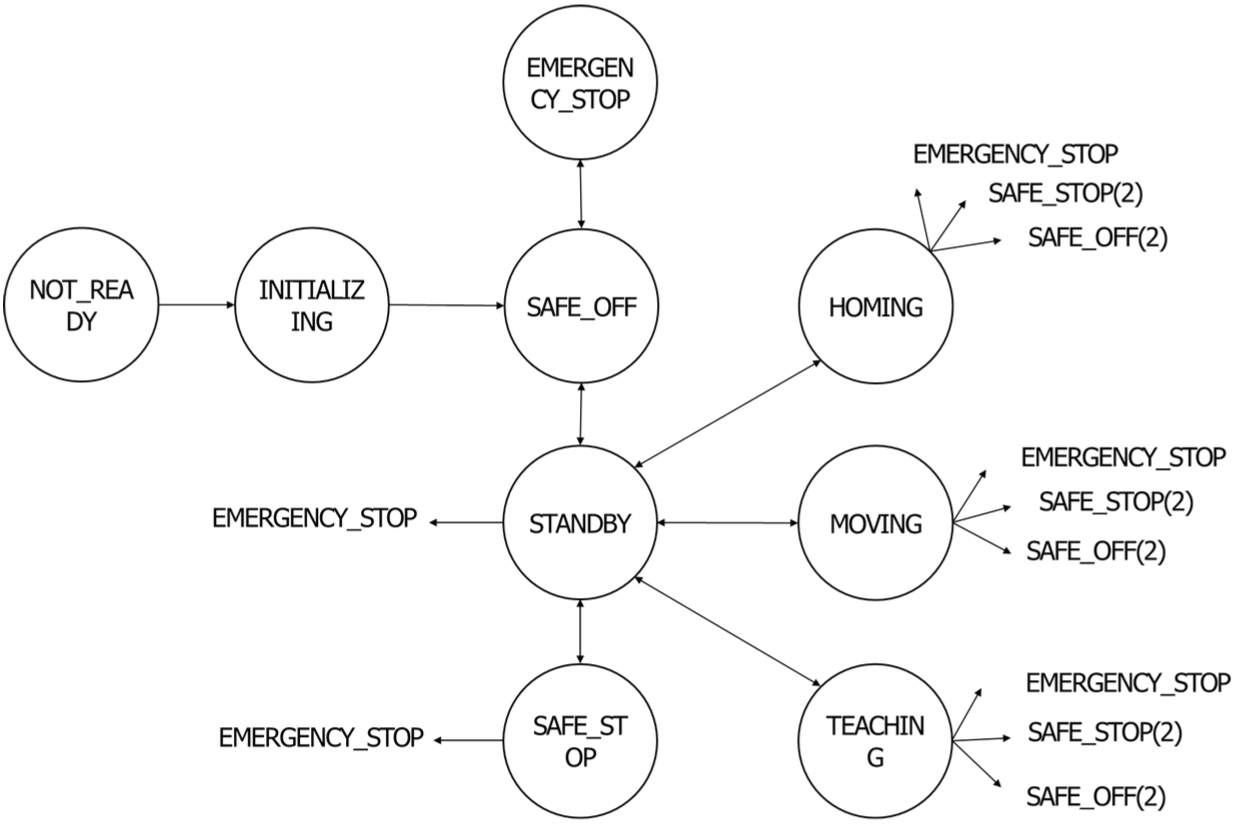
The EMERGENCY_STOP state is converted by the E/M button in whatever states excluding the initialization state (STATE_INITIALIZING) and robot stops. When an internal function or motion error of the robot controller occurs, it is converted into the SAFE_OFF state (motor and brake power cut-off) or SAFE_STOP state (control stop) and the robot stops.

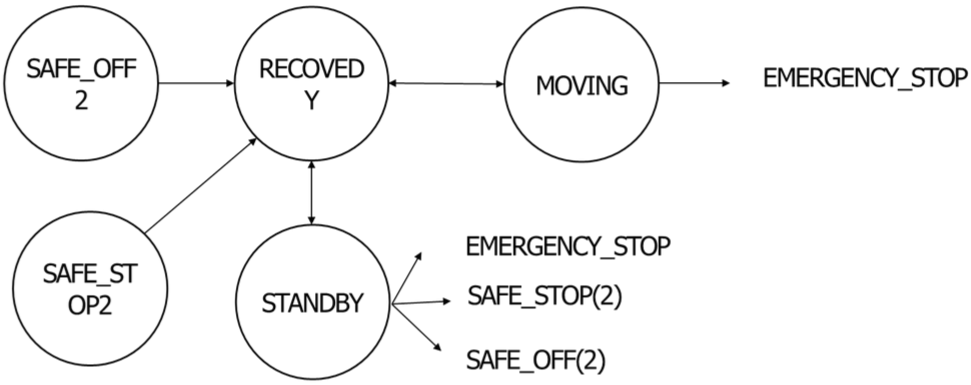
Also, the functions for which state should be converted directly by the user for safety are as follows, and these functions can be executed by the set_robot_control function.

Rank	Robot state control command	Description
0	CONTROL_INIT_CONFIG	It executes the function to convert from STATE_NOT_READY to STATE_INITIALIZING, and only the T/P application executes this function.
1	CONTROL_ENABLE_OPERATION	It executes the function to convert from STATE_INITIALIZING to STATE_STANDBY, and only the T/P application executes this function.

Rank	Robot state control command	Description
2	CONTROL_RESET_SAFET_STOP	It executes the function to convert from STATE_SAFE_STOP to STATE_STANDBY. Program restart can be set in the case of automatic mode.
3	CONTROL_RESET_SAFET_OFF	It executes the function to convert from STATE_SAFE_OFF to STATE_STANDBY.
4	CONTROL_RECOVERY_SAFE_STOP	It executes the S/W-based function to convert from STATE_SAFE_STOP2 to STATE_RECOVERY.
5	CONTROL_RECOVERY_SAFE_OFF	It executes the S/W-based function to convert from STATE_SAFE_OFF2 to STATE_RECOVERY.
6	CONTROL_RECOVERY_BACKDRIVE	It executes the H/W-based function to convert from STATE_SAFE_OFF2 to STATE_RECOVERY. It cannot be converted into STATE_STANDBY, and robot controller power should be rebooted.
7	CONTROL_RESET_RECOVERY	It executes the function to convert from STATE_RECOVERY to STATE_STANDBY.

SAFE_OFF is generally converted into the command standby state (STATE_STANDBY) by the RESET_SAFE_OFF command, which corresponds to servo on, SAFE_STOP is converted into the command standby state (STATE_STANDBY) by RESET_SAFE_STOP user command. Also, when an error that exceeds the robot limit threshold occurs, it is converted into SAFE_OFF2 (motor and brake power cut-off) or SAFE_STOP2 (control stop). In this case, the state should be converted into the command standby state (STATE_STANDBY) by the RESET_RECOVERY command after moving the robot inside the limit threshold to execute robot control normally without the occurrence of errors.





2.2.7 Program Execution and Shutdown

When the program is shut out due to inside/outside errors, or normally, the program stop (drl_stop) command must be executed, and as some time is required for complete internal shutdown of the program, shutdown of the program must be checked in the callback function for program shutdown (TOnProgramStoppedCB), and then program is restarted if needed.

Also, if the program is set as virtual robot system and program execution is shut down, the program is converted into the actual robot system. Therefore, caution should be paid.

2.2.8 Limits on Robot Safety Setting Function

Setting functions related to robot motion (TOOL, TCP, etc.) and safety (safety area, safety stop, safety I/O, etc.) are not provided by this API. They should be set directly in the T/P application. As the initialization process is limited to being executed only in the T/P application during the booting of the robot controller, the robot safety setting must be made in the T/P application.

As an exception, the setting of TOOL or TCP, which are used in motion control commands, is provided as an API function, but this is not interworked with the T/P application. This requires reregistration for use during the rebooting of the robot controller or reoperation of the program. Therefore, using it after setting in the T/P application is recommended.

With respect to this, robot setting is possible only in the following robot pause states: STATE_INITIALIZING, STATE_STANDBY, STATE_SAFE_OFF, and STATE_EMERGENCY. If setting is tried in a state other than these, errors occur and robot motion stops. And a subsequent log and alarm message are generated in the callback function (TOnLogAlarmCB).

3 Definition

3.1 Constant and Enumeration Type

3.1.1 enum.ROBOT_STATE

This is an enumeration type constant that refers to the operation status of robot controller, and is defined as follows.

Rank	Constant Name	Description
0	STATE_INITIALIZING	This is a state of automatic entrance of T/P application, and is an initialization condition for the setting of various parameters.
1	STATE_STANDBY	This is an operable basis state, and is a command standby state.
2	STATE_MOVING	A command operation state that is automatically converted while the robot is moving after the receipt of commands. Once moving is done, the state is automatically converted into a command standby state.
3	STATE_SAFE_OFF	This is a robot pause mode caused by functional and operational error, and is a servo off state (a state in which motor and brake power is cut off after control pause).
4	STATE_TEACHING	Direct teaching state
5	STATE_SAFE_STOP	This is a robot pause mode caused by functional and operational error, and is a safety stop state (a state in which only control pause was executed, and a temporary program pause state in the case of automatic mode)
6	STATE_EMERGENCY_STOP:	Emergency stop state
7	STATE_HOMMING	Homing Mode State (hardware-based array state of robot)
8	STATE_RECOVERY	Recovery mode state for moving robot into the operation range when that robot has stopped due to errors such as getting out of robot operation range
9	eSTATE_SAFE_STOP2	A state in which conversion into recovery mode is needed due to getting out of robot operation range, although it is the same as the eSTATE_SAFE_STOP state

Rank	Constant Name	Description
10	STATE_SAFE_OFF2	A state in which conversion into recovery mode is needed due to getting out of robot operation range, although it is the same as the eSTATE_SAFE_OFF state
11	STATE_RESERVED1	Reservation used
12	STATE_RESERVED2	Reservation used
13	STATE_RESERVED3	Reservation used
14	STATE_RESERVED4	Reservation used
15	STATE_NOT_READY	State for initialization after boot-up of robot controller It is converted into the initialization state by the T/P application.

3.1.2 enum.ROBOT_CONTROL

This is an enumeration type constant that can convert or change the operation status of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	CONTROL_INIT_CONFIG	It executes the function to convert from STATE_NOT_READY to STATE_INITIALIZING, and only the T/P application executes this function.
1	CONTROL_ENABLE_OPERATION	It executes the function to convert from STATE_INITIALIZING to STATE_STANDBY, and only the T/P application executes this function.
2	CONTROL_RESET_SAFET_STOP	It executes the function to convert from STATE_SAFE_STOP to STATE_STANDBY. Program restart can be set in the case of automatic mode.
3	CONTROL_RESET_SAFET_OFF	It executes the function to convert from STATE_SAFE_OFF to STATE_STANDBY.
4	CONTROL_RECOVERY_SAFE_STOP	It executes the S/W-based function to convert from STATE_SAFE_STOP2 to STATE_RECOVERY.
5	CONTROL_RECOVERY_SAFE_OFF	It executes the S/W-based function to convert from STATE_SAFE_OFF2 to STATE_RECOVERY.

Rank	Constant Name	Description
6	CONTROL_RECOVERY_BACKDRIVE	It executes the H/W-based function to convert from STATE_SAFE_OFF2 to STATE_RECOVERY. It cannot be converted into STATE_STANDBY, and robot controller power should be rebooted.
7	CONTROL_RESET_RECOVERY	It executes the function to convert from STATE_RECOVERY to STATE_STANDBY.

3.1.3 enum.MONITORING_SPEED

This is an enumeration type constant that refers to the velocity mode of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	SPEED_NORMAL_MODE	Normal Velocity Mode
1	SPEED_REDUCED_MODE	Deceleration Velocity Mode

3.1.4 enum.SPEED_MODE

This is defined the same as the MONITORING_SPEED enumeration constant.

3.1.5 enum.ROBOT_SYSTEM

This is an enumeration type constant that refers to the operation system of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	ROBOT_SYSTEM_REAL	Actual Robot System
1	ROBOT_SYSTEM_VIRTUAL	Virtual Robot System

3.1.6 enum.ROBOT_MODE

This is an enumeration type constant that refers to the operation mode of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	ROBOT_MODE_MANUAL	Manual Mode
1	ROBOT_MODE_AUTONOMOUS	Automatic Mode
2	ROBOT_MODE_MEASURE	Measurement Mode (currently not supported)

3.1.7 enum.ROBOT_SPACE

This is an enumeration type constant that refers to the coordinate space controlling robot in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	ROBOT_SPACE_JOINT	Joint Space
1	ROBOT_SPACE_TASK	Task Space

3.1.8 enum.SAFE_STOP_RESET_TYPE

This is an enumeration type constant that releases the operation state of the robot controller as STATE_SAFE_STOP and define a series of motions afterward, and is defined as follows.

Rank	Constant Name	Description
0	SAFE_STOP_RESET_TYPE_DEFAULT	Release of Simple State (Manual Mode)
	SAFE_STOP_RESET_TYPE_PROGRAM_STOP	Program Termination (Automatic Mode)
1	SAFE_STOP_RESET_TYPE_PROGRAM_RESUME	Program Restart (Automatic Mode)

3.1.9 enum.MANAGE_ACCESS_CONTROL

This is an enumeration type constant that can obtain and change control right of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MANAGE_ACCESS_CONTROL_FORCE_REQ UEEST	Transmission of message for forced collection of control right

Rank	Constant Name	Description
1	MANAGE_ACCESS_CONTROL_REQUEST,	Transmission of message for request of transfer of control right
2	MANAGE_ACCESS_CONTROL_RESPONSE_YES	Transmission of message for permission of transfer of control right
3	MANAGE_ACCESS_CONTROL_RESPONSE_NO	Transmission of message for rejection of transfer of control right

3.1.10 enum.MONITORING_ACCESS_CONTROL

This is an enumeration type constant that can check the change of control right in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MONITORING_ACCESS_CONTROL_REQUEST,	Reception of message for request of transfer of control right
1	MONITORING_ACCESS_CONTROL_DENY	Reception of message for rejection of transfer of control right
2	MONITORING_ACCESS_CONTROL_GRANT	Reception of message for acquisition of control right
3	MONITORING_ACCESS_CONTROL_LOSS	Reception of message for loss of control right

3.1.11 enum.COORDINATE_SYSTEM

This is an enumeration type constant that means the coordinate system in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	COORDINATE_SYSTEM_BASE	Base coordinate
1	COORDINATE_SYSTEM_TOOL	Tool coordinate
2	COORDINATE_SYSTEM_WORLD	World coordinate

Rank	Constant Name	Description
101	COORDINATE_SYSTEM_USER_MIN	User coordinate (101~200)
200	COORDINATE_SYSTEM_USER_MAX	User coordinate(101~200)

3.1.12 enum.JOG_AXIS

This is an enumeration type constant that refers to each axis that executes jog control in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	JOG_AXIS_JOINT_1	No. 1 joint or axis of robot
1	JOG_AXIS_JOINT_2	No. 2 joint or axis of robot
2	JOG_AXIS_JOINT_3	No. 3 joint or axis of robot
3	JOG_AXIS_JOINT_4	No. 4 joint or axis of robot
4	JOG_AXIS_JOINT_5	No. 5 joint or axis of robot
5	JOG_AXIS_JOINT_6	No. 6 joint or axis of robot
6	JOG_AXIS_TASK_X	X axis of robot TCP
7	JOG_AXIS_TASK_Y	Y axis of robot TCP
8	JOG_AXIS_TASK_Z	Z axis of robot TCP
9	JOG_AXIS_TASK_RX	RX axis of robot TCP
10	JOG_AXIS_TASK_RY	RY axis of robot TCP
11	JOG_AXIS_TASK_RZ	RZ axis of robot TCP

3.1.13 enum.JOINT_AXIS

This is an enumeration type constant that refers to each axis of robot with the standard of joint space coordinate system in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	JOINT_AXIS_1	No. 1 joint or axis of robot
1	JOINT_AXIS _2	No. 2 joint or axis of robot
2	JOINT_AXIS _3	No. 3 joint or axis of robot
3	JOINT_AXIS _4	No. 4 joint or axis of robot
4	JOINT_AXIS _5	No. 5 joint or axis of robot
5	JOINT_AXIS _6	No. 6 joint or axis of robot

3.1.14 enum.TASK_AXIS

This is an enumeration type constant that refers to each axis of robot with the standard of work space coordinate system in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	TASK_AXIS_X	X axis of robot TCP
1	TASK_AXIS_X	Y axis of robot TCP
2	TASK_AXIS_X	Z axis of robot TCP

3.1.15 enum.FORCE_AXIS

This is an enumeration constant that means the definition criteria for the coordinate reference criteria, when performing force control in the robot controller and is defined as follows.

Rank	Constant Name	Description
0	FORCE_AXIS_X	x axis
1	FORCE_AXIS_Y	y axis
2	FORCE_AXIS_Z	z axis
10	FORCE_AXIS_A	x axis rotation

Rank	Constant Name	Description
11	FORCE_AXIS_B	y axis rotation
12	FORCE_AXIS_C	z axis rotation

3.1.16 enum.MOVE_REFERENCE

This is an enumeration type constant that refers to the definition standard for the location to move to when motion is controlled with the work space as the standard in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MOVE_REFERENCE_BASE	Robot base standard
1	MOVE_REFERENCE_TOOL	Robot TCP standard
2	MOVE_REFERENCE_WORLD	Robot world coordinate
101	MOVE_REFERENCE_USER_MIN	User custom coordinate min(101~200)
200	MOVE_REFERENCE_USER_MAX	User custom coordinate max(101~200)

3.1.17 enum.MOVE_MODE

This is an enumeration type constant that refers to the display method for the location to move to when motion is controlled in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MOVE_MODE_ABSOLUTE	Absolute Coordinate
1	MOVE_MODE_RELATIVE	Relative Coordinate

3.1.18 enum.FORCE_MODE

This is an enumeration type constant that refers to the display method for the location to move to when performing for control is controlled in the robot controller, and is defined as follows.

RANK	Constatnt Name	Description
0	FORCE_MODE_ABSOLUTE	Absolute Coordinate

RANK	Constatnt Name	Description
1	FORCE_MODE_RELATIVE	Relative Coordinate

3.1.19 enum.BLENDING_SPEED_TYPE

This is an enumeration type constant that refers to the blending velocity type for each waypoint when motion is controlled in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	BLENDING_SPEED_TYPE_DUPLICATE	Processing by duplicating the velocity of the previous motion and that of the following motion
1	BLENDING_SPEED_TYPE_OVERRIDE	Processing by overriding the velocity of the previous motion to that of the following motion

3.1.20 enum.STOP_TYPE

This is an enumeration type constant that refers to the motion pause type that can stop the motion control when motion is controlled in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	STOP_TYPE_QUICK_STO,	Internal reservation used
1	STOP_TYPE_QUICK	Quick Stop (maintenance of motion trajectory)
2	STOP_TYPE_SLOW	Slow Stop (maintenance of motion trajectory)
3	STOP_TYPE_HOLD	Emergency Stop
	STOP_TYPE_EMERGENCY	Emergency Stop

3.1.21 enum.MOVEB_BLENDING_TYPE

This is an enumeration type constant that refers to the blending motion type for each waypoint when moveb motion is controlled in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MOVEB_BLENDING_TYPE_LINE	Line

Rank	Constant Name	Description
1	MOVEB_BLENDING_TYPE_CIRLCE	Circle

3.1.22 enum.SPLINE_VELOCITY_OPTION

This is an enumeration type constant that refers to velocity control option of each waypoint when spline motion is controlled in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	SPLINE_VELOCITY_OPTION_DEFAULT	Variable velocity motion
1	SPLINE_VELOCITY_OPTION_CONST	Constant velocity motion

3.1.23 enum.GPIO_CTRLBOX_DIGITAL_INDEX

This is an enumeration type constant that refers to the GPIO digital input/output terminal installed in the control box of the robot controller, and is defined as follows.

v2 controller : 0-15

v3 controller : 0-31

순번	상수명	설명
0	GPIO_CTRLBOX_DIGITAL_INDEX_1	Control Box GPIO No. 1 Input/Output Port
1	GPIO_CTRLBOX_DIGITAL_INDEX_2	Control Box GPIO No. 2 Input/Output Port
2	GPIO_CTRLBOX_DIGITAL_INDEX_3	Control Box GPIO No. 3 Input/Output Port
3	GPIO_CTRLBOX_DIGITAL_INDEX_4	Control Box GPIO No. 4 Input/Output Port
4	GPIO_CTRLBOX_DIGITAL_INDEX_5	Control Box GPIO No. 5 Input/Output Port
5	GPIO_CTRLBOX_DIGITAL_INDEX_6	Control Box GPIO No. 6 Input/Output Port
6	GPIO_CTRLBOX_DIGITAL_INDEX_7	Control Box GPIO No. 7 Input/Output Port
7	GPIO_CTRLBOX_DIGITAL_INDEX_8	Control Box GPIO No. 8 Input/Output Port
8	GPIO_CTRLBOX_DIGITAL_INDEX_9	Control Box GPIO No. 9 Input/Output Port

순번	상수명	설명
9	GPIO_CTRLBOX_DIGITAL_INDEX_10	Control Box GPIO No. 10 Input/Output Port
10	GPIO_CTRLBOX_DIGITAL_INDEX_11	Control Box GPIO No. 11 Input/Output Port
11	GPIO_CTRLBOX_DIGITAL_INDEX_12	Control Box GPIO No. 12 Input/Output Port
12	GPIO_CTRLBOX_DIGITAL_INDEX_13	Control Box GPIO No. 13 Input/Output Port
13	GPIO_CTRLBOX_DIGITAL_INDEX_14	Control Box GPIO No. 14 Input/Output Port
14	GPIO_CTRLBOX_DIGITAL_INDEX_15	Control Box GPIO No. 15 Input/Output Port
15	GPIO_CTRLBOX_DIGITAL_INDEX_16	Control Box GPIO No. 16 Input/Output Port
16	GPIO_CTRLBOX_DIGITAL_INDEX_17	Control Box GPIO No. 17 Input/Output Port
17	GPIO_CTRLBOX_DIGITAL_INDEX_18	Control Box GPIO No. 18 Input/Output Port
18	GPIO_CTRLBOX_DIGITAL_INDEX_19	Control Box GPIO No. 19 Input/Output Port
19	GPIO_CTRLBOX_DIGITAL_INDEX_20	Control Box GPIO No. 20 Input/Output Port
20	GPIO_CTRLBOX_DIGITAL_INDEX_21	Control Box GPIO No. 21 Input/Output Port
21	GPIO_CTRLBOX_DIGITAL_INDEX_22	Control Box GPIO No. 22 Input/Output Port
22	GPIO_CTRLBOX_DIGITAL_INDEX_23	Control Box GPIO No. 23 Input/Output Port
23	GPIO_CTRLBOX_DIGITAL_INDEX_24	Control Box GPIO No. 24 Input/Output Port
24	GPIO_CTRLBOX_DIGITAL_INDEX_25	Control Box GPIO No. 25 Input/Output Port
25	GPIO_CTRLBOX_DIGITAL_INDEX_26	Control Box GPIO No. 26 Input/Output Port
26	GPIO_CTRLBOX_DIGITAL_INDEX_27	Control Box GPIO No. 27 Input/Output Port
27	GPIO_CTRLBOX_DIGITAL_INDEX_28	Control Box GPIO No. 28 Input/Output Port
28	GPIO_CTRLBOX_DIGITAL_INDEX_29	Control Box GPIO No. 29 Input/Output Port

순번	상수명	설명
29	GPIO_CTRLBOX_DIGITAL_INDEX_30	Control Box GPIO No. 30 Input/Output Port
30	GPIO_CTRLBOX_DIGITAL_INDEX_31	Control Box GPIO No. 31 Input/Output Port
31	GPIO_CTRLBOX_DIGITAL_INDEX_32	Control Box GPIO No. 32 Input/Output Port

3.1.24 enum.GPIO_CTRLBOX_ANALOG_INDEX

This is an enumeration type constant that refers to GPIO analog input/output terminal installed in the control box of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	GPIO_CTRLBOX_ANALOG_INDEX_1	Control Box GPIO No. 1 Input/Output Port
1	GPIO_CTRLBOX_ANALOG_INDEX_2	Control Box GPIO No. 2 Input/Output Port

3.1.25 enum.GPIO_ANALOG_TYPE

This is an enumeration type constant that refers to the input/output type of the GPIO analog input/output terminal installed in the control box of the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	GPIO_ANALOG_TYPE_CURRENT	Current Input/Output
1	GPIO_ANALOG_TYPE_VOLTAGE	Voltage Input/Output

3.1.26 enum.GPIO_TOOL_DIGITAL_INDEX

This is an enumeration type constant that refers to the GPIO digital input/output terminal installed in the edge of the robot, and is defined as follows.

Rank	Constant Name	Description
0	GPIO_TOOL_DIGITAL_INDEX_1	Robot Edge GPIO No. 1 Input/Output Port
1	GPIO_TOOL_DIGITAL_INDEX_2	Robot Edge GPIO No. 2 Input/Output Port
2	GPIO_TOOL_DIGITAL_INDEX_3	Robot Edge GPIO No. 3 Input/Output Port

Rank	Constant Name	Description
3	GPIO_TOOL_DIGITAL_INDEX_4	Robot Edge GPIO No. 4 Input/Output Port
4	GPIO_TOOL_DIGITAL_INDEX_5	Robot Edge GPIO No. 5 Input/Output Port
5	GPIO_TOOL_DIGITAL_INDEX_6	Robot Edge GPIO No. 6 Input/Output Port

3.1.27 enum.MODBUS_REGISTER_TYPE

This is an enumeration type constant about the modbus register type that can be registered in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	MODBUS_REGISTER_TYPE_DISCRETE_INPUTS	Dircrete Input
1	MODBUS_REGISTER_TYPE_COILS	Coils
2	MODBUS_REGISTER_TYPE_INPUT_REGISTER	Input Register
3	MODBUS_REGISTER_TYPE_HOLDING_REGISTER	Holding Register

3.1.28 enum.DRL_PROGRAM_STATE

This is an enumeration type constant that refers to the execution state of program in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	DRL_PROGRAM_STATE_PLAY	Program Execution State
1	DRL_PROGRAM_STATE_STOP	Program Stop State
2	DRL_PROGRAM_STATE_HOLD	Program Hold State

3.1.29 enum.PROGRAM_STOP_CAUSE

This is an enumeration type constant that refers to the termination reason when the program is terminated in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	PROGRAM_STOP_CAUSE_NORMAL	Normal Program Termination
1	PROGRAM_STOP_CAUSE_FORCE	Forced Program Termination
2	PROGRAM_STOP_CAUSE_ERROR	Program Termination Caused by Inside/Outside Errors

3.1.30 enum.PATH_MODE

This is an enumeration constant that means the path mode and is defined as follows.

Rank	Constant Name	Description
0	PATH_MODE_DPOS	Cumulative
1	PATH_MODE_DVEL	Increment

3.1.31 enum.CONTROL_MODE

This is an enumeration constant that means the robot controller control mode, and is defined as follows.

Rank	Constant Name	Description
3	CONTROL_MODE_POSITION	Position Control Mode
4	CONTROL_MODE_TORQUE	Torque Control mode

3.1.32 enum.DATA_TYPE

This is an enumeration constant that means the data type of the variable to be monitored by the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	DATA_TYPE_BOOL	boolean
1	DATA _TYPE_INT	integer
2	DATA _TYPE_FLOAT	float

Rank	Constant Name	Description
3	DATA_TYPE_STRING	string
4	DATA_TYPE_POSJ	posj
5	DATA_TYPE_POSX	posx
6	DATA_TYPE_UNKNOWN	unknown

3.1.33 enum.VARIABLE_TYPE

It is an enumeration constant that means the type of variable to be monitored by the robot controller and is defined as follows.

Rank	Constant Name	Description
0	VARIABLE_TYPE_INSTALL	Installation Variable
1	VARIABLE_TYPE_GLOBAL	Global Variable

3.1.34 enum.SUB_PROGRAM

This is an enumeration constant that means the operation of a sub-program in the robot controller and is defined as follows.

Rank	Constant Name	Description
0	SUB_PROGRAM_DELETE	Delete Sub Program
1	SUB_PROGRAM_SAVE	Save Sub Program

3.1.35 enum.SINGULARITY_AVOIDANCE

This is an enumeration constant that means the method of avoiding singularity, and is defined as follows.

Rank	Constant Name	Description
0	SINGULARITY_AVOIDANCE_AVOID	Auto Avoidance Mode
1	SINGULARITY_AVOIDANCE_STOP	reduce / warning / task stop

Rank	Constant Name	Description
2	SINGULARITY_AVOIDANCE_VEL	Variable Speed

3.1.36 enum.MESSAGE_LEVEL

This is an enumeration constant that means the type of message to be provided to the user. It is defined as follows.

Rank	Constant Name	Description
0	MESSAGE_LEVEL_INFO	Information
1	MESSAGE_LEVEL_WARN	Warning
2	MESSAGE_LEVEL_ALARM	Error

3.1.37 enum.POPUP_RESPONSE

This is an enumeration constant that means user interaction with pop-up message. It is defined as follows.

Rank	Constant Name	Description
0	POPUP_RESPONSE_STOP	Stop Task
1	POPUP_RESPONSE_RESUME	Resume Task

3.1.38 enum.MOVE_HOME

This is and enumeration constant that related to the coordinate system to be referenced when homming. It is defined as follows.

Rank	Constant Name	Description
0	MOVE_HOME_MECHANIC	Mechanical home position[0,0,0,0,0,0]
1	MOVE_HOME_USER	User home position(custom)

3.1.39 enum.BYTE_SIZE

This is an enumerated constant about the size of data to be transmitted during serial communication, and is defined as follows.

Rank	Constant Name	Description
0	BYTE_SIZE_FIVEBITES	5 bits
1	BYTE_SIZE_SIXBITS	6 bits
2	BYTE_SIZE_SEVENBITS	7 bits
3	BYTE_SIZE_EIGHTBITS	8 bits

3.1.40 enum.STOP_BITS

This is an enumerated constant about Stopbits indicating the end of a frame during serial communication, and is defined as follows.

Rank	Constant Name	Description
0	STOPBITS_ONE	1 bit
1	STOPBITS_TWO	2 bits

3.1.41 enum.PARITY_CHECK

This is an enumerated constant for parity check during serial communication, it is defined as follows.

Rank	Constant Name	Description
0	PARITY_CHECK_NONE	None check
1	PARITY_CHECK_EVEN	Even check
2	PARITY_CHECK_ODD	Odd check

3.1.42 enum.RELEASE_MODE

It is an enumerated constant for setting the subsequent operation at the time of a protective stop, and is defined as follows.

Rank	Constant Name	Description
0	RELEASE_MODE_STOP	Program Stop

Rank	Constant Name	Description
1	RELEASE_MODE_RESUME	Program Resume
2	RELEASE_MODE_RELEASE	Release protective stop
3	RELEASE_MODE_RESET	Interlock Reset

3.1.43 enum.SAFETY_MODE

It is an enumerated constant for setting the subsequent operation at the time of a protective stop, and is defined as follows.

Rank	Constant Name	Description
0	SAFETY_MODE_MANUAL	Manual mode
1	SAFETY_MODE_AUTONOMOUS	Autonomous mode
2	SAFETY_MODE_RECOVERY	Recovery mode
3	SAFETY_MODE_BACDRIVE	Backdrive mode
4	SAFETY_MODE_MEASURE	Measure mode
5	SAFETY_MODE_INITIALIZE	Initializing mode

3.1.44 enum.SAFETY_STATE

It is an enumerated constant for setting the subsequent operation at the time of a protective stop, and is defined as follows.

Rank	Constant Name	Description
0	SAFETY_STATE_BP_START	boot-up start
1	SAFETY_STATE_BP_INIT	boot-up init(config)
2	SAFETY_STATE_VD_STO	mortor stop
3	SAFETY_STATE_VD_SOS	standby

Rank	Constant Name	Description
4	SAFETY_STATE_JH_SOS	jog & homing stop
5	SAFETY_STATE_JH_MOVE	jog & homing move
6	SAFETY_STATE_HG_MOVE	hand guiding move
7	SAFETY_STATE_RV_SOS	recovery standby
8	SAFETY_STATE_RV_MOVE	recovery move
9	SAFETY_STATE_RV_BACK	backdrive mode
10	SAFETY_STATE_RV_HG_MOVE	recovery mode hand guiding
11	SAFETY_STATE_SW_SOS	standalone workspace standby
12	SAFETY_STATE_SW_RUN	standalone workspace move
13	SAFETY_STATE_CW_SOS	collaborative workspace standby
14	SAFETY_STATE_CW_RUN	collaborative workspace run
15	SAFETY_STATE_CM_RUN	collision mute run
16	SAFETY_STATE_AM_RUN	auto-measure run
17	SAFETY_STATE_DRL_JH_SOS	DRL jog & homing standby
18	SAFETY_STATE_DRL_HG_MOVE	handguiding move

3.1.45 enum.SAFETY_MODE_EVENT

It is an enumerated constant to indicate the current state of the safety board, and is defined as follows.

Rank	Constant Name	Description
0	SAFETY_MODE_EVENT_ENTER	Enter
1	SAFETY_MODE_EVENT_MOVE	Move

Rank	Constant Name	Description
2	SAFETY_MODE_EVENT_STOP	Stop

3.1.46 enum.COG_REFERENCE

It is an enumerated constant to indicate the coordinate system based on the position of the center of gravity of the flange, and is defined as follows.

Rank	Constant Name	Description
0	COG_REFERENCE_TCP	TCP
1	COG_REFERENCE_FLANGE	Flange

3.1.47 enum.ADD_UP

It is an enumerated constant to indicate the state of the workpiece of the flange, and is defined as follows.

Rank	Constant Name	Description
0	ADD_UP_REPLACE	Replace workpiece
1	ADD_UP_ADD	Add workpiece
2	ADD_UP_REMOVE	Remove workpiece

3.1.48 enum.OUTPUT_TYPE

It is an enumerated constant to indicate the output type of the digital output installed on the flange, and is defined as follows.

Rank	Constant Name	Description
0	OUTPUT_TYPE_PNP	PNP
1	OUTPUT_TYPE_NPN	NPN

3.2 Definition of Structure

3.2.1 struct.SYSTEM_VERSION

This is structure information for checking detailed version information of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Version Information	char	-	Higher Level Controller Information (32byte)
32	Version Information	char	-	Lower Level Controller Information (32byte)
64	Version Information	char	-	DRL Version No. (32bytes)
96	Version Information	char	-	Inverter Information (32byte)
128	Version Information	char	-	Safety Board Information (32byte)
160	Version Information	char	-	Robot Serial No. (32byte)
192	Version Information	char	-	Robot Model No. (32byte)
224	Version Information	char		JTS Board No. (32byte)
256	Version Information	char		Flange Board No. (32byte)

3.2.2 struct.MONITORING_DATA

This is structure information for checking robot operation data information of the robot controller, and is composed of the following information on five operations.

BYTE#	Field Name	Data Type	Value	Remarks
0	Operation Information (#1)	-	-	Control-related Information
2	Operation Information (#2)	-	-	Joint Space Information
146	Operation Information (#3)	-	-	Task Space Information
327	Operation Information (#4)	-	-	Torque and External Force Information
423	Operation Information (#5)	-	-	Other Robot-related Input Information

Operation information (#1) is composed of control mode information about current robot motions as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Control Mode	uchar	0x00~0x01	Position Control: 0 Torque Control: 1
1	Control Space	uchar	0x00~0x01	Joint Angle Space: 0 Work Coordinate Space: 1

Operation information (#2) is composed of control input/output mode information about robot motions in joint space (based on robot joint) as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information	float	-	Information on Six Current Joint Angles
24	Location Information	float	-	Information on Six Current Joint Angles (Absolute Encoder)
48	Velocity Information	float	-	Information on Six Current Joint Velocity
72	Error Information	float	-	Information on Six Current Joint Errors
96	Location Information	float	-	Information on Six Targets Joint Location
120	Location Information	float	-	Information on Six Targets Joint Velocity

Operation information (#3) is composed of control input/output information about robot motions in joint space (based on tool installed in robot flange) as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information (Tool)	float	-	Information on Six Current Tool Locations
24	Location Information (Flange)	float	-	Information on Six Current Flange Locations
48	Velocity Information	float	-	Information on Six Current Tool Velocity
72	Error Information	float	-	Information on Six Current Tool Errors
96	Location Information	float	-	Information on Six Target Tool Locations
120	Velocity Information	float	-	Information on Six Targets Tool Velocity
144	Pose Information	uchar	0x00~0x07	Information on Robot Pose

- Pose information is one of eight pose information items that the robot can point to at one point.
- The velocity information of operation information (#3) is the current and target velocity information of X, Y, Z, Rx, Ry, and Rz.

Operation information (#4) is composed of control input/output information about robot motions in torque control mode such as torque and external force as follows.

BYTE #	Field Name	Data Type	Value	Remarks
0	Torque Information	float	-	Information on six Currently Calculated Dynamic Torque
24	Torque Information	float	-	Information on Six Currently Measured JTS sensor
48	External Force Information	float	-	Information on six Current Joint External Force by Axis

BYTE #	Field Name	Data Type	Value	Remarks
72	External Force Information	float	-	Information on External Force based on Six Current tools

Operation information (#5) is composed of other robot-related input/output information as follows.

BYTE #	Field Name	Data Type	Value	Remarks
0	Time Information	double	-	Information on Internal Clock Counter
8	I/O Information (#1)	uchar	-	Information on Six Digital On/Off
14	I/O Information (#2)	uchar		Information on Six Digital On/Off
20	Brake Information	uchar	-	Information on state of six brakes
26	Button Information	uint		Information on five robot buttons
46	Current Information	float	-	Information on six motors' current consumption
70	Temperature Information	float		Information on temperature of six inverters

- Brake information 0 means the released brake state and 1 means the locked brake state, which is the state where the robot cannot be operated. (Currently this is not supported.)
- An explanation of I/O information is as follows.

I/O Information	Description
I/O Information (#1)	Information on six digital inputs attached to the edge of the robot
I/O Information (#2)	Information on six digital outputs attached to the edge of the robot

- Button information means the on/off state of five push buttons attached to six axes.

3.2.3 struct.MONITORING_DATA_EX

This is structure information for checking robot operation data information of the robot controller, and is composed of the following information on seven operations.

BYTE#	Field Name	Data Type	Value	Remarks
0	Operation Information(#1)	-	-	Control-related Information
2	Operation Information(#2)	-	-	Joint Space Information
146	Operation Information(#3)	-	-	Task Space Information
327	Operation Information(#4)	-	-	Torque and External Force Information
439	Operation Information(#5)	-	-	World Space Information
643	Operation Information(#6)	-	-	User Space Information
825	Operation Information(#7)	-	-	Other Robot-related Input Information
919	Reserved Space(#1)	-	-	120 bytes Reserved Space
1039	Reserved Space(#2)	-	-	120 bytes Reserved Space

Operation Information(#1) is composed of control mode information about current robot motions as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Control Mode	uchar	0x00~0x01	Position Control : 0 Torque Control : 1
1	Control Space	uchar	0x00~0x01	Joint Angle Space : 0 Task Coordinate Space : 1

Operation Information(#2)는 is composed of control input/output mode information about robot motions in joint space (based on robot joint) as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information	float	-	Information on Six Current Joint Angles
24	Location Information	float	-	Information on Six Current Joint Angles (Absolute Encoder)
48	Velocity Information	float	-	Information on Six Current Joint Velocity
72	Error Information	float	-	Information on Six Current Joint Errors
96	Location Information	float	-	Information on Six Targets Joint Location
120	Location Information	float	-	Information on Six Targets Joint Velocity

Operation Information(#3) is composed of control input/output information about robot motions in joint space (based on tool installed in robot flange) as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information (Tool)	float	-	Information on Six Current Tool Locations
24	Location Information (Flange)	float	-	Information on Six Current Flange Locations
48	Velocity Information	float	-	Information on Six Current Tool Velocity
72	Error Information	float	-	Information on Six Current Tool Errors

BYTE#	Field Name	Data Type	Value	Remarks
96	Location Information	float	-	Information on Six Target Tool Locations
120	Velocity Information	float	-	Information on Six Targets Tool Velocity
144	Pose Information	uchar	0x00~0x07	Information on Robot Pose
145	Rotation Matrix	float	-	Information on 3x3 matrix

- Pose information is one of eight pose information items that the robot can point to at one point.
- The velocity information of operation information (#3) is the current and target velocity information of X, Y, Z, Rx, Ry, and Rz.

Operation information (#4) is composed of control input/output information about robot motions in torque control mode such as torque and external force as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Torque Information	float	-	Information on Six Currently Calculated Dynamic Torque
24	Torque Information	float	-	Information on Six currently measured JTS sensor
48	External Force Information	float	-	Information on Six current joint External force by axis
72	External Force Information	float	-	Information on external force based on six current tools

Operation information (#5) is composed of other robot-related input/output information as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Relationship between World and Base coordinate systems	float	-	Information on Six position deviation

BYTE#	Field Name	Data Type	Value	Remarks
24	Location Information (Tool)	float	-	Information on Six Current Tool Location
48	Location Information (Flange)	float	-	Information on Six Current Flange Location
72	Velocity Information	float	-	Information on Six Current Tool Velocity
96	External Force Information	float	-	Information on Six Current Tool External force by Axis
120	Location Information	float	-	Information on Six Target Tool Location
144	Velocity Information	float	-	Information on Six Target Tool Velocity
168	Rotation Matrix	float	-	Information on 3x3 matrix

Operation information (#6) is composed of control input/output information about robot motions in user space as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	User Coordinate ID	float	-	Coordinate ID(101~200)
1	Parent coordinate system of User coordinate system	float	-	Base : 0 World : 2
2	Location Information (Tool)	float	-	Information on Six Current Tool Location Information
26	Location Information (Flange)	float	-	Information on Six Current Flange Location Information
50	Velocity Information	float	-	Information on Six Current Tool Velocity Information

BYTE#	Field Name	Data Type	Value	Remarks
74	External Force Information	float	-	Information on Six current tool External force by axis
98	Location Information	float	-	Information on Six Target Tool Location
122	Velocity Information	float	-	Information on Six Target Tool Velocity
146	Rotation Matrix	float	-	Information on 3x3 matrix

Operation information (#7) is composed of other robot-related input/output information as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Time Information	double	-	Information on Internal Clock Counter
8	I/O Information (#1)	uchar	-	Information on Six Digital On/ Off
14	I/O Information (#2)	uchar		Information on Six Digital On/ Off
20	Brake Information	uchar	-	Information on state of six brakes
26	Button Information	uint		Information on five robot buttons
46	Current Information	float	-	Information on six motors' current consumption
70	Temperature Information	float		Information on temperature of six inverters

Reserved Space (#1) is a reserve space for expansion of operational information.

The button information is the same as the button information of the operation information (# 7), but includes information in which the data type and button size have been changed.

BYTE#	Field Name	Data Type	Value	Remarks
0	Button Information	uchar	-	Information on six buttons

Reserved Space (# 2) is a reserve space for provided in the for a small model.

The following input / output information is provided in the for a small model.

BYTE#	Field Name	Data Type	Value	Remarks
0	Torque Information	float	-	Information on Six Mesured Current FTS sensor
24	Torque Information	float	-	Information on Six Measured Current CS sensor
48	Velocity Information	float	-	Information on three current measured acceleration sensor
60	Singularity Information	uchar	-	Minimum Singular Value

- If the brake information is 0, it is unlocked, and if it is 1, it is locked and the robot cannot be operated (currently not supported).
- Description of I / O information is as follows.

I/O Information	Description
I/O Information (#1)	Digital input information of 6 attached to the end of the robot
I/O Information (#2)	Digital output information of 6 attached to the end of the robot

- Button information means On / Off status of 5 push buttons attached to 6 axis.

3.2.4 struct.MONITORING_CTRLIO

This is structure information for checking I/O current state information installed in the control box of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Information (#1)	uchar	-	Information on On/Off of 16 Digitals

BYTE#	Field Name	Data Type	Value	Remarks
16	I/O Information (#2)	float		Two Analogs Numeric Information
24	I/O Information (#3)	uchar	-	Information on On/Off of Three Switches
27	I/O Information (#4)	uchar		Information on On/Off of Two Safeties
29	I/O Information (#5)	uchar		Information on On/Off of Two Encoders
31	I/O Information (#6)	uint		Two Encoders Numeric Information
39	I/O Information (#7)	uchar	-	Information on On/Off of 16 Digitals
55	I/O Information (#8)	float		Two Analogs Numeric Information

An explanation of I/O information is as follows.

I/O Information	Description
I/O Information (#1)	Information on 16 digital inputs attached to the control box
I/O Information (#2)	Information on two analog inputs attached to the control box
I/O Information (#3)	Information on the state of three switches attached to the control box and T/P such as the direct teaching button
I/O Information (#4)	Information on two safety-related inputs attached to the control box
I/O Information (#5)	Information on two encoder-related inputs attached to the control box
I/O Information (#6)	Information on two encoder-related raw data attached to the control box

I/O Information	Description
I/O Information (#7)	Information on 16 digital outputs attached to the control box
I/O Information (#8)	Information on two analog outputs attached to the control box

3.2.5 struct.MONITORING_CTRLIO_EX

This is structure information for checking I/O current state information installed in the control box of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Information(#1)	-	-	Information on I/O Digital Input signal
31	I/O Information(#2)	-		Information on I/O Digital Output signal
57	I/O Information(#3)	-	-	Information on Encorder Data signal
69	Reserved Space	-	-	24 bytes Reserved Space

I/O Information (#1) consists of I / O input information attached to Safety B'd in the control box as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital signal	uchar	0x00~0x01	Information on On/Off of 16 Digitals
16	Analog signal	float	-	Two Analogs Numeric Information
24	Switch signal	uchar	0x00~0x01	Information on On/Off of Three Switches
27	Safety signal	uchar	0x00~0x01	Information on On/Off of Two Safeties
29	Analog mode	uchar	0x00~0x01	Two Analogs Numeric Information

- Switch signal information Three switch status information attached to the control box and T / P, such as a direct teaching button, and the safety signal is input information of two safety emergency input signals and two protective-stop signals attached to the control box.

I / O Information (# 2) consists of I / O output information attached to Safety B'd in the control box.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital signal	uchar	0x00~0x01	Information on On/Off of 16 Digitals
16	Analog signal	float	-	Two Analogs Numeric Information
24	Analog mode	uchar	0x00~0x01	Two Analogs mode Numeric Information

I / O Information (# 3) consists of Encorder data information attached to Safety B'd in the control box as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Strobe signal	uchar	0x00~0x01	Information on On/Off of 2 Encorder
2	Raw data	uint	-	Two Encorder Numeric Information
10	Reset signal	uchar	0x00~0x01	Information on reset of 2 Encorder

- The following input / output information is provided for the small model in the Reserved Space.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital input signal of process button	uchar	0x00~0x01	4 Digital On/Off Information

3.2.6 struct.MONITORING_MODBUS

This is structure information for checking modbus I/O current state information when there is modbus I/O information registered in the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Number of I/O	ushort	-	Number of modbus I/O signals
2	I/O Information (#1)	-	-	Modbus I/O signal information
…	…	..	..	Modbus I/O signal information
2+(100*34)	I/O Information (#100)	-	-	Modbus I/O signal information

- I/O information (#N) is composed of the following fields that identify modbus I/O information.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Name	char	-	Modbus I/O Signal Name (32bytes)
2	I/O State	ushort	-	Modbus I/O signal value

3.2.7 struct.LOG_ALARM

This is structure information for checking functional and operational errors when they occur due to inside/ outside factors in the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
1	Log Level	uchar	-	Info: 0 Warning: 1 Alarm: 2
	Classification No.	uchar	-	Classification Code
	Log No.	uint	-	Message No.
5	Parameter (#1)	char	-	Reservation Space: 256
261	Parameter (#2)	char	-	Reservation Space: 256
517	Parameter (#3)	char	-	Reservation Space: 256

Classification and error messages deliver previously defined contents and send related parameters if needed. Refer to the log and alarm definition part for further details.

3.2.8 struct.MOVE_POSB

This is a structure for setting waypoint information when moveb motion is controlled in the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information (#1)	float	-	Information on Six Task Spaces
24	Location Information (#2)	float	-	Information on Six Task Spaces
48	Motion Type	uchar	0x00~0x03	Combination of first motion and second motion Line: 0 Circle: 1
49	Curve Curvature	float		Radius Information (mm Unit)

- moveb consists of a combination of line motions and circle motions. Line motions need one waypoint, except for the starting point, while circle motions need two waypoints, except for the starting point. In other words, location information (#2) is ignored in the case of line motion.
- For reference standard, a base coordinate is selected in the case of base frame, while a tool is selected in the case of coordinate.

3.2.9 struct.ROBOT_POSE

This is a structure for expressing current location information for the get_current_pose command of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
Location Information	Location Information	float	Six Locations Information Items	Location Information

3.2.10 struct.USER_COORDINATE

This is a structure for displaying the current user coordinate system information for the get_user_cart_coord command in the robot controller and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Request ID	uchar	101~120	ID: 101~120
1	Coordinate Reference	uchar	0x00, 0x02	Base : 0 World : 2
2	Location Information	float	-	Location Information

3.2.11 struct.MESSAGE_POPUP

This is a structure for providing typographic message related information is composed of the following fields when a program created through a typographic application is executed in a robot controller

BYTE#	Field Name	Data Type	Value	Remarks
0	Message Information	char	-	256 byte string
256	Message Level	uchar	-	Message : 0 Warning : 1 Alarm : 2
257	Button Type	uchar	-	Resume&Stop : 0 Stop : 1

3.2.12 struct.MESSAGE_INPUT

This is a structure for providing user input related information is composed of the following fields when a robot controller needs to receive and process user input when executing a DRL program

BYTE#	Field Name	Data Type	Value	Remarks
0	Message Information	char	-	256 byte string
256	Data Type	uchar	-	int : 0 float : 1 string : 2 bool : 3

3.2.13 struct.MESSAGE_PROGRESS

This is a structure information to provide information on the current progress when the robot controller executes the DRL program, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Current progress	uchar	-	progress information
256	Total progress	uchar	-	progress information

3.2.14 struct.FLANGE_SERIAL_DATA

Serial communication information is composed of a union, and is composed of a structure for serial communication setting information and serial communication data information.

BYTE#	**Field Name**	**Data Type**	**Value**	**Remarks**
0	Command	uchar	-	0 : Open 1 : Close 2 : Send 3 : Recv
1	Serial communication informatino	-	-	Serial communication information

Union for serial communication information is defined as follows.

BYTE#	**Field Name**	**Data Type**	**Value**	**Remarks**
0	Serial communication setting information	-	-	10bytes structure
	Serial communication data information	-	-	34bytes structure

Serial communication setting information is defined as follows.

BYTE#	**Field Name**	**Data Type**	**Value**	**Remarks**
0	Baudrate	uchar	-	7bytes baudrate value

BYTE#	Field Name	Data Type	Value	Remarks
7	Datasize	uchar	-	Byte Size
8	Parity	uchar	-	Parity Check
9	Stop bits	uchar	-	Stop bits

Serial communication data information is defined as follows

BYTE#	Field Name	Data Type	Value	Remarks
0	Data length	ushort	-	RX_DATA length
2	RX_DATA			32bytes data value

3.2.15 struct.FLANGE_SER_RXD_INFO

This is a structure to receive the transmitted serial data value when using Flange Serial, and it is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Data length	short	-	TX_DATA length
2	TX_DATA	uchar		256bytes data value

3.2.16 struct.READ_FLANGE_SERIAL

This is a structure information to check whether data that can be received exists in the buffer when using Flange Serial is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Data flag	uchar	-	0 : non-receive 1 : received

3.2.17 struct. INVERSE_KINEMATIC_RESPONSE

This is a structure information to pass the calculation return value of inverse kinematics, and it consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Location Information	float[6]	-	Information on Six Current Joint Angles
32	Availability	int	-	0 : normal return 1 : out of operation area 2 : wrist axis singularity

3.2.18 struct.UPDATE_SW_MODULE_RESPONSE

This is a structure information to deliver the details of the current SW module after the SW module installation/ deletion command, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Process history	float[6]	-	0 : install 1 : delete
32	String information for detailed SW Module details	char	-	2048 bytes string

3.2.19 struct.CONFIG_JOINT_RANGE

This is a structure information to set limits in joint space and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Normal mode	struct.JOINT_RANGE	-	Refer to the Definition of Struct
72	Reduced mode	struct.JOINT_RANGE	-	Refer to the Definition of Struct

3.2.20 struct.GENERAL_RANGE

This is a structure information to set limits in force and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Maximum force	float	-	-

BYTE#	Field Name	Data Type	Value	Remarks
4	Maximum power	float	-	-
8	Maximum Speed	float	-	-
12	Maximum momentum	float	-	-

3.2.21 struct.CONFIG_GENERAL_RANGE

This is a structure information to set limits in force and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Normal mode	struct.GENERAL_RANGE	-	Refer to the Definition of Struct
4	Reduced mode	struct.GENERAL_RANGE	-	Refer to the Definition of Struct

3.2.22 struct.POINT_2D

This is a structure information for expressing two-dimensional coordinates, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	x coordinate	float	-	x coordinate
4	y coordinate	float	-	y coordinate

3.2.23 struct.POINT_3D

This is a structure information for expressing 3-dimensional coordinates, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	x-axis	float	-	x coordinate
4	y-axis	float	-	y coordinate
8	z-axis			z coordinate

3.2.24 struct.LINE

This is a structure information to express the distance between two two-dimensional points, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	x-axis	float	-	x coordinate
4	y-axis	float	-	y coordinate
8	z-axis			z coordinate

3.2.25 union.CONFIG_SAFETY_FUNCTION

This is a structure information to set the safety function, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Stop code	tStopCode[17]	-	Refer to the Definition of Below Struct
34	Stop function	unsigned char[17]	-	SF05. Emergency Stop SF06. Proctective Stop SF07. StadnStill Monitoring SF08. Joint Angle Monitoring SF09. Joint Speed Monitoring SF10. Joint Torque Monitoring SF11. Collisoin Detection SF12. TCP Posiiont Monitoring SF13. TCP Orientation Monitoring SF14. TCP Speed Monitoring SF15. TCP Force Monitoring SF16. Momentum Monitoring SF17. Power Mon.

3.2.26 struct._tStopCode

BYTE#	Field Name	Data Type	Value	Remarks
0	Normal	unsigned char	4	-
34	Collaborative	unsigned char	4	-

3.2.27 struct.CONFIG_INSTALL_POSE

This is a structure information to set the install pose, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Gradient	float	-	-
4	Rotation	float	-	-

3.2.28 struct.CONFIG_SAFETY_IO

This is a structure information to set the safety I/O, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	IO port	unsigned char[2][8]	-	16 controller I/O port

3.2.29 struct.CONFIG_SAFETY_IO_EX

This is a structure information to set the safety I/O, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	IO Port	unsigned char[2][8]	-	16 controller I/O port
16	Trigger level	unsigned char[2][8]	-	Trigger level

3.2.30 union.VIRTUAL_FENCE_OBJECT

This is a structure information to limit the operation space, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Cube Operation Space	_CUBE	-	Refer to the Definition of Below Struct
20	Polygon Operation Space	_POLYGON	-	Refer to the Definition of Below Struct
125	Cylinder Operation Space	_CYLINDER	-	Refer to the Definition of Below Struct
137	Reserved Space	unsigned char[10]	-	Reserved Space 10bytes

3.2.31 struct.CONFIG_VIRTUAL_FENCE

This is a structure information to limit the operation space, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Coordinate Reference	unsigned char	-	BASE : 0 TOOL : 1 101~200 : User Coordinate
1	Fence Type	unsigned char	-	0: cube
2	Operation Space	struct.VIRTUAL_FENCE_OBJECT	-	Refer to the Definition of Struct

3.2.32 struct.CONFIG_SAFE_ZONE

This is a structure information to set the safety zone, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Coordinate Reference	unsigned char	-	BASE : 0 TOOL : 1 101~200 : User Coordinate
1	Line	struct.LINE[2]	-	Information of 2 lines
33	Coordinate	struct POINT_2D[3]	-	Information of 3 points

3.2.33 struct.ENABLE_SAFE_ZONE

This is a structure information to enable the safety zone, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Information of activation of the safety zone	unsigned char[3]	-	Information of activation

3.2.34 struct.SAFETY_OBJECT_SPHERE

This is object structure information to set the safe space in a spherical shape, and it consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Radius	float	-	Radius
4	3-dimentional coordinate	struct.POINT_3D	-	3-dimentional coordinate

3.2.35 struct.SAFETY_OBJECT_CAPSULE

This is object structure information to set the safe space in a capsule shape, and it consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Radius	float	-	Radius
4	3-dimentional coordinate	struct.POINT_3D[2]	-	3-dimentional coordinate

3.2.36 struct.SAFETY_OBJECT_CUBE

This is object structure information to set the safe space in a cube shape, and it consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	3-dimentional coordinate	struct.POINT_3D[3]	-	3-dimentional coordinate

3.2.37 struct.SAFETY_OBJECT_OBB

This is object structure information to set the safe space in the form of a pentagon, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	3-dimentional coordinate	struct.POINT_3D[4]	-	3-dimentional coordinate

3.2.38 struct.SAFETY_OBJECT_POLYPRISM

This is object structure information to set the safety space in the form of a polygonal column, and it consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Number of angles	unsigned char	-	Number of angles
1	2-Dimentional coordinate	struct.POINT_2D[10]	-	2-Dimentional coordinate
81	Limitation of Z-axis	float	-	Limitation of Z-axis(Lower)
85	Limitation of Z-axis	float	-	Limitation of Z-axis(Upper)

3.2.39 union.SAFETY_OBJECT_DATA

This is object structure information for safe space setting and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
16	Sphere object	struct.SAFETY_OBJECT_SPHERE	-	Refer to the Definition of Struct
28	Capsule object	struct.SAFETY_OBJECT_CAPSULE	-	Refer to the Definition of Struct
36	Cube object	struct.SAFETY_OBJECT_CUBE	-	Refer to the Definition of Struct
48	Obb object	struct.SAFETY_OBJECT_OBB	-	Refer to the Definition of Struct
89	Polyprism object	struct.SAFETY_OBJECT_POLYPRIS M	-	Refer to the Definition of Struct
100	Reserved Space	unsigned char	-	Reserved Space 100bytes

3.2.40 struct.SAFETY_OBJECT

The safety setting object structure information is composed of the following fields.

BYTE#	Field Name	Data Type	Valu e	Remarks
0	Coordinate Reference	unsigned char	-	BASE : 0 TOOL : 1 101~200 : User Coordination
1	Object Type	unsigned char	-	0 : Sphere 1 : Capsule 2 : Cube 3 : Pentagon 4 : PolyPrism
2	information of Safety Zone Object	union.SAFETY_OBJECT_D ATA	-	Refer to the Definition of Struct

3.2.41 struct.CONFIG_PROTECTED_ZONE

This is a structure information for safe space setting, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Validity	unsigned char[10]	-	Filed validity
10	Object type	struct.SAFETY_OBJECT	-	Refer to the Definition of Struct

3.2.42 struct.CONFIG_COLLISION_MUTE_ZONE_PROPERTY

This is a structure information to set the collision monitoring invalid space, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	ID	char[32]	-	ID
32	Flag	unsigned char	-	Enable flag

BYTE#	Field Name	Data Type	Value	Remarks
33	Collision sensitivity	float	-	Collision sensitivity(0~100%)
37	Object type	struct.SAFETY_OBJECT	-	Refer to the Definition of Struct

3.2.43 struct.CONFIG_COLLISION_MUTE_ZONE

This is a structure information to set the collision monitoring invalid space, and consists of the following fields.

BYTE#	Field Name	Data Type	Val ue	Remarks
0	Validity	unsigned char[10]	-	Field validity
10	Information of collision mute zone	struct.CONFIG_COLLISION_MUTE_ZO NE_PROPERTY	-	Refer to the Definition of Struct

3.2.44 struct.SAFETY_TOOL_ORIENTATION_LIMIT

This is a structure information to set the limitation of tool orientation, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	3-Dimensional coordinate	struct.POINT_3D	-	Refer to the Definition of Struct
12	Target angle	float	-	Target Angle

3.2.45 struct.CONFIG_TOOL_ORIENTATION_LIMIT_ZONE

This is a structure information to set the tool rotation limit area and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Validity	unsigned char[10]	-	Field validity
10	Safety object	struct.SAFETY_OBJECT[10]	-	Safety object
1030	Limitation o tool orientation	struct.SAFETY_TOOL_ORIEN TATION_LIMIT[10]		Rotation limit

3.2.46 struct.CONFIG_NUDGE

This is a structure information to set the Nudge area, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Flag	unsigned char	-	Enable flag
1	Input force	float	-	Input flag
4	Delay	float	-	Delay time

3.2.47 struct.CONFIG_COCKPIT_EX

This is a structure information for cockpit setting and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Flag	unsigned char	-	Enable flag
1	Button	unsigned char[2]	-	0 : Direct teach 1: TCP-Z 2: rotation 3: position
3	Recovery teach	unsigned char	-	Enable recovery teach

3.2.48 struct.CONFIG_IDLE_OFF

This is a structure information for setting auto Servo Off function, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Flag	unsigned char	-	Enable flag
1	Elapse time	float	-	Elapse time(sec)

3.2.49 struct.WRITE_MODBUS_DATA

This is a structure information for writing Modbus TCP data, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Name	char[32]	-	32bytes indentifier
32	IP Address	char[16]	-	TCP IP address
52	Port	unsigned short	-	Port number
56	Slave ID	int	-	1~255
60	Register type	unsgiend char	-	Register type
61	index	unsigned short	-	index
65	value	unsigned short	-	value(0~65535)

3.2.50 struct.WRITE_MODBUS_RTU_DATA

This is a structure information for writing Modbus RTU data, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Name	char[32]	-	32bytes indentifier
32	IP Address	char[16]	-	TCP IP address
52	Port	unsigned short	-	Port number
56	Slave ID	int	-	1~255
60	Register type	unsgiend char	-	Register type
61	index	unsigned short	-	index
65	value	unsigned short	-	value(0~65535)

3.2.51 struct.MODBUS_DATA

This is a structure information to store Modbus data, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Type	unsigned char	-	0: TCP 1: RTU
1	Modbus object	_tData	-	Refer to the Definition of Union

3.2.52 struct.MODBUS_DATA_LIST

This is a structure information to store a lot of Modbus data, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Data number	unsigned short	-	Number of modbus data
2	Modbus data	struct.MODBUS_DATA	-	Refer to the Definition of Struct

3.2.53 struct.CONFIG_WORLD_COORDINATE

This is a structure information to set the world coordinate system, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Configuration type	unsigned char	-	0: world2base 1: base2ref 2: world2ref
2	Target position	float[6]	-	Target position

3.2.54 struct.CONFIG_CONFIGURABLE_IO

This is a structure information to set IO input and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Information	unsigned char[2][16]	-	Information of Safety I/O

3.2.55 struct.CONFIG_CONFIGURABLE_IO_EX

This is a structure information to set IO input and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Information	unsigned char[2][16]	-	Information of Safety I/O
2	Level	unsigned char[2][16]	-	Trigger level

3.2.56 struct.CONFIG_TOOL_SHAPE

This is a structure information to set the tool shape, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Validity	unsigned char[5]	-	Field validity
5	Safety setting object	struct.SAFETY_OBJECT[5]	-	Safety object

3.2.57 struct.CONFIG_TOOL_SYMBOL

This is a structure information to set the tool name, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Name	char[32]	-	32bytes Identifier
32	Tool setting object	struct.CONFIG_TOOL	-	Tool setting object

3.2.58 struct.CONFIG_TCP_SYMBOL

This is a structure information to set the tcp name, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Name	char[32]	-	32bytes Identifier
32	TCP setting object	struct.CONFIG_TCP	-	TCP setting object

3.2.59 struct.CONFIG_TOOL_LIST

This is a structure information to set multiple tool names. It consists of the following fields as structure information.

BYTE#	Field Name	Data Type	Value	Remarks
0	Number	int	-	Number of tool
4	Tool name object	struct.CONFIG_TOOL_SYMBOL[50]	-	Tool name object (Maximum 50)

3.2.60 struct.CONFIG_TOOL_SHAPE_SYMBOL

This is a structure information to set the tool shape name, and consists of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Name	char[32]	-	32bytes Identifier
32	Tool shape setting object	struct.CONFIG_TOOL_SHAPE	-	Tool shape setting object

3.2.61 struct.CONFIG_TCP_LIST

This is a structure information to set multiple tcp names. It consists of the following fields as structure information.

BYTE#	Field Name	Data Type	Value	Remarks
0	Number	int	-	Number of tool
4	TCP name object	struct.CONFIG_TCP_SYMBOL[50]	-	TCP name object (Maximum 50)

3.2.62 struct.CONFIG_TOOL_SHAPE_LIST

This is a structure information to set multiple tool shape names. It consists of the following fields as structure information.

BYTE#	Field Name	Data Type	Value	Remarks
0	Number	int	-	Number of tool shape
4	Tool shape name object	struct.CONFIG_TOOL_SHAPE_SYMBOL[50]	-	Tool shape name object (Maximum 50)

3.2.63 struct.SAFETY_CONFIGURATION_EX

This is a structure information to provide current safety setting information and consists of the following fields.

> This type is deprecated. get_safety_configuration() API doesn't return this type anymore.

BYTE#	Field Name	Data Type	Value	Remarks
0	version	unsigned int	-	Data version
4	Joint space range	struct.CONFIG_JOINT_RANGE	-	Refer to the Definition of Struct
232	Force range	struct.CONFIG_GENERAL_RANGE	-	Refer to the Definition of Struct
280	Collision sensitivity	float	-	collision sensitivity
284	Safety function	union.CONFIG_SAFETY_FUNCTION	-	Refer to the Definition of Struct
318	Tool name setting	struct.CONFIG_TOOL_SYMBOL	-	Refer to the Definition of Struct
390	TCP name setting	struct.CONFIG_TCP_SYMBOL	-	Refer to the Definition of Struct
446	configuration of install pose	struct.CONFIG_INSTALL_POSE	-	Refer to the Definition of Struct
454	safety I/O	struct.CONFIG_SAFETY_IO	-	Refer to the Definition of Struct
470	Operation space	struct.CONFIG_VIRTUAL_FENCE	-	Refer to the Definition of Struct
582	Safety zone	struct.CONFIG_SAFE_ZONE	-	Refer to the Definition of Struct
639	Enable safety zone	struct.ENABLE_SAFE_ZONE	-	Refer to the Definition of Struct
642	Protected zone	struct.CONFIG_PROTECTED_ZONE	-	Refer to the Definition of Struct

BYTE#	Field Name	Data Type	Value	Remarks
1672	Collision mute zone setting	struct.CONFIG_COLLISION_MUTE_ZONE	-	Refer to the Definition of Struct
3072	Tool orientation limit zone setting	struct.CONFIG_TOOL_ORIENTATION_LIMIT_ZONE	-	Refer to the Definition of Struct
4262	Tool shape setting	struct.CONFIG_TOOL_SHAPE	-	Refer to the Definition of Struct
4777	Nudge setting	struct.CONFIG_NUDGE	-	Refer to the Definition of Struct
4786	Cockpit setting	struct.CONFIG_COCKPIT_EX	-	Refer to the Definition of Struct
4790	set auto servo off	struct.CONFIG_IDLE_OFF	-	Refer to the Definition of Struct
4795	TCP list setting	struct.CONFIG_TCP_LIST	-	Refer to the Definition of Struct
7599	Tool list setting	struct.CONFIG_TOOL_LIST	-	Refer to the Definition of Struct
11203	Tool shape list setting	struct.CONFIG_TOOL_SHAPE_LIST	-	Refer to the Definition of Struct
38557	List of activative TCP	char[32]	-	List of activative TCP
38589	List of activative tool	char[32]	-	List of activative tool
38621	List of activative tool shape	char[32]	-	List of activative tool shape
38653	Modbus list	struct.MODBUS_DATA_LIST	-	Refer to the Definition of Struct
45755	World coordination setting	struct.CONFIG_WORLD_COORDINATE	-	Refer to the Definition of Struct
45780	Cws speed	float	-	Speed Cws

BYTE#	Field Name	Data Type	Value	Remarks
45784	Io speed	float	-	Speed Io
45788	IO configuration	struct.CONFIG_CONFIGURABLE_IO	-	Refer to the Definition of Struct

3.2.64 struct.SAFETY_CONFIGURATION_EX2

This is a structure information to provide the current safety configuration details with the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	_iDataVersion	unsigned int	-	Data version
4	_tJointRange	struct.CONFIG_JOINT_RANGE	-	Refer to the Definition of Struct
232	_tGeneralRange	struct.CONFIG_GENERAL_RANGE	-	Refer to the Definition of Struct
280	_fCollisionSensitivity	float	-	collision sensitivity
284	_tSafetyFunc	union.CONFIG_SAFETY_FUNCTION	-	Refer to the Definition of Struct
318	_tTool	struct.CONFIG_TOOL_SYMBOL	-	Refer to the Definition of Struct
390	_tTcp	struct.CONFIG_TCP_SYMBOL	-	Refer to the Definition of Struct
446	_tInstallPose	struct.CONFIG_INSTALL_POSE	-	Refer to the Definition of Struct
454	_tSafetyIO	struct.CONFIG_SAFETY_IO_OP	-	Refer to the Definition of Struct
520	_tSafetySpaceVF	struct.CONFIG_VIRTUAL_FENCE	-	Refer to the Definition of Struct
632	_tSafetySpaceSZ	struct.CONFIG_SAFE_ZONE	-	Refer to the Definition of Struct

BYTE#	Field Name	Data Type	Value	Remarks
689	_tSafetySpaceESZ	struct.ENABLE_SAFE_ZONE	-	Refer to the Definition of Struct
692	_tSafetySpacePZ	struct.CONFIG_PROTECTED_ZONE	-	Refer to the Definition of Struct
1722	_tSafetySpaceCM	struct.CONFIG_COLLISION_MUTE_ZONE	-	Refer to the Definition of Struct
3122	_tSafetySpaceTO	struct.CONFIG_TOOL_ORIENTATION_LIMIT_ZONE	-	Refer to the Definition of Struct
4312	_tSafetySpaceTS	struct.CONFIG_TOOL_SHAPE	-	Refer to the Definition of Struct
4827	_tConfigNudge	struct.CONFIG_NUDGE	-	Refer to the Definition of Struct
4836	_tCockPit	struct.CONFIG_COCKPIT_EX	-	Refer to the Definition of Struct
4840	_tIdleOff	struct.CONFIG_IDLE_OFF	-	Refer to the Definition of Struct
4845	_tConfigTCP	struct.CONFIG_TCP_LIST	-	Refer to the Definition of Struct
7649	_tConfigTool	struct.CONFIG_TOOL_LIST	-	Refer to the Definition of Struct
11253	_tConfigToolShape	struct.CONFIG_TOOL_SHAPE_LIST	-	Refer to the Definition of Struct
38607	_szActiveTcp	char[32]	-	List of activative TCP
38639	_szActiveTool	char[32]	-	List of activative tool
38671	_szActiveToolShape	char[32]	-	List of activative tool shape
38703	_tModbusList	struct.MODBUS_DATA_LIST	-	Refer to the Definition of Struct

BYTE#	Field Name	Data Type	Value	Remarks
45805	_tWorld2BaseRelation	struct.CONFIG_WORLD_COORDINATE	-	Refer to the Definition of Struct
45830	m_CwsSpeedRatio	float	-	Speed Cws
45834	m_IoSpeedRatio	float	-	Speed Io
45838	_tConfigurableIO	struct.CONFIG_CONFIGURABLE_IO	-	Refer to the Definition of Struct

3.2.65 struct.ROBOT_VEL

This is a structure for expressing velocity information of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Velocity Information	float	Six Velocity Information Items	Velocity Information

3.2.66 struct.ROBOT_FORCE

This is a structure for expressing force information of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	Force Information	float	Six Forces Information Items	Force Information

3.2.67 struct.CONFIG_SAFETY_IO_OP

This is a structure information to configure Safety I/O with the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	_iIO	unsigned char[2][16]	-	32 controller I/O port

BYTE#	Field Name	Data Type	Value	Remarks
32	_iTBI_Op	unsigned char	-	TBSFT Input Option
33	_iReserved	unsigned char	-	reserved
34	_iIO_Op	unsigned char[2][16]	-	32 controller I/O port

3.2.68 struct.MONITORING_CTRLIO_EX2

This is structure information for checking I/O current state information installed in the control box of the robot controller, and is composed of the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	I/O Information(#1)	-	-	Information on I/O Digital Input signal
48	I/O Information(#2)	-		Information on I/O Digital Output signal
90	I/O Information(#3)	-	-	Information on Encorder Data signal
102	Reserved Space	-	-	24 bytes Reserved Space

I/O Information (#1) consists of I / O input information attached to Safety B'd in the control box as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital signal	uchar	0x00~0x01	Information on On/Off of 32 Digitals
32	Analog signal	float	-	Two Analogs Numeric Information
40	Switch signal	uchar	0x00~0x01	Information on On/Off of Three Switches
43	Safety signal	uchar	0x00~0x01	Information on On/Off of Two Safeties

BYTE#	Field Name	Data Type	Value	Remarks
46	Analog mode	uchar	0x00~0x01	Two Analogs Numeric Information

- Switch signal information Three switch status information attached to the control box and T / P, such as a direct teaching button, and the safety signal is input information of two safety emergency input signals and two protective-stop signals attached to the control box.

I / O Information (# 2) consists of I / O output information attached to Safety B'd in the control box.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital signal	uchar	0x00~0x01	Information on On/Off of 32 Digitals
32	Analog signal	float	-	Two Analogs Numeric Information
40	Analog mode	uchar	0x00~0x01	Two Analogs mode Numeric Information

I / O Information (# 3) consists of Encorder data information attached to Safety B'd in the control box as follows.

BYTE#	Field Name	Data Type	Value	Remarks
0	Strobe signal	uchar	0x00~0x01	Information on On/Off of 2 Encorder
2	Raw data	uint	-	Two Encorder Numeric Information
10	Reset signal	uchar	0x00~0x01	Information on reset of 2 Encorder

- The following input / output information is provided for the small model in the Reserved Space.

BYTE#	Field Name	Data Type	Value	Remarks
0	Digital input signal of process button	uchar	0x00~0x01	4 Digital On/Off Information

3.2.69 struct.SAFETY_CONFIGURATION_EX2_V3

This is a structure information to provide the current safety configuration details with the following fields.

BYTE#	Field Name	Data Type	Value	Remarks
0	_iDataVersion	unsigned int	-	Data version
4	_tJointRange	struct.CONFIG_JOINT_RANGE	-	Refer to the Definition of Struct
232	_tGeneralRange	struct.CONFIG_GENERAL_RANGE	-	Refer to the Definition of Struct
280	_fCollisionSensitivity	float	-	collision sensitivity
284	_tSafetyFunc	union.CONFIG_SAFETY_FUNCTION	-	Refer to the Definition of Struct
318	_tTool	struct.CONFIG_TOOL_SYMBOL	-	Refer to the Definition of Struct
390	_tTcp	struct.CONFIG_TCP_SYMBOL	-	Refer to the Definition of Struct
446	_tInstallPose	struct.CONFIG_INSTALL_POSE	-	Refer to the Definition of Struct
454	_tSafetyIO	struct.CONFIG_SAFETY_IO_OP	-	Refer to the Definition of Struct
520	_tSafetySpaceVF	struct.CONFIG_VIRTUAL_FENCE	-	Refer to the Definition of Struct
632	_tSafetySpaceSZ	struct.CONFIG_SAFE_ZONE	-	Refer to the Definition of Struct
689	_tSafetySpaceESZ	struct.ENABLE_SAFE_ZONE	-	Refer to the Definition of Struct
692	_tSafetySpacePZ	struct.CONFIG_PROTECTED_ZONE	-	Refer to the Definition of Struct

BYTE#	Field Name	Data Type	Value	Remarks
1722	_tSafetySpaceCM	struct.CONFIG_COLLISION_MUTE_ZONE	-	Refer to the Definition of Struct
3122	_tSafetySpaceTO	struct.CONFIG_TOOL_ORIENTATION_LIMIT_ZONE	-	Refer to the Definition of Struct
4312	_tSafetySpaceTS	struct.CONFIG_TOOL_SHAPE	-	Refer to the Definition of Struct
4827	_tConfigNudge	struct.CONFIG_NUDGE	-	Refer to the Definition of Struct
4836	_tCockPit	struct.CONFIG_COCKPIT_EX	-	Refer to the Definition of Struct
4840	_tIdleOff	struct.CONFIG_IDLE_OFF	-	Refer to the Definition of Struct
4845	_tConfigTCP	struct.CONFIG_TCP_LIST	-	Refer to the Definition of Struct
7649	_tConfigTool	struct.CONFIG_TOOL_LIST	-	Refer to the Definition of Struct
11253	_tConfigToolShape	struct.CONFIG_TOOL_SHAPE_LIST	-	Refer to the Definition of Struct
38607	_szActiveTcp	char[32]	-	List of activative TCP
38639	_szActiveTool	char[32]	-	List of activative tool
38671	_szActiveToolShape	char[32]	-	List of activative tool shape
38703	_tModbusList	struct.MODBUS_DATA_LIST	-	Refer to the Definition of Struct
45805	_tWorld2BaseRelation	struct.CONFIG_WORLD_COORDINATE	-	Refer to the Definition of Struct
45830	m_CwsSpeedRatio	float	-	Speed Cws

BYTE#	Field Name	Data Type	Value	Remarks
45834	m_IoSpeedRatio	float	-	Speed Io
45838	_iSafetyZoneCount	int	-	zone array length
45842	_tSafetyZone	struct.CONFIG_SAFETY_ZONE	-	zone info
53942	_iUserCoordCount	int	-	coord array length
53946	_tUserCoordinates	struct.CONFIG_USER_COORDINATE_EX2	-	coord info
57946	_tConfigurableIO	struct.CONFIG_CONFIGURABLE_IO_EX	-	Io info

3.3 Definition of Log and Alarm

3.3.1 LOG_LEVEL

This is an enumeration type constant that refers to the alarm level in the robot controller, and is defined as follows.

Rank	**Constant Name**	**Description**
0	LOG_LEVEL_RESERVED	Internal reservation state
1	LOG_LEVEL_SYSINFO	Informative messages about simple functions and operational errors
2	LOG_LEVEL_SYSWARN	Robot stop state caused by simple function and operational error
3	LOG_LEVEL_SYSERROR	Robot stop state caused by safety issue or device error

3.3.2 LOG_GROUP

This is an enumeration type constant that refers to the alarm group bell in the robot controller, and is defined as follows.

Rank	Constant Name	Description
0	LOG_GROUP_RESERVED	
1	LOG_GROUP_SYSTEMFMK	Lower Level Controller (Framework)
2	eLOG_GROUP_MOTIONLIB,	Lower Level Controller (Algorithm)
3	LOG_GROUP_SMARTTP	Higher Level Controller (GUI)
4	LOG_GROUP_INVERTER	Robot Inverter Board
5	LOG_GROUP_SAFETYCONTROLLER	Safety Board (Safety Controller)

3.3.3 LOG_CODE

Log and alarm code are defined as enum variable in the DRSC.h file. Refer to the description of this in the separate definition sheet for log and alarm codes.

3.4 Definition of Callback Function

3.4.1 TOnMonitoringStateCB

Features

This is a callback function for checking changes in operation state information in the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
eState	enum.ROBOT_STATE	-	Refer to the Definition of Constant and Enumeration Type

Return

None

Example

```
void OnMonitoringStateCB(const ROBOT_STATE eState)
{
switch((unsigned char)eState)
    {
        case STATE_SAFE_OFF:
            // Robot controller servo on
drfl.set_robot_control(CONTROL_RESET_SAFET_OFF);
        break;
        default:
        break;
}
}

int main()
{
drfl.set_on_monitoring_state(OnMonitoringStateCB);
}
```

3.4.2 TOnMonitoringDataCB

Features

This is a callback function for checking robot operation data that is the same as the current location of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
*pData	struct. MONITORING_DATA	-	Refer to definition of structure

Return

None

Example

```
void OnMonitoringDataCB(const LPMONITORING_DATA pData)
{
// Displays the joint space robot location data
cout << "joint data "
```

```
<< pData->_tCtrl._tJoint._fActualPos[0]
<< pData->_tCtrl._tJoint._fActualPos[1]
<< pData->_tCtrl._tJoint._fActualPos[2]
<< pData->_tCtrl._tJoint._fActualPos[3]
<< pData->_tCtrl._tJoint._fActualPos[4]
<< pData->_tCtrl._tJoint._fActualPos[5] << endl;
}

int main()
{
drfl.set_on_monitoring_data(OnMonitoringDataCB);
}
```

3.4.3 TOnMonitoringDataExCB

Features

This is a callback function for checking robot operation data that is the same as the current location of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 100 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
*pData	struct. MONITORING_DATA_EX	-	Refer to definition of structure

Return

None

Example

```
void OnMonitoringDataExCB(const LPMONITORING_DATA_EX pData)
{
cout << "joint data "
    << pData->_tCtrl._tJoint._fActualPos[0]
    << pData->_tCtrl._tJoint._fActualPos[1]
    << pData->_tCtrl._tJoint._fActualPos[2]
    << pData->_tCtrl._tJoint._fActualPos[3]
    << pData->_tCtrl._tJoint._fActualPos[4]
    << pData->_tCtrl._tJoint._fActualPos[5] << endl;
}

int main()
{
```

```
Drfl.SetOnMonitoringExData(OnMonitoringDataCB);
}
```

3.4.4 TOnMonitoringCtrlIOCB

Features

This is a callback function for checking I/O state data installed in the control box of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
*pCtrlIO	struct. MONITORING_MODBUS	-	Refer to definition of structure

Return

None.

Example

```
void OnMonitoringCtrlIOCB(const LPMONITORING_CTRLIO pCtrlIO)
{
// Displays the digital input GPIO state data
cout << "gpio data" << endl;
for (int i = 0; i < NUM_DIGITAL; i++)
     cout << "DI#"<< i << ": " << pCtrlIO->_tInput._iActualDI[i] << endl;
}

int main()
{
drfl.SetOnMonitoringCtrlIO(OnMonitoringCtrlIOCB)
}
```

3.4.5 TOnMonitoringCtrlIOExCB

Features

This is a callback function for checking I/O state data installed in the control box of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
*pCtrlIO	struct. MONITORING_CTRLIO_EX	-	Refer to definition of structure

Return

None.

Example

```
void OnMonitoringCtrlIOExCB(const LPMONITORING_CTRLIO_EX pCtrlIO)
{
    cout << "gpio data" << endl;
for (int i = 0; i < NUM_DIGITAL; i++)
     cout << "DI#"<< i << ": " << pCtrlIO->_tInput._iActualDI[i] << endl;
}

int main()
{
Drfl.SetOnMonitoringCtrlIOEx(OnMonitoringCtrlIOExCB)
}
```

3.4.6 TOnMonitoringModbusCB

Features

This is a callback function for checking modbus I/O state data registered in the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
*pModbus	struct. MONITORING_MODBUS	-	Refer to definition of structure

Return

None.

Example

```
void OnMonitoringModbusCB(const LPMONITORING_MODBUS pModbus)
{
// Displays the modbus IO state
cout << "modbus data" << endl;
for (int i = 0; i < pModbus->_iRegCount; i++)
        cout << pModbus->_tRegister[i]._szSymbol <<": "
<< pModbus->_tRegister[i]._iValue<< endl;
}

int main()
{
drfl.SetOnMonitoringModbus(OnMonitoringModbusCB)
}
```

3.4.7 TOnLogAlarmCB

Features

This is a callback function for checking all alarms and log information generated in the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
* pLogAlarm	struct.LOG_ALARM	-	Refer to definition of structure

Return

None.

Example

```
void OnLogAlarm(LPLOG_ALARM pLogAlarm)
{
    switch(pLogAlarm->_iGroup)
    {
    case LOG_GROUP_SYSTEMFMK:
        switch(pLogAlarm->_iLevel)
        {
        case LOG_LEVEL_SYSINFO:
```

```
                cout << "index(" << pLogAlarm->_iIndex << "), ";
                cout << "param(" << pLogAlarm->_szParam[0]<< ", ";
                cout << "param(" << pLogAlarm->_szParam[1]<< ", ";
                cout << "param(" << pLogAlarm->_szParam[2]<< ")" << endl;
                break;
            default:
                break;
            }
            break;
        default:
            break;
        }
}

int main()
{
    Drfl.SetOnLogAlarm(OnLogAlarmCB)
}
```

3.4.8 TOnMonitoringAccessControlCB

Features

This is a callback function for checking changes in the state of control right (request/permission/reject). As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
eAccCtrl	enum. MONITORING_ACCESS_CONTROL	-	Refer to the Definition of Constant and Enumeration Type

Return

None

Example

```
void OnMonitoringAccessControlCB(const MONITORING_ACCESS_CONTROL eAccCtrl)
{
// Receives the message for transfer of control right
case MONITORING_ACCESS_CONTROL_REQUEST:
    // Rejects the transfer of control right
drfl.manage_access_control(MANAGE_ACCESS_CONTROL_RESPONSE_NO);
```

```
break;
defatul:
break;
}

int main()
{
drfl.SetOnMonitoringAccessControl (OnMonitoringAccessControlCB);
}
```

3.4.9 TOnHommingCompletedCB

Features

This is a callback function for checking whether homing has been completed when the robot controller is in homing control mode. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

None

Return

None

Example

```
void OnHommingCompletedCB()
{
// Generates a message for the completion of homing
cout << "homming completed" << endl;
drfl.Homme(False)

}

int main()
{
drfl.SetOnHommingCompleted(OnHommingCompletedCB);
}
```

3.4.10 TOnTpInitializingCompletedCB

Features

This is a callback function for checking whether initialization has been completed when the robot controller carries out initialization by T/P application after the booting of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

None

Return

None

Example

```
void OnTpInitializingCompletedCB()
{
// Requests control right after checking Tp initialization
    cout << "tp initializing completed" << endl;
drfl.manage_access_control(MANAGE_ACCESS_CONTROL_REQUEST);
 }

int main()
{
drfl.SetOnTpInitializingCompleted(OnTpInitializingCompletedCB);
}
```

3.4.11 TOnMonitoringSpeedModeCB

Features

This is a callback function for checking the current velocity mode of the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
eSpeedMode	enum.MONITORING_SPEED	-	Refer to the Definition of Constant and Enumeration Type

Return

None.

Example

```
void OnMonitoringSpeedModeCB(const MONITORING_SPEED eSpdMode)
{
// Displays the velocity mode
cout << "speed mode: " << (int)eSpeedMode <<endl;
}

int main()
{
drfl.SetOnMonitoringSpeedMode(OnMonitoringSpeedModeCB);
}
```

3.4.12 TOnMasteringNeedCB

Features

This is a callback function for checking if the robot's axes have been twisted due to external force in the robot controller. The axes of robot can be aligned again through the home commands and it can be checked whether the alignment of axes have been completed through the SetOnHomingCompleted callback function. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

None

Return

None.

Example

```
void OnMasteringNeedCB()
{
// Starts the homing mode for the alignment of the robot's axes
drfl.move_home(True);
}

int main()
{
drfl.SetOnMasteringNeedCB(TOnMasteringNeedCB);
}
```

3.4.13 TOnProgramStoppedCB

Features

This is a callback function for checking if program execution in the robot controller has been completely terminated when the program is terminated due to errors or user command during execution in automatic mode. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
eStopCause	enum. PROGRAM_STOP_CAUSE	-	Refer to the Definition of Constant and Enumeration Type

Return

None.

Example

```
void OnProgramStoppedCB(const PROGRAM_STOP_CAUSE eStopCause)
{
// Shows whether restart of program is possible
}

int main()
{
drfl.SetOnProgramStopped(OnProgramStoppedCB);
```

```
}
```

3.4.14 TOnDisconnectedCB

Features

This is a callback function for checking if the connection with the robot controller has been terminated by external force or user. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

None

Return

None.

Example

```
void OnDisconnectedCB()
{
// Needs the reconnection of the robot controller and error handling
}

int main()
{
drfl.SetOnDisconnected (OnDisconnectedCB);
}
```

3.4.15 TOnTpPopupCB

Features

This is a callback function that is called when a user pop-up feature is used in the robot controller.

As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
tPopup	struct.MESSAGE_POPUP	-	Refer to definition of structure

Return

None.

Example

```
void OnTpPopupCB(LPMESSAGE_POPUP tPopup)
{
//When tp popup command is called
}

int main()
{
Drfl.SetOnTpPopup(OnTpPopupCB);
}
```

3.4.16 TOnTpLogCB

Features

This is a callback function that is called when user log features are used in the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
strLog	string	-	256bytes string

Return

None.

Example

```
void OnTpLogCB(const char* strLog)
{
// When tp log command is called
}

int main()
{
Drfl.SetOnTpLog(OnTpLogCB);
```

```
}
```

3.4.17 TOnTpGetUserInputCB

Features

This is a callback function that is called when user input features are used in the robot controller. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
tInput	struct.MESSAGE_INPUT	-	Refer to definition of structure

Return

None.

Example

```
void OnTpGetUserInputCB(LPMESSAGE_INPUT tInput)
{
// When tp user input command is called

}

int main()
{
Drfl.SetOnTpGetUserInput(OnTpGetUserInputCB);
}
```

3.4.18 TOnTpProgressCB

Features

This is a callback function that is called when the robot controller outputs the information of the execution phase. As the callback function is executed automatically in the case of a specific event, a code that requires an excessive execution time (within 50 msec) within the callback function should not be made.

Parameter

Parameter Name	Data Type	Default Value	Description
tProgress	struct.MESSAGE_PROGRESS	-	Refer to definition of structure

Return

None.

Example

```
void OnTpProgressCB(LPMESSAGE_PROGRESS tProgress)
{
//When tp progresscommand is called
}

int main()
{
Drfl.SetOnTpProgress(OnTpProgressCB);
}
```

3.4.19 TOnMonitoringRobotSystemCB

Features

This is a callback function called when the robot operation system is changed in the robot controller. Since the callback function is automatically executed when a specific event occurs, you should not write code that requires an excessive execution time (within 50msec) within the callback function.

Parameter

Parameter Name	Data Type	Default Value	Description
eRobotSystem	enum.ROBOT_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

None.

Example

```
void OnMonitoringRobotSystemCB(ROBOT_STATE eRobotState)
{
//When changing robot operation system
}

int main()
{
Drfl.set_on_monitroing_robot_system(OnMonitoringRobotSystemCB);
}
```

3.4.20 TOnMonitoringSafetyStateCB

Features

This is a callback function called when the safety state is updated in the robot controller. Since the callback function is automatically executed when a specific event occurs, you should not write code that requires an excessive execution time (within 50msec) within the callback function.

Parameter

Parameter Name	Data Type	Default Value	Description
eSafetyState	enum.SAFETY_STATE	-	Refer to the Definition of Structure

Return

None.

Example

```
void OnMonitoringSafetyStateCB (SAFETY_STATE iState)
{
//When safety state is updated
}

int main()
{
Drfl.set_on_monitoring_safety_state(OnMonitoringSafetyState);
}
```

4 Function

4.1 Robot Connection Function

4.1.1 CDRFLEx.open_connection

Features

This is a function for connecting with the robot controller using TCP/IP communication. As TCP/IP is internally fixed, there is no need to designate it separately. When two or more robot controllers are used, the IP address should be changed in the T/P application.

Parameter

Parameter Name	Data Type	Default Value	Description
strIpAddr	string	“192.168.137.100”	Controller IP

Return

Value	Description
0	Error
1	Success

Example

```
CDRFLEx drfl;
bool bConnected = drfl.open_connection("192.168.137.100");
if (bConnected) {
    SYSTEM_VERSION tSysVerion = {'\0', };
    Drfl.get_system_version(&tSysVerion)
    cout << "System version: " << tSysVerion._szController << endl;
}
```

4.1.2 CDRFLEx.close_connection

Features

This is a function for disconnecting the robot controller.

Parameter

None

Return

Value	Description
1	Success

Example

```
Drfl.close_connection();
```

4.2 Robot Property Function

4.2.1 CDRFLEx.get_system_version

Features

This is a function for checking version information on each subsystem that constitutes the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
*pVersion	struct.SYSTEM_VERSION	-	Refer to definition of structure

Return

Value	Description
0	Error

Value	Description
1	Success

Example

```
SYSTEM_VERSION tSysVerion = {'\0', };
Drfl.get_system_version(&tSysVerion);
cout << "version: " << tSysVerion._szController << endl;
```

4.2.2 CDRFLEx.get_library_version

Features

This is a function for checking information on this API.

Parameter

None

Return

Value	Description
Character String (Maximum 32 byte)	Version Information (e.g., GL:010105)

Example

```
char *lpszLibVersion = get_library_version();
cout << "LibVersion: " << lpszLibVersion << endl;
```

4.2.3 CDRFLEx.get_robot_mode

Features

This is a function for checking information on the current operation mode of the robot controller. The automatic mode is a mode for automatically executing motions (programs) configured in sequential order, while manual mode is a mode for executing single motions like jog.

Parameter

None

Return

Value	Description
enum.ROBOT_MODE	Refer to the Definition of Constant and Enumeration Type

Example

```
string strDrlProgram = "\r\n\
loop = 0\r\n\
while loop < 3:\r\n\
 movej(posj(10,10.10,10,10.10), vel=60, acc=60)\r\n\
 movej(posj(00,00.00,00,00.00), vel=60, acc=60)\r\n\
 loop+=1\r\n\
movej(posj(10,10.10,10,10.10), vel=60, acc=60)\r\n";

if (drfl.get_robot_state() == eSTATE_STANDBY) {

if (drfl.get_robot_mode() == ROBOT_MODE_MANUAL) {
    // Manual Mode
       drfl.jog(JOG_AXIS_JOINT_3, MOVE_REFERENCE_BASE, 60.f);
       sleep(2);
 drfl.jog(JOG_AXIS_JOINT_3, MOVE_REFERENCE_BASE, 0.f);
  }
else {
   // Automatic Mode
ROBOT_SYSTEM eTargetSystem = ROBOT_SYSTEM_VIRTUAL;
      drfl.drl_start(eTargetSystem, strDrlProgram)
  }
}
```

4.2.4 CDRFLEx.set_robot_mode

Features

This is a function for setting information on the current operation mode of the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eMode	enum.ROBOT_MODE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Success
1	Error

Example

```
if (drfl.get_robot_mode() == ROBOT_MODE_MANUAL) {
    // Converts to the automatic mode
    drfl.set_robot_mode(ROBOT_MODE_AUTONOMOUS);
}
```

4.2.5 CDRFLEx.get_robot_state

Features

This is a function for checking information on the current operation mode of the robot controller along with the TOnMonitoringStateCB callback function, and the user should transfer the operation state depending on the state using the etRobotControl function for safety.

Parameter

None

Return

Value	Description
enum.ROBOT_STATE	Refer to the Definition of Constant and Enumeration Type

Example

```
if (drfl.get_robot_state() == eSTATE_STANDBY) {
    if (drfl.get_robot_mode() == ROBOT_MODE_MANUAL) {
       // Manual Mode
       drfl.jog(JOG_AXIS_JOINT_3, MOVE_REFERENCE_BASE, 60.f);
       sleep(2);
       drfl.jog(JOG_AXIS_JOINT_3, MOVE_REFERENCE_BASE, 0.f);
 }
}
```

4.2.6 CDRFLEx.set_robot_control

Features

This is a function that the user can set and convert the current operation state in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eControl	enum.ROBOT_CONTROL	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
if (drfl.get_robot_state () == eSTATE_SAFE_OFF) {
    // Servo on
    drfl.set_robot_control(eCONTROL_RESET_SAFET_OFF);
}
else if (drfl.get_robot_state () == eSTATE_SAFE_OFF2) {
    // Enters the Recovery mode
    drfl.set_robot_control(CONTROL_RECOVERY_SAFE_OFF);
}
```

4.2.7 CDRFLEx.set_robot_system

Features

This is a function for setting and changing the current operation robot system in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eRobotSystem	enum.ROBOT_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
ROBOT_SYSTEM eRobotSystem = drfl.get_robot_system();
if(eRobotSystem != ROBOT_SYSTEM_REAL) {
    //Converts to automatic mode
    drfl.set_robot_system(ROBOT_SYSTEM_REAL);
}
else {
  //do somting …
}
```

4.2.8 CDRFLEx.get_robot_speed_mode

Features

This is a function for checking the current velocity mode (normal mode, deceleration mode) in the robot controller along with the TOnMonitoringSpeedModeCB callback function.

Parameter

None

Return

Value	Description
enum.SPEED_MODE	Refer to the Definition of Constant Type and Enumeration Type

Example

```
if (drfl.get_robot_speed_mode()== SPEED_REDUCED_MODE){
   // Changes the speed reduced mode to normal speed mode
drfl.set_robot_speed_mode(SPEED_NORMAL_MODE);
}
```

4.2.9 CDRFLEx.set_robot_speed_mode

Features

This is a function for setting and changing the current operation robot system in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eSpeedMode	enum.SPEED_MODE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
if (drfl.get_robot_speed_mode()== SPEED_REDUCED_MODE){
    // Changes the speed reduced mode to normal speed mode
drfl.set_robot_speed_mode(SPEED_NORMAL_MODE);
}
```

4.2.10 CDRFLEx.get_program_state

Features

This is a function for checking information on the execution state of the program that is currently being executed in automatic mode in the robot controller.

Parameter

None

Return

Value	Description
enum.DRL_PROGRAM_STATE	Refer to the Definition of Constant and Enumeration Type

Example

```
if (drfl.get_program_state() == DRL_PROGRAM_STATE_PLAY){
   // Stops the program execution state
   drfl.drl_stop(STOP_TYPE_SLOW)
}
else if (drfl.get_program_state() == DRL_PROGRAM_STATE_HOLD) {
   // Resumes the program execution state
drfl.drl_resume()
}
```

4.2.11 CDRFLEx.get_robot_system

Features

This is a function for checking information on the current operation robot system (virtual robot, actual robot) in the robot controller.

Parameter

None.

Return

Value	Description
enum.ROBOT_SYSTEM	Refer to the Definition of Constant and Enumeration Type

Example

```
// Checks the current robot system
ROBOT_SYSTEM eRobotSystem = drfl.get_robot_system();

if(eRobotSystem != ROBOT_SYSTEM_REAL) {
    drfl.set_robot_system(ROBOT_SYSTEM_REAL);
}
else {
  //do somting …
}
```

4.2.12 CDRFLEx.set_safe_stop_reset_type

Features

This is a function for defining a series of motions that are executed automatically after the state conversion using set_robot_control function when the information on the operation state of the robot controller is SAFE_STOP. When the robot operation mode is automatic, program replay can be defined and set, and when it is manual, the setting is ignored.

Parameter

Parameter Name	Data Type	Default Value	Description
eResetType	enum.SAFE_STOP_RESET_TYPE	SAFE_STOP_RESET_TYPE_DEFAULT	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	Error

Example

```
void OnMonitoringStateCB(const ROBOT_STATE eState)
{
switch((unsigned char)eState)
    {
case eSTATE_SAFE_STOP:
  if (drfl.get_robot_mode(ROBOT_MODE_AUTONOMOUS) {
        // Replays the program after conversion of state if the current state is an automatic mode
drfl.set_safe_stop_reset_type(SAFE_STOP_PROGRAM_RESUME);
drfl.set_robot_control(eCONTROL_RESET_SAFET_STOP);
}
else {
     // Converts to STATE_STANDBY if the current state is a manual mode
drfl.set_robot_control(eCONTROL_RESET_SAFET_STOP);
}
        break;
//...
```

4.2.13 CDRFLEx.get_current_pose

Features

This is a function for checking information on the current location by axis of the robot according to the coordinates (joint space or task space) in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eSpaceType	enum.ROBOT_SPACE	ROBOT_SPACE_JOINT	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
struct.ROBOT_POSE	Refer to definition of structure

Example

```
LPROBOT_POSE lpPose = drfl.get_current_pose(ROBOT_SPACE_JOINT);
```

```
// Displays the current location of the robot in joint space
for (int k = 0; k < NUM_JOINT; k++ )
    cout << lpPose->_fPosition[k] << endl;
```

4.2.14 CDRFLEx.get_current_posj

Features

This is a function for checking information on the current joint angle by axis of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_POSE	Refer to definition of structure

Example

```
LPROBOT_POSE lpPose = drfl.get_current_posj();
```

4.2.15 CDRFLEx.get_desired_posj

Features

This is a function for checking information on the desired joint angle by axis of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_POSE	Refer to definition of structure

Example

```
LPROBOT_POSE lpPose = drfl.get_desired_posj();
```

4.2.16 CDRFLEx.get_current_velj

Features

This is a function for checking information on the velocity of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_VEL	Refer to definition of structure

Example

```
LPROBOT_VEL lpVel = drfl.get_current_velj();
```

4.2.17 CDRFLEx.get_current_posx

Features

This is a function for calculating the posture and solution space of the current task coordinate system. At this time, the posture is based on 'eTargetRef'.

Parameter

Parameter Name	**Data Type**	**Default Value**	**Description**
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
struct.ROBOT_TASK_POSE	Refer to definition of structure

Example

```
ROBOT_TASK_POSE* result = Drfl.get_current_posx();
float* pos = new float[NUM_TASK];
int sol = result->_iTargetSol;
memcpy(pos, result->_fTargetPos, sizeof(float) * NUM_TASK);
```

4.2.18 CDRFLEx.get_desired_posx

Features

This is a function for calculating the target posture of the current tool. At this time, the posture is based on 'eTargetRef'.

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
enum.ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
ROBOT_POSE* result = Drfl.get_desired_posx();
for(int i = 0; i < NUM_TASK; i++)
{
    cout << result->_fPosition[i] << endl;
}
```

4.2.19 CDRFLEx.get_current_tool_flange_posx

Features

This is a function for checking information on the tool flange pose of the robot in the robot controller. At this time, the posture is based on 'eTargetRef'.

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
struct.ROBOT_POSE	Refer to definition of structure

Example

```
LPROBOT_POSE lpPose1 =
drfl.get_current_tool_flange_posx(COORDINATE_SYSTEM_BASE);
LPROBOT_POSE lpPose2 = drfl.get_desired_posj(COORDINATE_SYSTEM_TOOL);
```

4.2.20 CDRFLEx.get_current_velx

Features

This is a function for checking information on the current task velocity of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_VEL	Refer to definition of structure

Example

```
LPROBOT_VEL lpVel = drfl.get_current_velx();
```

4.2.21 CDRFLEx.get_desired_velx

Feature

This is a function for checking information on the desired task velocity of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_VEL	Refer to definition of structure

Example

```
LPROBOT_VEL lpVel = drfl.get_desired_velx();
```

4.2.22 CDRFLEx.get_joint_torque

Features

This is a function for checking information on the joint torque sensor of the robot in the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_FORCE	Refer to definition of structure

Example

```
LPROBOT_FORCE lpForce = drfl.get_joint_torque();
```

4.2.23 CDRFLEx.get_control_space

Features

This is a function for checking information on the control space of the robot in the robot controller.

Parameter

None

Return

Value	Description
enum.ROBOT_SPACE	Refer to the Definition of Constant and Enumeration Type

Example

```
ROBOT_SPACE eSpace = drfl.get_control_space();
```

4.2.24 CDRFLEx.get_external_torque

Features

This is a function to check the value of the external force acting on the current tool in the reference coordinate system input from the robot controller.

Parameter

None

Return

Value	Description
struct.ROBOT_FORCE	Refer to definition of structure

Example

```
LPROBOT_FORCE lpETT = drfl.get_external_torque();
```

4.2.25 CDRFLEx.get_tool_force

Features

This is a function for checking information on the pose of the robot in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
struct.ROBOT_FORCE	Refer to definition of structure

Example

```
LPROBOT_FORCE lpForce = drfl.get_tool_force();
```

4.2.26 CDRFLEx.get_current_solution_space

Features

This is a function for checking information on the pose of the robot in the robot controller.

Parameter

None

Return

Value	Description
unsigned char(0~7)	Information on the pose of the robot

Example

```
unsigned char iSolutionSpace = Drfl.get_current_solution_spaces();
// Moves (Move J) if the robot maintains the current pose.
float point[6] = { 10, 10, 10, 10, 10, 10 };
Drfl.movejx(point, iSolutionSpace, 30, 30);
```

4.2.27 CDRFLEx.get_last_alarm

Features

This is a function for checking the latest log and alarm code that have occurred in the robot controller.

Parameter

None

Return

Value	Description
Struct.LOG_ALARM*	Refer to definition of structure

Example

```
LPLOG_ALARM pLogAlarm= Drfl.get_last_alarm();
switch(pLogAlarm->_iGroup)
{
case LOG_GROUP_SYSTEMFMK:
    switch(pLogAlarm->_iLevel)
    {
    case LOG_LEVEL_SYSINFO:
        cout << "index(" << pLogAlarm->_iIndex << "), ";
        cout << "param(" << pLogAlarm->_szParam[0]<< ", ";
        cout << "param(" << pLogAlarm->_szParam[1]<< ", ";
        cout << "param(" << pLogAlarm->_szParam[2]<< ")" << endl;
        break;
    default:
        break;
    }
    break;
default:
    break;
}
```

4.2.28 CDRFLEx.get_solution_space

Features

This is a function for calculating solution space

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target joint location for six axes

Return

Value	Description
unsigned char(0~7)	Solution space

Example

```
float p1[6] = {0, 0, 0, 0, 0, 0};
```

```
int sol = Drfl.get_solution_space(p1)
cout << sol << endl;
```

4.2.29 CDRFLEx.get_orientation_error

Features

This is a function to calculate the orientation error value between arbitrary pose ‘fPosition1’ and ‘fPosition2’ for the axial ‘eTaskAxis’.

Parameter

Parameter Name	Data Type	Default Value	Description
fPosition1	float[6]	-	Target task location for six axes
fPosition2	float[6]		Target task location for six axes
eTaskAxis	enum.TASK_AXIS		Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
float	Orientation Error Value

Example

```
float x1[6] = {0, 0, 0, 0, 0, 0};
float x2[6] = {10, 20, 30, 40, 50, 60};
float diff = Drfl.get_orientation_error(x1, x2, TASK_AXIS_X);
cout << diff << endl;
```

4.2.30 CDRFLEx.get_control_mode

Features

This is a function for checking the current control space.

Parameter

None

Return

Value	Description
CONTROL_MODE	Refer to the Definition of Constant and Enumeration Type

Example

```
CONTROL_MODE mode =Drfl.get_control_mode();
cout << mode << endl;
```

4.2.31 CDRFLEx.get_current_rotm

Features

This is a function to check the rotation matrix of the current tool corresponding to the input reference coordinate system (eTargetRef).

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetRef	enum.COORDINATE_SYSTEM	CORODINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
float[3][3]	Rotation Matrix

Example

```
float(*result)[3] = Drfl.get_current_rotm();
for (int i=0; i<3; i++)
{
```

```
        for (int j=0; j<3; j++)
        {
            cout << result[i][j] ;
        }
        cout << endl;
    }
```

4.2.32 CDRFLEx.get_safety_configuration

Features

This is a function for return safety configuration.

Parameter

None

Return

Value	Description
struct.SAFETY_CONFIGURATION_EX2	Refer to the Definition of Struct
struct.SAFETY_CONFIGURATION_EX2_V3	Refer to the Definition of Struct

Example

```
LPSAFETY_CONFIGURATION_EX2 tParam =
drfl.get_safety_configuration();     // In case of DRCF_VERSION == 2
LPSAFETY_CONFIGURATION_EX2_V3 tParam =
drfl.get_safety_configuration(); // In case of DRCF_VERSION == 3
```

4.3 Functions That Register the Callback Functions

4.3.1 CDRFLEx.set_on_monitoring_state

Features

This is a function for registering the callback function that automatically checks changes in the information on the operation state of the robot controller. It is useful when functions that should be executed automatically are made during the change of data. .

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringStateCB	-	Refer to definition of callback function

Return

None

Example

```
void OnMonitoringStateCB(const ROBOT_STATE eState)
{
switch((unsigned char)eState)
    {
        case STATE_SAFE_OFF:
            // Robot controller servo on
drfl.set_robot_control(CONTROL_RESET_SAFET_OFF);
        break;
        default:
        break;
}
}

int main()
{
drfl.set_on_monitoring_state(OnMonitoringStateCB);
}
```

4.3.2 CDRFLEx.set_on_monitoring_data

Features

This is a function for registering the callback function that automatically checks information on operation data of the robot such as the current location in the robot controller. It is useful when functions that should be executed automatically are made during the change of data.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringDataCB	-	Refer to definition of callback function

Return

None

Example

```
void OnMonitoringDataCB(const LPMONITORING_DATA pData)
{
// Displays the joint space robot location data
cout << "joint data "
    << pData->_tCtrl._tJoint._fActualPos[0]
    << pData->_tCtrl._tJoint._fActualPos[1]
    << pData->_tCtrl._tJoint._fActualPos[2]
    << pData->_tCtrl._tJoint._fActualPos[3]
    << pData->_tCtrl._tJoint._fActualPos[4]
    << pData->_tCtrl._tJoint._fActualPos[5] << endl;
}

int main()
{
drfl.set_on_monitoring_data(OnMonitoringDataCB);
}
```

4.3.3 CDRFLEx.set_on_monitoring_data_ex

Features

This is a function for registering the callback function that automatically checks information on operation data of the robot such as the current location in the robot controller. It is useful when functions that should be executed automatically are made during the change of data.

It is activated when the monitoring data version is changed to 1 using the set_up_monitoring_version function.

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringDataExCB	-	Refer to definition of callback function

Return

None

Example

```
void OnMonitoringDataExCB(const LPMONITORING_DATA_EX pData)
{
cout << "joint data "
    << pData->_tCtrl._tUser._fActualPos[0][0]
    << pData->_tCtrl._tUser._fActualPos[0][1]
    << pData->_tCtrl._tUser._fActualPos[0][2]
    << pData->_tCtrl._tUser._fActualPos[0][3]
    << pData->_tCtrl._tUser._fActualPos[0][4]
    << pData->_tCtrl._tUser._fActualPos[0][5] << endl;
}

int main()
{
Drfl.set_on_monitoring_data_ex(OnMonitoringDataExCB);
}
```

4.3.4 CDRFLEx.set_on_monitoring_ctrl_io

Features

This is a function for registering the callback function that automatically checks information on the current state of the I/O installed in the control box of the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringCtrlIOCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringCtrlIOCB(const LPMONITORING_CTRLIO pCtrlIO)
{
// Displays the digital input GPIO state data
cout << "gpio data" << endl;
for (int i = 0; i < NUM_DIGITAL; i++)
cout << "DI#"<< i << ": " << pCtrlIO->_tInput._iActualDI[i] << endl;
}

int main()
{
drfl.set_on_monitoring_ctrl_io(OnMonitoringCtrlIOCB)
}
```

4.3.5 CDRFLEx.set_on_monitoring_ctrl_io_ex

Features

This is a function for registering the callback function that automatically checks information on the current state of the I/O installed in the control box of the robot controller. It is useful when functions that should be executed automatically are made.

It is activated when the monitoring data version is changed to 1 using the set_up_monitoring_version function.

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringCtrlIOExCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringCtrlIOExCB(const LPMONITORING_CTRLIO_EX pCtrlIO)
{
    // Displays the digital input GPIO state data
    cout << "gpio data" << endl;
    for (int i = 0; i < NUM_DIGITAL; i++)
        cout << "DI#"<< i << ": " << pCtrlIO->_tInput._iActualDI[i] <<
endl;
}

int main()
{
    drfl.set_on_monitoring_ctrl_io_ex(OnMonitoringCtrlIOCBEx)
}
```

4.3.6 CDRFLEx.set_on_monitoring_modbus

Features

This is a function for registering the callback function that automatically checks information on the current state of the modbus I/O registered in the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringModbusCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringModbusCB(const LPMONITORING_MODBUS pModbus)
{
    // Displays the modbus IO state
    cout << "modbus data" << endl;
    for (int i = 0; i < pModbus->_iRegCount; i++)
          cout << pModbus->_iRegCount[i]._szSymbol <<": "
                     << pModbus->_iRegCount[i]._iValue<< endl;}
```

```
int main()
{
    drfl.set_on_monitoring_modbus(OnMonitoringModbusCB)
}
```

4.3.7 CDRFLEx.set_on_log_alarm

Features

This is a function for registering the callback function that automatically checks all alarms and log information generated in the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnLogAlarmCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnLogAlarm(LPLOG_ALARM pLogAlarm)
{
    switch(pLogAlarm->_iGroup)
    {
    case LOG_GROUP_SYSTEMFMK:
        switch(pLogAlarm->_iLevel)
        {
        case LOG_LEVEL_SYSINFO:
            cout << "index(" << pLogAlarm->_iIndex << "), ";
            cout << "param(" << pLogAlarm->_szParam[0]<< ", ";
            cout << "param(" << pLogAlarm->_szParam[1]<< ", ";
            cout << "param(" << pLogAlarm->_szParam[2]<< ")" << endl;
            break;
        default:
            break;
        }
        break;
    default:
        break;
    }
}
```

```
int main()
{
    drfl.set_on_log_alarm(OnLogAlarmCB)
}
```

4.3.8 CDRFLEx.set_on_tp_popup

Features

This function is used to register a callback function to check the popup message when the tp_popup command is used in DRL. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnTpPopupCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnTpPopup(LPMESSAGE_POPUP tPopup)
{
    cout << "Popup Message: " << tPopup->_szText << endl;
    cout << "Message Level: " << tPopup->_iLevel << endl;
    cout << "Button Type: " << tPopup->_iBtnType << endl;
}
int main()
{
    drfl.set_on_tp_popup(OnTpPopupCB)
}
```

4.3.9 CDRFLEx.set_on_tp_log

Features

This function is used to register a callback function to check log messages when the tp_log command is used in DRL. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnTpLogCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnTpLog(const char* strLog)
{
    cout << "Log Message: " << strLog << endl;
}
int main()
{
    drfl.set_on_tp_log(OnTpLogCB)
}
```

4.3.10 CDRFLEx.set_on_tp_progress

Features

This function is used to register the callback function to check the information of the execution step when the tp_progress command is used in DRL. It is useful when functions that should be executed automatically are made. Parameter

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnTpProgressCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnTpProgress(LPMESSAGE_PROGRESS tProgress)
{
    cout << "Progress cnt : " << tProgress->_iCurrentCount << endl;
    cout << "Current cnt : " << tProgress->_iCurrentCount << endl;
}
int main()
{
    drfl.set_on_tp_progress(OnTpProgressCB)
}
```

4.3.11 CDRFLEx.set_on_tp_get_user_input

Features

This function is used to register a callback function to check user input when the tp_get_user_input command is used in DRL. It is useful when functions that should be executed automatically are made. Parameter

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnTpGetUserInputCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnTpGetuserInput(LPMESSAGE_INPUT tInput)
{
    cout << "User Input : " << tInput->_szText << endl;
    cout << "Data Type : " << tInput->_iType << endl;
}
int main()
{
    drfl.set_on_tp_get_user_input(OnTpGetUserInputCB)
}
```

4.3.12 CDRFLEx.set_on_monitoring_access_control

Features

This is a function for registering the callback function that automatically checks changes in the state of control right (request/permission/reject) in the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringAccessControlCB	-	Refer to definition of callback function

Return

None

Example

```
void OnMonitoringAccessControlCB(const MONITORING_ACCESS_CONTROL eAccCtrl)
{
    // Receives the message for transfer of control right
    case MONITORING_ACCESS_CONTROL_REQUEST:
        // Rejects the transfer of control right
        drfl.manage_access_control(MANAGE_ACCESS_CONTROL_RESPONSE_NO);
        break;
    defatul:
        break;
}

int main()
{
    drfl.set_on_monitoring_access_control(OnMonitoringAccessControlCB);
}
```

4.3.13 CDRFLEx.set_on_homming_completed

Features

This is a function for registering the callback function that automatically checks whether homing has been completed when the robot controller is in homing control mode. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnHommingCompleted CB	-	Refer to definition of callback function

Return

None.

Example

```
void OnHommingCompletedCB()
{
    // Generates a message for the completion of homing
    cout << "homming completed" << endl;
    drfl.Homme(False)

}

int main()
{
    drfl.set_on_homming_completed(OnHommingCompletedCB);
}
```

4.3.14 CDRFLEx.set_on_tp_initializing_completed

Features

This is a function for registering the callback function that automatically checks whether initialization has been completed when the robot controller carries out initialization by T/P application after the booting of the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnTpInitializingComplete dCB	-	Refer to definition of callback function

Return

None

Example

```
void OnTpInitializingCompletedCB()
{
    // Requests control right after checking Tp initialization
    cout << "tp initializing completed" << endl;
    drfl.manage_access_control(MANAGE_ACCESS_CONTROL_REQUEST);
 }

int main()
{
    drfl.set_on_tp_initializing_completed(OnTpInitializingCompletedCB);
}
```

4.3.15 CDRFLEx.set_on_monitoring_speed_mode

Features

This is a function for registering the callback function that automatically checks the current velocity mode of the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringSpeedModeCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringSpeedModeCB(const MONITORING_SPEED eSpdMode)
{
    // Displays the velocity mode
    cout << "speed mode: " << (int)eSpeedMode <<endl;
}

```

```
int main()
{
    drfl.set_on_monitoring_speed-mode(OnMonitoringSpeedModeCB);
}
```

4.3.16 CDRFLEx.set_on_mastering_need

Features

This is a function for registering the callback function that automatically checks if the robot's axes have been twisted due to external force in the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMasteringNeedCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMasteringNeedCB()
{
    // Starts the homing mode for the alignment of the robot's axes
    drfl.Homme(True)
}

int main()
{
    drfl.set_on_mastering_need(TOnMasteringNeedCB);
}
```

4.3.17 CDRFLEx.set_on_program_stopped

Features

This is a function for registering the callback function that automatically checks if program execution has been completely terminated when the program is terminated due to errors or user command during execution in automatic mode in the robot controller. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnProgramStoppedCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnProgramStoppedCB(const PROGRAM_STOP_CAUSE eStopCause)
{
    // Shows whether restart of program is possible
}

int main()
{
    drfl.set_on_program_stopped(OnProgramStoppedCB);
}
```

4.3.18 CDRFLEx.set_on_disconnected

Features

This is a function for registering the callback function that automatically checks if the connection with the robot controller has been terminated by external force or user. It is useful when functions that should be executed automatically are made.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnDisconnectedCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnDisconnectedCB()
{
    // Needs the reconnection of the robot controller and error handling
}

int main()
{
    drfl.set_on_disconnected(OnDisconnectedCB);
}
```

4.3.19 CDRFLEx.set_on_monitoring_robot_system

Features

This is a function to register a callback function to automatically check when the operating system of the robot controller is changed. This is useful when writing functions that should be executed automatically.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringRobotSystemCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringRobotSystemCB(ROBOT_STATE eRobotState)
{
    //When changing robot operation status
}

int main()
{
    Drfl.set_on_monitroing_robot_system(OnMonitoringRobotSystemCB);
```

```
}
```

4.3.20 CDRFLEx.set_on_monitoring_safety_state

Features

This is a function to register a callback function to automatically check when the safety state of the controller is updated. This is useful when writing functions that should be executed automatically.

Parameter

Parameter Name	Data Type	Default Value	Description
pCallbackFunc	TOnMonitoringSafetyStateCB	-	Refer to definition of callback function

Return

None.

Example

```
void OnMonitoringSafetyStateCB (SafetyState iState)
{
    //When sw module is updated
}

int main()
{
    Drfl.set_on_monitoring_safety_state(OnMonitoringSafetyStateCB);
}
```

4.4 Functions That Manage Control Right

4.4.1 CDRFLEx.manage_access_control

Features

This is a function for sending the control right request message of the robot controller or for processing the user response when the control right request message is received.

Parameter

Parameter Name	Data Type	Default Value	Description
eAccessControl	enum.MANAGE_ACCESS_CONTROL	MANAGE_ACCESS_CONTROL_REQUEST	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
void OnMonitoringAccessControlCB(const MONITORING_ACCESS_CONTROL eAccCtrl)
{
    // Receives the rejection of transfer of control right
    case MONITORING_ACCESS_CONTROL_DENY:
        // Displays no control right
    break;
    defatul:
    break;
}

int main()
{
    drfl.SetOnMonitoringAccessControl(OnMonitoringAccessControlCB);
    // Requests the transfer of control right
    drfl.manage_access_control(MANAGE_ACCESS_CONTROL_REQUEST);

}
```

4.5 Basic Control Functions

4.5.1 CDRFLEx.jog

Features

This is a function for executing the control of jog movement for each axis of the robot in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eJointAxis	enum.JOG_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eMoveReference	enum.MOVE_REFERENCE	-	Refer to the Definition of Constant and Enumeration Type
fVelocity	float	-	jog Velocity (% Unit) +: Positive Direction 0: Stop -: Negative Direction

Return

Value	Description
0	Error
1	Success

Example

```
// Jogs at 60% speed in the positive direction based on the robot base
drfl.jog(JOG_AXIS_JOINT_1, MOVE_REFERENCE_BASE, 60.f);
// Stops
drfl.jog(JOG_AXIS_JOINT_1, MOVE_REFERENCE_BASE, 0.f);
// Jogs at 60% speed in the negative direction based on the robot base
drfl.jog(JOG_AXIS_JOINT_1, MOVE_REFERENCE_BASE, -60.f);
// Stops
drfl.jog(JOG_AXIS_JOINT_1, MOVE_REFERENCE_BASE, 0.f);
```

4.5.2 CDRFLEx.move_home

Features

This is a function for aligning each axis of the robot when the robot's axes have been twisted due to internal/ external errors in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eMode	MOVE_HOME	MOVE_HOME_MECHANIC	Refer to the Definition of Constant and Enumeration Type
bRun	unsigned char	1	0: Stop 1: Motion

Return

Value	Description
0	Error
1	Success

Example

```
// Executes Homing motion
drfl.move_home()
// Stops Homing motion
drfl.move_home(MOVE_HOME_MECHANIC, (unsigned char)0);
```

Caution

When using the Move command after homing is complete, it is recommended to use the wait command after homing.

4.6 Functions That Control Motion

4.6.1 CDRFLEx.movej

Features

This is a function for moving the robot from the current joint location to target joint location in the robot controller.

Parameter

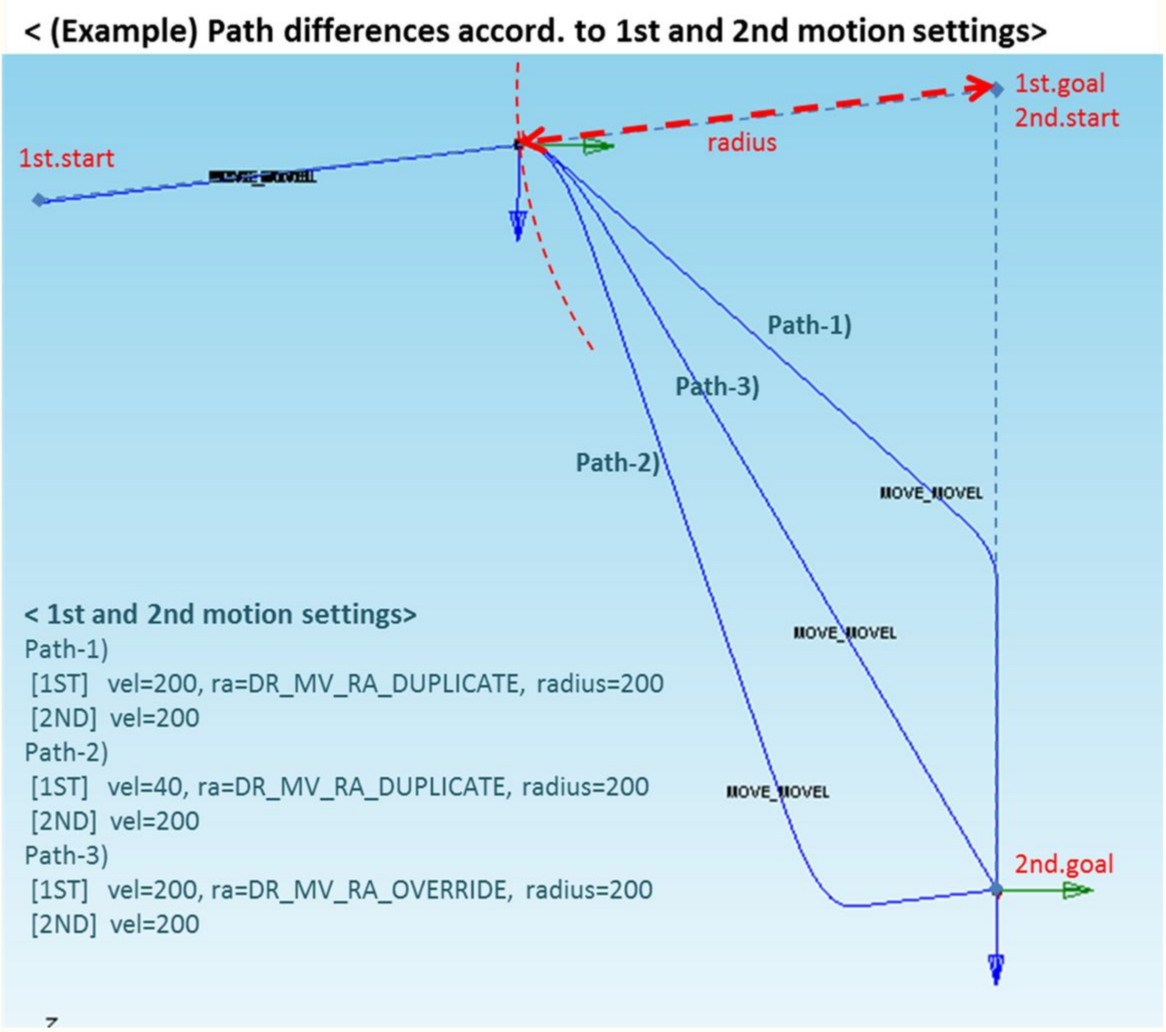
Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target joint location for six axes
fTargetVel	float	-	Velocity
fTargetAcc	float	-	Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
fBlendingRadius	float	0.f	Radius for blending
eBlendingType	enum.BLENDING_SPEED _TYPE	BLENDING_SPEED_ TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, fTargetVel and fTargetAcc are ignored, and the process is done based on fTargetTime .

Caution

If the following motion is blended with the conditions of eBlendingType being BLENDING_SPEED_TYPE_DUPLICATE and fBlendingRadius>0, the preceding motion can be terminated after the following motion is terminated first when the remaining motion time, which is determined by the remaining distance, velocity, and acceleration of the preceding motion, is greater than the motion time of the following motion. Refer to the following image for more information.



Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1
float q0[6] = { 0, 0, 90, 0, 90, 0 };
```

```
float jvel=10;
float jacc=20;
drfl.movej(q0, jvel, jacc);
# Move to the q0 joint angle with velocity 10(deg/sec) and acceleration 20(deg/sec2)

// CASE 2
float q0[6] = { 0, 0, 90, 0, 90, 0 };
float jTime=5;
drfl.movej(q0, 0, 0, jTime)
# Moves to the q0 joint angle with a reach time of 5 sec.

// CASE 3
float q0[6] = { 0, 0, 90, 0, 90, 0 };
float q1[6] = {90, 0, 90, 0, 90, 0 };
float jvel=10;
float jacc=20;
float blending_radius=50;
drfl.movej(q0, jvel, jacc, 0, MOVE_MODE_ABSOLUTE, blending_radius);
// Moves to the q0 joint angle and is set to execute the next motion
// when the distance from the location that corresponds to the q0 joint angle is 50 mm.
drfl.movej(q1, jvel, jacc, 0, MOVE_MODE_ABSOLUTE, 0, BLENDING_SPEED_TYPE_DUPLICATE));
    // blends with the last motion to move to the q1 joint angle.
```

4.6.2 CDRFLEx.movel

Features

This is a function for moving the robot along a straight line to the target position (pos) within the task space in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target TCP Position for six axes
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration

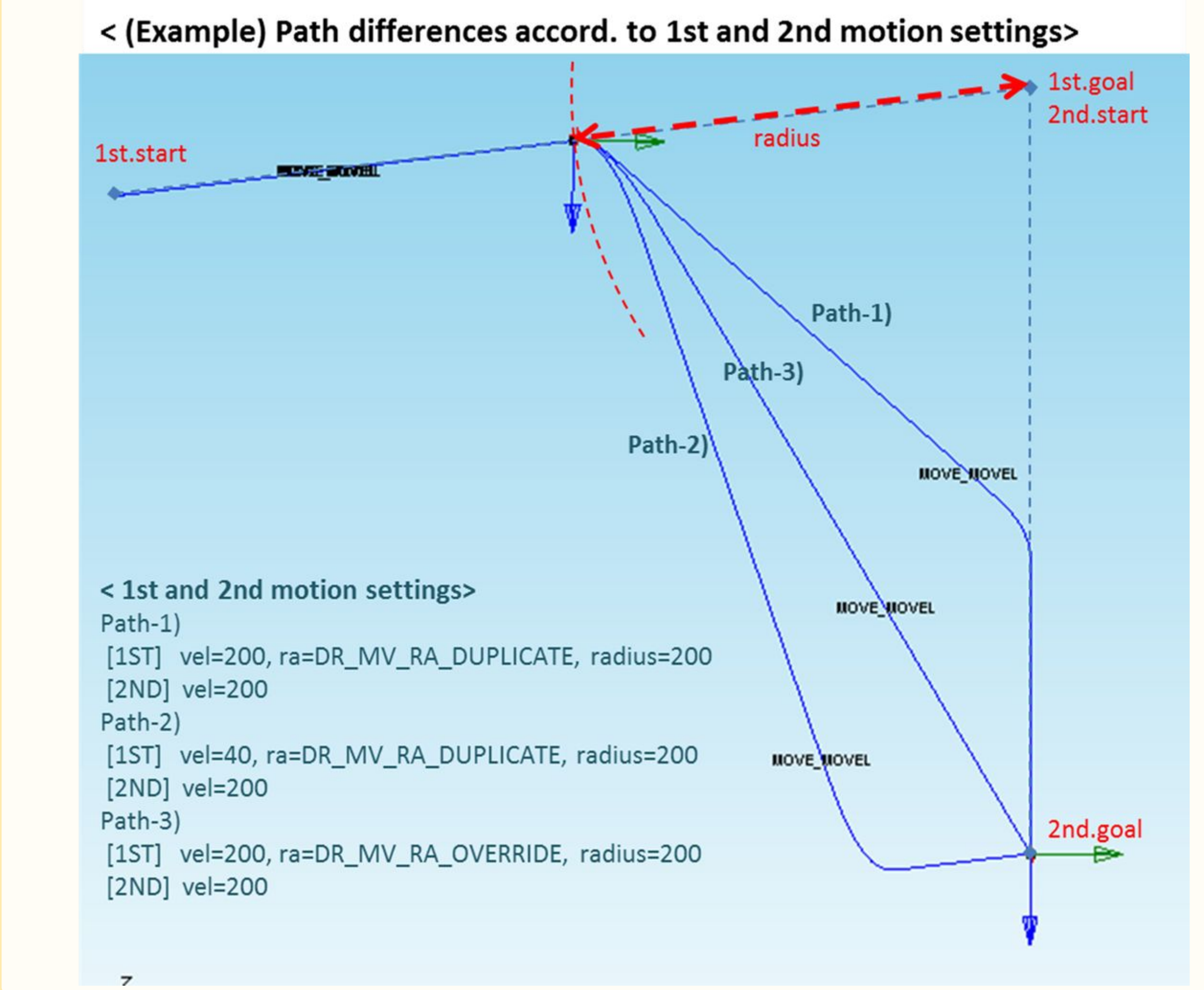
Parameter Name	Data Type	Default Value	Description
fTargetTime	float	0.f	Reach Time [sec] * If the time is specified, values are processed based on time, ignoring vel and acc.
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
fBlendingRadius	float	0.f	Radius for blending
eBlendingType	enum.BLENDING_SPEED_TYPE	BLENDING_SPEED_TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel ={30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc ={60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc .

Caution

If the following motion is blended with the conditions of eBlendingType being BLENDING_SPEED_TYPE_DUPLICATE and fBlendingRadius>0, the preceding motion can be terminated after the following motion is terminated first when the remaining motion time, which is determined by the remaining distance, velocity, and acceleration of the preceding motion, is greater than the motion time of the following motion. Refer to the following image for more information.



Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1
float x1[6] = { 559, 434.5, 651.5, 0, 180, 0 };
```

```
float tvel = { 50, 50 };
float tacc = { 100, 100 };
drfl.movel(x1, tvel, tacc);
// Moves to the x1 position with velocity 50(mm/sec) and acceleration 100(mm/sec2)

// CASE 2
float x1[6] = { 559, 434.5, 651.5, 0, 180, 0 };
float tTime = 5;
drfl.movel(x1, 0, 0, tTime);
// Moves to the x1 position with a reach time of 5 sec.

// CASE 3
float x1[6] = { 559, 434.5, 651.5, 0, 180, 0 };
float tvel = 50;
float tacc = 100;
drfl.movel(x1, tvel, tacc, 0, MOVE_MODE_RELATIVE, MOVE_REFERENCE_TOOL);
// Moves the robot from the start position to the relative position of x1 in the tool coordinate system
float x1[6] = { 559, 434.5, 651.5, 0, 180, 0 };
float x2[6] = { 559, 434.5, 251.5, 0, 180, 0 };
float tvel = 50;
float tacc = 100;
float blending_radius = 100;
drfl.movel(x1, tvel, tacc, 0, MOVE_MODE_ABSOLUTE, MOVE_REFERENCE_BASE, blending_radius);
```

4.6.3 CDRFLEx.movejx

Features

This is a function for moving the robot to the target position (pos) within the joint space in the robot controller. Since the target position is inputted as a posx form in the task space, it moves in the same way as Movel. However, since this robot motion is performed in the joint space, it does not guarantee a linear path to the target position. In addition, one of eight types of joint combinations (robot configurations) corresponding to the task space coordinate system (posx) must be specified in iSolutionSpace (solution space). When 255 is entered into iSolutionSpace, it moves to robot configuration that is the closest to the current configuration, (the smallest L2 norm in the joint space of axes 2-5) among the eight joint combination types.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target TCP Position for six axes

Parameter Name	Data Type	Default Value	Description
iSolutionSpace	unsigned char	-	joint combination shape (Refer to the below description)
fTargetVel	float/float[6]	-	Velocity
fTargetAcc	float/float[6]	-	Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
fBlendingRadius	Float	0.f	Radius for blending
eBlendingType	enum.BLENDING_SPEED_TYPE	BLENDING_SPEED_TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc.
- Using blending in the preceding motion generates an error in the case of input with relative motion (eMoveMode = MOVE_MODE_ABSOLUTE), and it is recommended to blend using movej() or movel().
- Refer to the description of movej() and movel() for blending according to option eBlendingType and fTargetVel / fTargetAcc.

Robot configuration (Shape vs. solution space)

Solution space	Binary	Shoulder	Elbow	Wrist
0	000	Lefty	Below	No Flip
1	001	Lefty	Below	Flip

Solution space	Binary	Shoulder	Elbow	Wrist
2	010	Lefty	Above	No Flip
3	011	Lefty	Above	Flip
4	100	Righty	Below	No Flip
5	101	Righty	Below	Flip
6	110	Righty	Above	No Flip
7	111	Righty	Above	Flip
255	Auto Calculation (the smallest L2 norm in the joint space of axes 2-5)			

Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1
float x1[6] = { 559, 34.5, 651.5, 0, 180, 0 };
float sol=2;
float jvel=10;
float jacc=20;
drfl.movejx(x1, sol, jvel, jacc);
        // Move to the joint angle that corresponds to x1 and
configuration with velocity 10(deg/sec)
        //and acceleration 20(deg/sec2).
// CASE 2
float x1[6] = { 559, 34.5, 651.5, 0, 180, 0 };
float sol=2;
float jTime=5;
drfl.movejx(x1, sol, 0, 0, jTime);
        // Moves to the joint angle that corresponds to x1 and
configuration with a reach time of 5 sec.
```

```
// CASE 3
float x1[6] = { 559, 34.5, 651.5, 0, 180, 0 };
float x2[6] = { 559, 434.5, 651.5, 0, 180, 0 };
float sol=2;
float jvel=10;
float jacc=20;
float blending_radius=50;
drfl.movejx(x1, sol, jvel, jacc, 0, MOVE_MODE_ABSOLUTE, blending_radius);
        // Moves to the joint angle that corresponds to x1 and
configuration and is set to execute the next motion
        // when the distance from x1 location is 50 mm.
drfl.movejx(x2, sol, jvel, jacc, 0, MOVE_MODE_ABSOLUTE, 0,
BLENDING_SPEED_TYPE_DUPLICATE);
        // blends with the last motion to move to the joint angle that
corresponds to x2 and configuration.
```

4.6.4 CDRFLEx.movec

Features

This is a function for moving the robot along an arc to the target position via a waypoint or to a specified angle from the current position based on the task space in the task space.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos[0]	float[6]	-	Waypoint
fTargetPos[1]	float[6]		Target location
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.0	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type

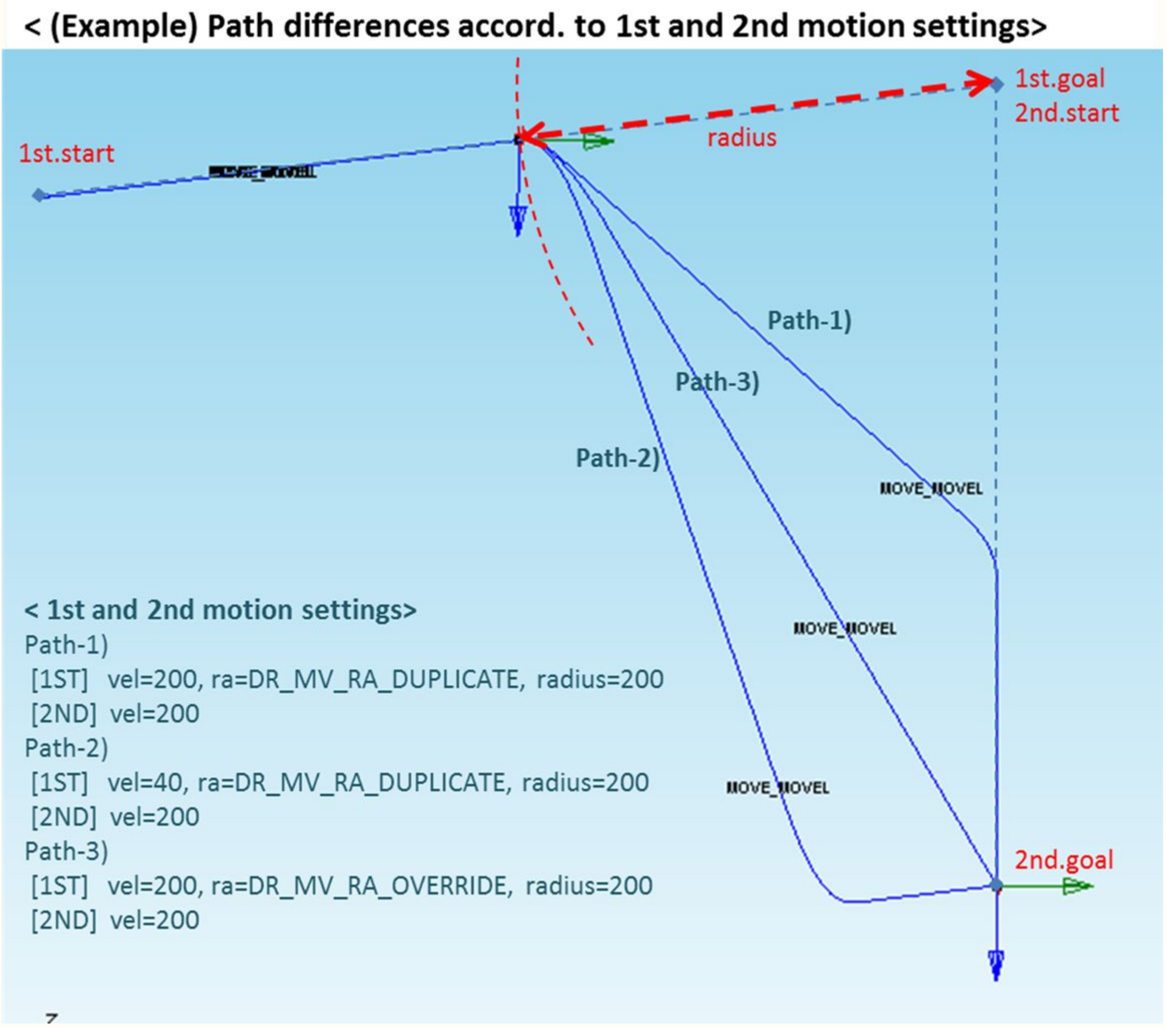
Parameter Name	Data Type	Default Value	Description
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
fTargetAngle1	float	0.0	angle1
fTargetAngle2	float	0.0	angle2
fBlendingRadius	float	0.0	Radius for blending
eBlendingType	enum.BLENDING_SPEED_TYPE	BLENDING_SPEED_TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel ={30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc ={60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc .
- If the eMoveMode is MOVE_MODE_RELATIVE, fTargetPos[0] and fTargetPos[1] are defined in the relative coordinate system of the previous position. (fTargetPos[0] is the relative coordinate from the starting point, while fTargetPos[1] is the relative coordinate from fTargetPos[0].)
- If fTargetAngle1 is more than 0, and fTargetAngle2 is equal to 0, the total rotated angle on the circular path is applied to fTargetAngle1.
- When fTargetAngle1 and fTargetAngle2 are more than 2, fTargetAngle1 refers to the total rotating angle moving at a constant velocity on the circular path, while fTargetAngle2 refers to the rotating angle in the rotating section for acceleration and deceleration. In that case, the total moving angle fTargetAngle1+ 2 X fTargetAngle2 moves along the circular path.

Caution

If the following motion is blended with the conditions of eBlendingType being BLENDING_SPEED_TYPE_DUPLICATE and fBlendingRadius>0, the preceding motion can be terminated after the following motion is terminated first when the remaining motion time, which is determined by the remaining distance, velocity, and acceleration of the preceding motion, is greater than the motion time of the following motion. Refer to the following image for more information.



Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1
float x1[2][6] = {{559,434.5,651.5,0,180,0}, {559,434.5,251.5,0,180,0}};
float tvel = {50,50}; # Set the task velocity to 50(mm/sec) and 50(deg/sec).
float tacc = {100,100}; # Set the task acceleration to 100(mm/sec2) and 100(deg/sec2).
```

```
drfl.movec(x1, tvel, tacc);
// Moves to x1[1] with a velocity of 50(mm/sec) and acceleration of
100(mm/sec2)
// via x1[0] along the arc trajectory.

// CASE 2
float x1[2][6] = {{559,434.5,651.5,0,180,0},{559,434.5,251.5,0,180,0}};
float tTime = 5;
drfl.movec(x1, 0, 0, tTime);
// Moves along the arc trajectory to x1[1] via x1[0] with a reach time of
5 seconds

// CASE 3
float x1[2][6] = {{559,434.5,651.5,0,180,0},{559,434.5,251.5,0,180,0}};
float x2[2][6] = {{559,234.5,251.5,0,180,0 },{559,234.5,451.5,0,180,0}};
float tvel = {50,50};
float tacc = {100,100};
float blending_radius = 50;
drfl.movec(x1, tvel, tacc, 0, MOVE_MODE_ABSOLUTE, MOVE_REFERENCE_BASE, 0,
0, blending_radius);
drfl.movec(x2, tvel, tacc, 0, MOVE_MODE_ABSOLUTE, MOVE_REFERENCE_BASE, 0,
0, 0, BLENDING_SPEED_TYPE_DUPLICATE);
```

4.6.5 CDRFLEx.movesj

Features

This is a function for moving the robot along a spline curve path that connects the current position to the target position (the last waypoint) via the waypoints of the joint space. The input velocity/acceleration means the maximum velocity/acceleration in the path, and the acceleration and deceleration during the motion are determined according to the position of the waypoint.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	Float[MAX_SPLINE_POINT][6]	-	Maximum 100 waypoint list
nPosCount	unsigned char	-	Number of valid waypoints
fTargetVel	float/float[6]	-	Velocity
fTargetAcc	float/float[6]	-	Acceleration
fTargetTime	float	0.f	Reach Time [sec]

Parameter Name	Data Type	Default Value	Description
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, values are processed based on fTargetTime, ignoringfTargetVel and fTargetAcc.
- When eMoveMode is MOVE_MODE_RELATIVE, each pos of the position list is defined as a relative coordinate for the preceding pos. (If position list=[q1, q2, ...,q(n-1), q(n)], q1 is the relative angle of the starting point, while q(n) is the relative coordinate of pq(n-1))
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1 : Absolute coordinate input (mod=MOVE_MODE_ABSOLUTE)
float jpos[4][6];
float jvel=10;
float jacc=10;
int jposNum = 4;
jpos[0][0]=0; jpos[0][1]=0; jpos[0][2]=-30; jpos[0][3]=0; jpos[0][4]=-30; jpos[0][5]=0;
jpos[1][0]=90; jpos[1][1]=0; jpos[1][2]=0; jpos[1][3]=0; jpos[1][4]=0; jpos[1][5]=0;
jpos[2][0]=0; jpos[2][1]=0; jpos[2][2]=-30; jpos[2][3]=0; jpos[2][4]=-30; jpos[2][5]=0;
jpos[3][0]=-90; jpos[3][1]=0; jpos[3][2]=0; jpos[3][3]=0; jpos[3][4]=0; jpos[3][5] = 0;
drfl.movesj(jpos, jposNum, jvel, jacc);
// Moves the spline curve that connects the absolute waypoints defined in the jpos
// with a maximum velocity of 10(deg/sec) and maximum acceleration of 10(deg/sec2)

// CASE 2 : Relative angle input (mod=MOVE_MODE_RELATIVE)
float jpos[4][6];
```

```
float jvel=10;
float jacc=10;
int jposNum = 4;
jpos[0][0]=0; jpos[0][1]=0; jpos[0][2]=-30; jpos[0][3]=0; jpos[0][4]=-30;
jpos[0][5]=0;
// Start Position + jpos[0]
jpos[1][0]=90;  jpos[1][1]=0; jpos[1][2]=30; jpos[1][3]=0; jpos[1][4]=30;
jpos[1][5]=0;
// Start Position + jpos[0] + jpos[1]
jpos[2][0]=-90; jpos[2][1]=0; jpos[2][2]=-30; jpos[2][3]=0; jpos[2]
[4]=-30; jpos[2][5]=0;
// Start Position + jpos[0] + jpos[1] + jpos[2]
jpos [3][0]=-90; jpos [3][1]=0; jpos[3][2]=30; jpos[3][3]=0; jpos[3]
[4]=30; jpos[3][5]=0;
// Start Position + jpos[0] + jpos[1] + jpos[2] + jpos[3]
drfl.movesj(jpos, jposNum, jvel, jacc, time, MOVE_MODE_RELATIVE);
// Moves the spline curve that connects the relative waypoints defined in
the jpos
// with a maximum velocity of 10(deg/sec) and maximum acceleration of
10(deg/sec2)
```

4.6.6 CDRFLEx.movesx

Features

This is a function for moving the robot along a spline curve path that connects the current position to the target position (the last waypoint) via the waypoints of the task space. The input velocity/acceleration means the maximum velocity/acceleration in the path, and the constant velocity motion is performed with the input velocity according to the condition if the option for constant speed motion is selected.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float [MAX_SPLINE_POINT][6]	-	Maximum 100 waypoint information
nPosCount	unsigned char	-	Number of valid waypoint
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.f	Reach Time [sec]

Parameter Name	Data Type	Default Value	Description
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMove Reference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
eVelOpt	enum.SPLINE_VELOCITY_OPTION	SPLINE_VELOCITY_OPTION_DEFAULT	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel ={30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc = {60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc .
- When eMoveMode is MOVE_MODE_RELATIVE, each pos of the position list is defined as a relative coordinate for the preceding pos. (If position list=[q1, q2, ...,q(n-1), q(n)], q1 is the relative angle of the starting point, while q(n) is the relative coordinate of pq(n-1))
- This function does not support online blending of previous and subsequent motions.

Caution

The constant velocity motion according to the distance and velocity between the input waypoints cannot be used if the "SPLINE_VELOCITY_OPTION_CONST" option (constant velocity option) is selected for eVelOpt, and the motion is automatically switched to the variable velocity motion (eVelOpt =SPLINE_VELOCITY_OPTION_DEFAULT) in that case.

Return

Value	Description
0	Error
1	Success

Example

```
// CASE 1 : Moves absolute coordinate standard (mod= MOVE_MODE_ABSOLUTE)
float xpos[4][6];
int xposNum = 4;
float tvel={ 50, 100 };
float tacc={ 50, 100 };
xpos[0][0]=559; xpos[0][1]=434.5; xpos[0][2]=651.5;
xpos[0][3]=0; xpos[0][4]=180; xpos[0][5]=0;
xpos[1][0]=559; xpos[1][1]=434.5; xpos[1][2]=251.5;
xpos[1][3]=0; xpos[1][4]=180; xpos[1][5]=0;
xpos[2][0]=559; xpos[2][1]=234.5; xpos[2][2]=251.5;
xpos[2][3]=0; xpos[2][4]=180; xpos[2][5]=0;
xpos[3][0]=559; xpos[3][1]= 234.5; xpos[3][2]=451.5;
xpos[3][3]=0; xpos[3][4]=180; xpos[3][5]=0;
drfl.movesx(xpos, xposNum, tvel, tacc, 0, MOVE_MODE_ABSOLUTE);
     // Moves the spline curve that connects the waypoints defined in the xpos
     // with a maximum velocity of 50, 50(mm/sec, deg/sec) and maximum acceleration of 100, 100(mm/sec2,
     // deg/sec2)

// CASE 2 : Moves relative coordinate standard (mod= MOVE_MODE_RELATIVE)
float xpos[4][6];
int xposNum = 4;
float tvel={ 50, 100 };
float tacc={ 50, 100 };
xpos[0][0]=0; xpos[0][1]=400; xpos[0][2]=0;
xpos[0][3]=0; xpos[0][4]=0; xpos[0][5]=0;
// Homogeneous transformation of xpos[0] from the start position → x0
xpos[1][0]=0; xpos[1][1]=0; xpos[1][2]=-400;
xpos[1][3]=0; xpos[1][4]=0; xpos[1][5]=0;
// Homogeneous transformation of xpos[1] from x0 → x1
xpos[2][0]=0; xpos[2][1]=-200; xpos[2][2]=0;
xpos[2][3]=0; xpos[2][4]=0; xpos[2][5]=0;
// Homogeneous transformation of xpos[2] from x1 → x2
xpos[3][0]=0; xpos[3][1]=0; xpos[3][2]=200;
xpos[3][3]=0; xpos[3][4]=0; xpos[3][5]=0;
// Homogeneous transformation of xpos[3] from x2 → x3
drfl.movesx(xpos, xposNum, tvel, tacc, 0, MOVE_MODE_RELATIVE);
    // Moves the spline curve that connects the waypoints defined in the xpos
// with a maximum velocity of 50, 50(mm/sec, deg/sec) and maximum acceleration of 100,
// 100(mm/sec2, deg/sec2)
```

4.6.7 CDRFLEx.moveb

Features

This is a function for moving the robot at constant velocity by blending the robot with the blending radius set in the path information after receiving path information consisting of one or more lines or arcs.

Parameter

Parameter Name	Data Type	Default Value	Description
tTargetPos	struct MOVE_POSB [MAX_MOVEB_POINT]	-	Maximum 25 path information
nPosCount	unsigned char	-	Number of valid paths
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to Constant and Enumeration Type Constant
eMoveReference	enum.MOVE_REFERENC E	MOVE_REFERENCE_BAS E	Refer to Constant and Enumeration Type Constant

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel = {30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc = {60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc .
- When eMoveMode is MOVE_MODE_RELATIVE, each pos of the posb list is defined as a relative coordinate for the preceding pos.

Caution

- A user input error is generated if the blending radius in posb is 0.
- A user input error is generated due to the duplicated input of Line if contiguous Line-Line segments have the same direction.
- If the blending condition causes a rapid change in direction, a user input error is generated to prevent sudden acceleration.
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
MOVE_POSB xb[4];
memset(xb, 0x00, sizeof(xb));
int segNum = 4;
float tvel = { 50, 50 };
float tacc = { 100, 100 };
xb[0]._iBlendType = 0;  // line
xb[0]._fBlendRad  = 50;
xb[0]._fTargetPos[0][0] = 559;
xb[0]._fTargetPos[0][1] = 234.5;
xb[0]._fTargetPos[0][2] = 651.5;
xb[0]._fTargetPos[0][3] = 0;
xb[0]._fTargetPos[0][4] = 180;
xb[0]._fTargetPos[0][5] = 0;
xb[1]._iBlendType = 1;  // circle
xb[1]._fBlendRad  = 50;
xb[1]._fTargetPos[0][0] = 559;
xb[1]._fTargetPos[0][1] = 234.5;
xb[1]._fTargetPos[0][2] = 451.5;
xb[1]._fTargetPos[0][3] = 0;
xb[1]._fTargetPos[0][4] = 180;
xb[1]._fTargetPos[0][5] = 0;
xb[1]._fTargetPos[1][0] = 559;
xb[1]._fTargetPos[1][1] = 434.5;
xb[1]._fTargetPos[1][2] = 451.5;
```

```
xb[1]._fTargetPos[1][3] = 0;
xb[1]._fTargetPos[1][4] = 180;
xb[1]._fTargetPos[1][5] = 0;
xb[2]._iBlendType = 0;  // line
xb[2]._fBlendRad = 50;
xb[2]._fTargetPos[0][0] = 559;
xb[2]._fTargetPos[0][1] = 434.5;
xb[2]._fTargetPos[0][2] = 251.5;
xb[2]._fTargetPos[0][3] = 0;
xb[2]._fTargetPos[0][4] = 180;
xb[2]._fTargetPos[0][5] = 0;
xb[3]._iBlendType = 0;  // line
xb[3]._fBlendRad  = 50;
xb[3]._fTargetPos[0][0] = 559;
xb[3]._fTargetPos[0][1] = 234.5;
xb[3]._fTargetPos[0][2] = 251.5;
xb[3]._fTargetPos[0][3] = 0;
xb[3]._fTargetPos[0][4] = 180;
xb[3]._fTargetPos[0][5] = 0;
drfl.moveb(xb, segNum, tvel, tacc);
    // Moves the robot from the current position through a trajectory consisting of LINE→CIRCLE→LINE→LINE,
// maintaining velocity 50, 50(mm/sec, deg/sec) and acceleration 100, 100(mm/sec2, deg/sec2).
// Blending to the next segment begins
    // when the distance of 50 mm from the end point of each segment is reached.
```

4.6.8 CDRFLEx.move_spiral

Features

This is a function for the radius increasing in a radial direction and the robot's moving in parallel with the rotating spiral motion in an axial direction. It moves the robot along the spiral trajectory on the surface that is perpendicular to the axis on the coordinate specified as eMoveReferenceand the linear trajectory in the axis direction.

Parameter

Parameter Name	Data Type	Default Value	Range	Description
eTaskAxis	enum.TASK_AXIS	-	-	Refer to the Definition of Constant and Enumeration Type
fRevolution	float	-	rev > 0	Total number of revolutions [revolution]

Parameter Name	Data Type	Default Value	Range	Description
fMaximuRadius	float	-	rmax > 0	Final spiral radius [mm]
fMaximumLength	float	-		Distance moved in the axis direction [mm]
fTargetVel	float[2]	-		Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-		Linear Acceleration, Angular Acceleration
fTargetTime	float	0.0	time ≥ 0	Total execution time <sec>
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_TOOL		Refer to the Definition of Constant and Enumeration Type

Note

- fRevolution refers to the total number of revolutions of the spiral motion.
- fMaximuRadius refers to the maximum radius of the spiral motion.
- fMaximumLength refers to the parallel distance in the axis direction during the motion. A negative value means the parallel distance in the axis direction.
- fTargetVel refers to the moving velocity of the spiral motion.
- fTargetAcc refers to the moving acceleration of the spiral motion.
- When fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc.
- eTaskAxis defines the axis that is perpendicular to the surface defined by the spiral motion.
- eMoveReference refers to the reference coordinate system defined by the spiral motion.
- This function does not support online blending of previous and subsequent motions.

Caution

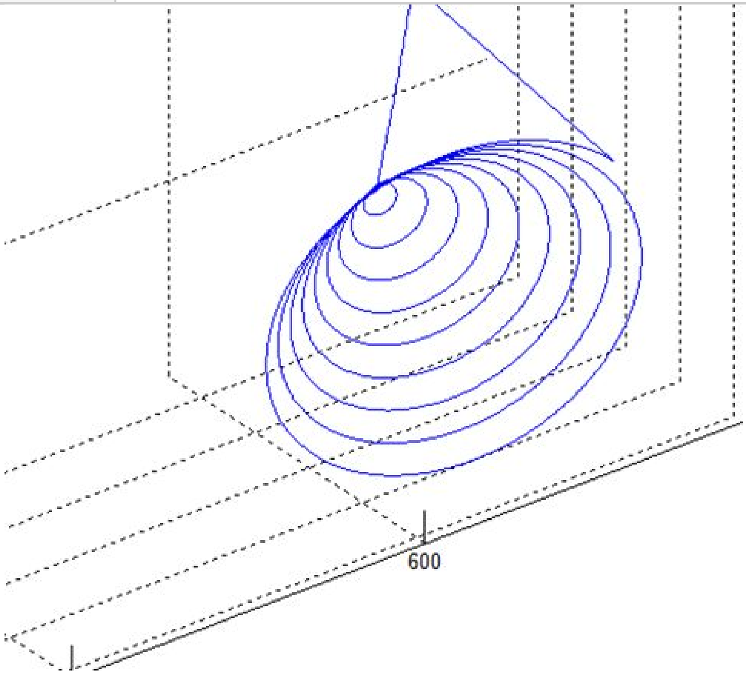
- If the rotating acceleration calculated by the spiral path is too great. an error can be generated to ensure safe motion. In this case, reduce fTargetVel, fTargetAcc or fTargetTime value.

Return

Value	Description
0	Error
1	Success

Example

```
float rev = 3;
float rmax = 50;
float lmax = 50;
float tvel = { 50, 50 };
float tacc = { 100, 100 };
Drfl.move_spiral(TASK_AXIS_Z, rev, rmax, lmax, tvel, tacc);
// Moves the robot from the current position maintaining
// velocity 50, 50(mm/sec, deg/sec) and acceleration 100, 100(mm/sec2, deg/sec2) for spiral trajectory to the maximum radius, 50 mm and
// moves in the direction of Tool z by 50 mm at the same time.
```



4.6.9 CDRFLEx.move_periodic

Features

This command performs a cyclic motion based on the sine function of each axis (parallel and rotation) of the reference coordinate (eMoveReference) input as a relative motion that begins at the current position. The attributes of the motion on each axis are determined by fAmplitude and fPeriodic, and the acceleration/deceleration time and the total motion time are set by the interval and repetition count.

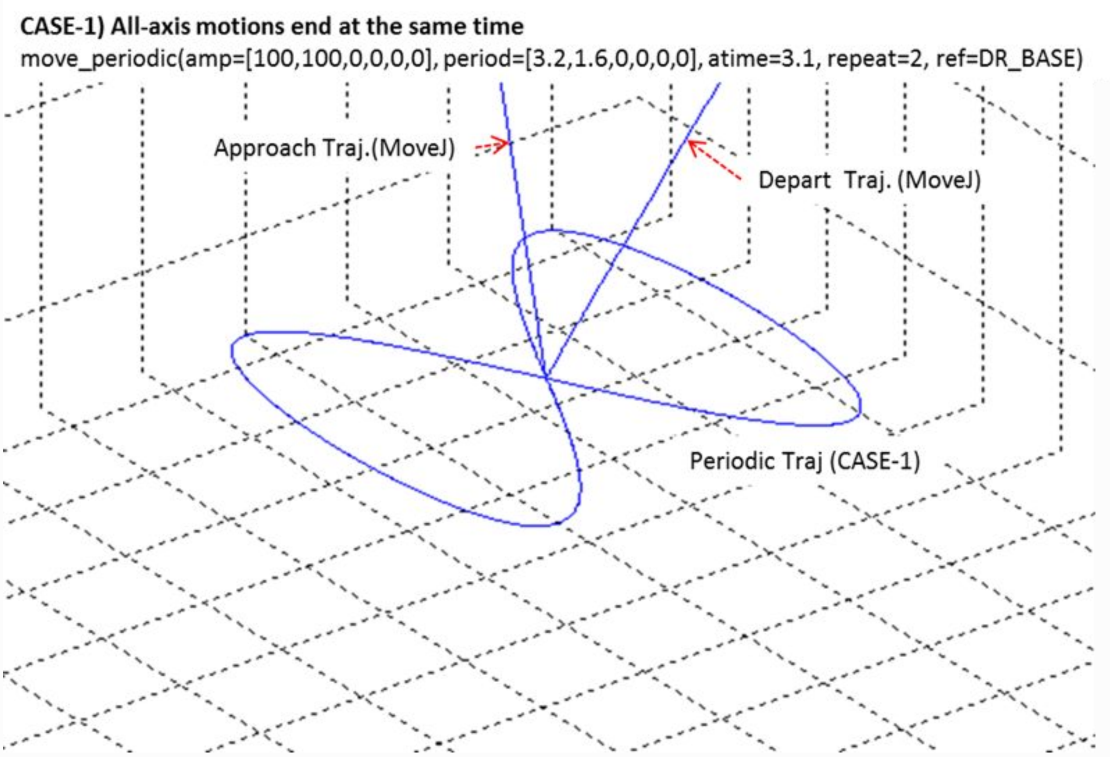
Parameter

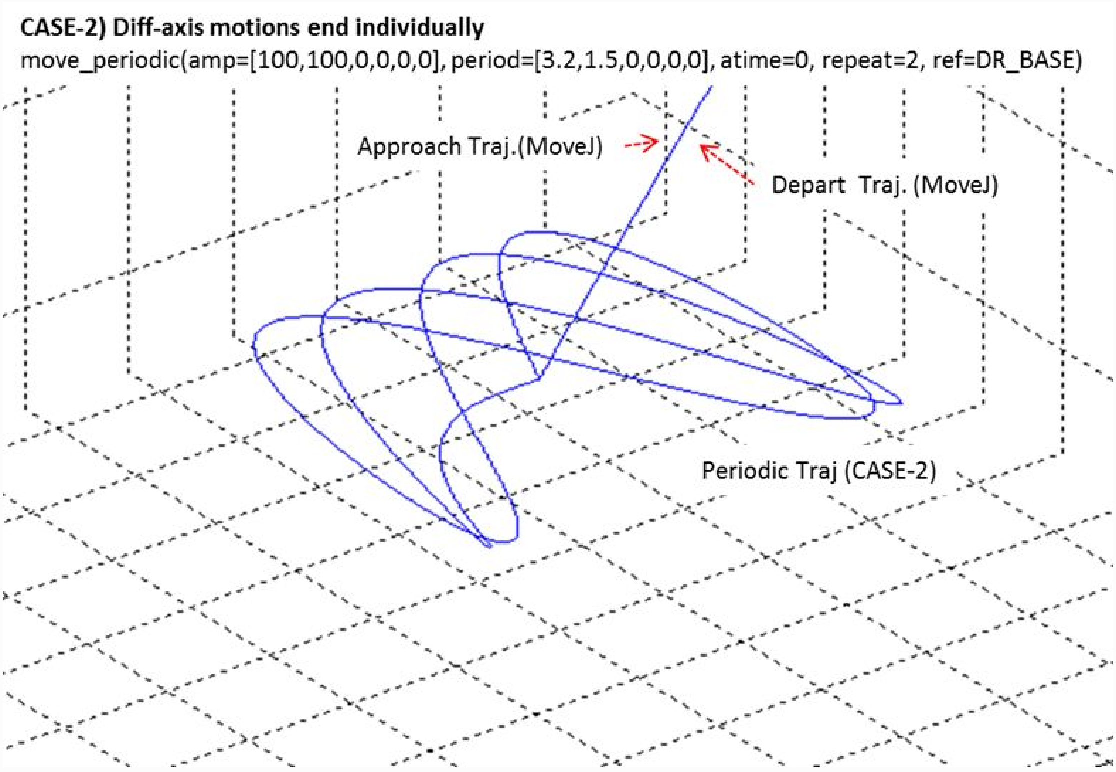
Parameter Name	Data Type	Default Value	Range	Description
fAmplitude	float[6]	-	>=0	Amplitude (motion between -amp and +amp) [mm] or [deg]
fPeriodic	float[6]	-	>=0	period(time for one cycle) [sec]
fAccelTime	float	-	>=0	Acc-, dec- time [sec]
nRepeat	unsigned char	-	> 0	Repetition count
eMoveReference	enum	MOVE_REFERENCE_TOOL		Refer to the Definition of Constant and Enumeration Type

Note

- fAmplitude refers to the amplitude. The input is a list of six elements that are the amp values for the axes (x, y, z, rx, ry, and rz). The amp input on the axis that does not have a motion must be 0.
- fPeriodic refers to the time needed to complete a motion in the direction, the amplitude. The input is a list of six elements that are the periods for the axes (x, y, z, rx, ry, and rz) or representative value.
- fAccelTime Atime refers to the acceleration and deceleration time at the beginning and end of the periodic motion. The largest of the inputted acceleration/deceleration times and maximum period*1/4 is applied. An error is generated when the inputted acceleration/deceleration time exceeds 1/2 of the total motion time.
- nRepeat refers to the number of repetitions of the axis (reference axis) that has the largest period value and determines the total motion time. The number of repetitions for each of the remaining axes is determined automatically according to the motion time. If the motion terminates normally, the motions for the remaining axes can be terminated before the

reference axis's motion terminates so that the end position matches the starting position. If the motions of all axes are not terminated at the same time, the deceleration section will deviate from the previous path. Refer to the following image for more information.





- eMoveReference refers to the reference coordinate system of the repeated motion.

- If a maximum velocity error is generated during a motion, adjust the amplification and period using the following formula.
  velocity=Amplification (amp)*2*pi(3.14)/Period(period)
  (i.e., Max. velocity=62.83 mm/sec if amp=10 mm and Period=1 sec)
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
<#1>
float fAmplitude[NUM_TASK] = {10,0,0,0,30,0};
float fPeriod[NUM_TASK] = { 1, 1, 1, 1, 1, 1};
drfl.amove_periodic(fAmplitude, fPeriod, 0.2, 5, MOVE_REFERENCE_TOOL);
    # Repeats the x-axis (10 mm amp and 1 sec. period) motion and y
rotating axis (30 deg amp and 1 sec. period) motion in the tool coordinate
system
    # five times.
<#2>
float fAmplitude[NUM_TASK] = {10,0,20,0,0.5,0};
float fPeriod[NUM_TASK] = { 1,0,1.5,0,0,0};
drfl.amove_periodic(fAmplitude, fPeriod, 0.5, 3, MOVE_REFERENCE_BASE);
    # Repeats the x-axis (10 mm amp and 1 sec. period) motion and z axis
(20 mm amp and 1.5 sec. period) motion in the base coordinate system
    # three times. The y rotating motion is not executed, as the period is
“zero(0).”
    # As the z-axis period is large, the total motion time is about 5.5
sec. (1.5 sec.*3 times + acceleration/deceleration 1 sec.)
    #, x-axis motion repeats 4.5 times
```

4.6.10 CDRFLEx.amovej

Features

As an asynchronous movej, it operates the way same as the movej function except for not having the fBlendingRadius argument for blending. However, the command returns with motion start at the same time and executes the next line without waiting for the termination of motion due to the characteristic of the asynchronous type.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target joint location for six axes
fTargetVel	float/float[6]	-	Velocity
fTargetAcc	float/float[6]	-	Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eBlendingType	enum.BLENDING_SPEED _TYPE	BLENDING_SPEED_ TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, values are processed based on fTargetTime ignoring fTargetVel and
- Refer to the motion description of movej() for blending according to option eBlendingType and fTargetVel / fTargetAcc.

Return

Value	Description
0	Error
1	Success

Example

```
float q0[6] = { 0, 0, 90, 0, 90, 0 };
float q1[6] = { 90, 0, 90, 0, 90, 0 };
float q99[6] = { 0, 0, 0, 0, 0, 0 };
float jvel=10;
float jacc=20;
drfl.amovej(q0, jvel, jacc);
Sleep(3000);

Drfl.amovej(q1, jvel, jacc);
drfl.mwait(); // Temporarily suspends the program execution until the motion is terminated

Drfl.movej(q99, jvel, jacc);
```

4.6.11 CDRFLEx.amovel

Features

As an asynchronous movel, it operates the same as the movel function except for not having the fBlendingRadius argument for blending. However, the command returns with motion start at the same time and executes the next line without waiting for the termination of motion due to the characteristic of the asynchronous type.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target TCP Position for six axes
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	-	Reach Time [sec] * If the time is specified, values are processed based on time, ignoring vel and acc.
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type

Parameter Name	Data Type	Default Value	Description
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
eBlendingType	enum.BLENDING_SPEED_TYPE	BLENDING_SPEED_TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel = {30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc = {60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc.
- Refer to the motion description of movel() for blending according to option eBlendingType and fTargetVel / fTargetAcc.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 2 seconds after the motion starts with x1
float x1[6] = { 559, 434.5, 651.5, 0, 180, 0 };
float tvel = { 50, 50 };
float tacc = { 100, 100 };
drfl.amovel(x1, tvel, tacc);
Sleep(2000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.6.12 CDRFLEx.amovec

Features

As an asynchronous movec, it operates the same as movec except for not having the fBlendingRadius argument for blending. However, the command returns with motion start at the same time and executes the next line without waiting for the termination of motion due to the characteristic of the asynchronous type.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos[0]	float[6]	-	Waypoint
fTargetPos[1]	float[6]		Target location
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type
fTargetAngle2	float	0.f	angle1
fTargetAngle2	float	0.f	angle2
eBlendingType	enum.BLENDING_SPEED_TYPE	BLENDING_SPEED_TYPE_DUPLICATE	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel = {30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc = {60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc.
- If the eMoveMode is MOVE_MODE_RELATIVE, fTargetPos[0] and fTargetPos[1] are defined in the relative coordinate system of the previous position. (fTargetPos[0] is the relative coordinate from the starting point, while fTargetPos[1] is the relative coordinate from fTargetPos[0].)
- If fTargetAngle1 is more than 0, and fTargetAngle2 is equal to 0, the total rotated angle on the circular path is applied to fTargetAngle1.
- When fTargetAngle1 and fTargetAngle2 are more than 2, fTargetAngle1 refers to the total rotating angle moving at a constant velocity on the circular path, while fTargetAngle2 refers to the rotating angle in the rotating section for acceleration and deceleration. In that case, the total moving angle fTargetAngle1+ 2 X fTargetAngle2 moves along the circular path.
- Refer to the motion description of movej() for blending according to option eBlendingType and fTargetVel / fTargetAcc.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the circle motion through two points of x1 begins
float x1[2][6] = { { 559, 434.5, 651.5, 0, 180, 0 }, { 559, 434.5, 251.5, 0, 180, 0 } };
float tvel = {50, 50}; # Set the task velocity to 50(mm/sec) and 50(deg/sec).
float tacc = {100, 100}; # Set the task acceleration to 100(mm/sec2) and 100(deg/sec2).
drfl.amovec(x1, tvel, tacc);
Sleep(3000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.6.13 CDRFLEx.amovesj

Features

As an asynchronous movesj, it operates the same as movesj() except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion. A new motion command generated before the motion is terminated by amovesj() causes errors for security reasons. Therefore, the termination of the amovesj() motion must be confirmed using mwait() between amovesj() and the following motion command before the new motion command is executed.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	Float[MAX_SPLINE_POINT][6]	-	Maximum 100 waypoint list
nPosCount	unsigned char	-	Number of valid waypoints
fTargetVel	float/float[6]	-	Velocity
fTargetAcc	float/float[6]	-	Acceleration
fTargetTime	float	0.0	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ABSOLUTE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and
- When eMoveMode is MOVE_MODE_RELATIVE, each pos on the position list is defined as a relative coordinate for the preceding pos. (If position list=[q1, q2, ...,q(n-1), q(n)], q1 is the relative angle of the starting point, while q(n) is the relative coordinate of q(n-1))
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the spline motion through all points of jpos begins
float jpos[4][6];
float jvel=10;
float jacc=10;
int jposNum = 4;
jpos[0][0]=0; jpos[0][1]=0; jpos[0][2]=-30; jpos[0][3]=0; jpos[0][4]=-30; jpos[0][5]=0;
jpos[1][0]=90; jpos[1][1]=0; jpos[1][2]=0; jpos[1][3]=0; jpos[1][4]=0; jpos[1][5]=0;
jpos[2][0]=0; jpos[2][1]=0; jpos[2][2]=-30; jpos[2][3]=0; jpos[2][4]=-30; jpos[2][5]=0;
jpos[3][0]=-90; jpos[3][1]=0; jpos[3][2]=0; jpos[3][3]=0; jpos[3][4]=0; jpos[3][5] = 0;

drfl.movesj(jpos, jposNum, jvel, jacc);
Sleep(3000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.6.14 CDRFLEx.amovesx

Features

As an asynchronous movesx, it operates the same as movesx() except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion. A new motion command generated before the motion is terminated by amovesj() causes errors for security reasons. Therefore, the termination of the amovesx() motion must be confirmed using mwait() between the amovesx function () and the following motion command before the new motion command is executed.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float [MAX_SPLINE_ POINT][6]	-	Maximum 100 waypoint information
nPosCount	unsigned char	-	Number of valid waypoints
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.0	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMove Reference	enum.MOVE_REFERENC E	MOVE_REFERENCE_BAS E	Refer to the Definition of Constant and Enumeration Type
eVelOpt	enum.SPLINE_VELOCITY _OPTION	SPLINE_VELOCITY_OPTI ON_DEFAULT	Refer to the Definition of Constant and Enumeration Type

Note

- If an argument is inputted to fTargetVel (e.g., fTargetVel = {30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc = {60, 0}), the input argument corresponds to the linear acceleration of the motion, while the angular acceleration is determined proportionally to the linear acceleration.
- If fTargetTime is specified, values are processed based on fTargetTime, ignoring fTargetVel and fTargetAcc .
- When eMoveMode is MOVE_MODE_RELATIVE, each pos on the position list is defined as a relative coordinate for the preceding pos. (If position list=[q1, q2, ...,q(n-1), q(n)], q1 is the relative angle of the starting point, while q(n) is the relative coordinate of q(n-1))
- This function does not support online blending of previous and subsequent motions.

> **Caution**
>
> The constant velocity motion according to the distance and velocity between the input waypoints cannot be used if the "SPLINE_VELOCITY_OPTION_CONST" option (constant velocity option) is selected for eVelOpt, and the motion is automatically switched to the variable velocity motion (eVelOpt =SPLINE_VELOCITY_OPTION_DEFAULT) in that case.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the spline motion through all points of xpos begins
float xpos[4][6];
int xposNum = 4;
float tvel={ 50, 100 };
float tacc={ 50, 100 };
xpos[0][0]=559; xpos[0][1]=434.5; xpos[0][2]=651.5;
xpos[0][3]=0; xpos[0][4]=180; xpos[0][5]=0;
xpos[1][0]=559; xpos[1][1]=434.5; xpos[1][2]=251.5;
xpos[1][3]=0; xpos[1][4]=180; xpos[1][5]=0;
xpos[2][0]=559; xpos[2][1]=234.5; xpos[2][2]=251.5;
xpos[2][3]=0; xpos[2][4]=180; xpos[2][5]=0;
xpos[3][0]=559; xpos[3][1]= 234.5; xpos[3][2]=451.5;
xpos[3][3]=0; xpos[3][4]=180; xpos[3][5]=0;
drfl.amovesx(xpos, xposNum, tvel, tacc, 0, MOVE_MODE_ABSOLUTE);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.6.15 CDRFLEx.amoveb

Features

As an asynchronous moveb, it operates the same as the moveb() function except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion. A new motion command generated before the motion is terminated by amovesj() function causes errors for security reasons. Therefore, the termination of the amoveb() motion must be confirmed using the mwait() function between amovesj() and the following motion command before the new motion command is executed.

Parameter

Parameter Name	Data Type	Default Value	Description
tTargetPos	struct MOVE_POSB [MAX_MOVEB_POINT]	-	Maximum 25 path information
nPosCount	unsigned char	-	Number of valid paths
fTargetVel	float[2]	-	Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-	Linear Acceleration, Angular Acceleration
fTargetTime	float	0.f	Reach Time [sec]
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_BASE	Refer to the Definition of Constant and Enumeration Type

Note

- When fTargetTime is specified, values are processed based on fTargetTime ignoring fTargetVel and
- When eMoveMode is MOVE_MODE_RELATIVE, each pos on the posb list is defined as a relative coordinate for the preceding pos.
- This function does not support online blending of previous and subsequent motions.

Caution

- A user input error is generated if the blending radius in tTargetPos is 0.
- A user input error is generated due to the duplicated input of Line if contiguous Line-Line segments have the same direction.
- If the blending condition causes a rapid change in direction, a user input error is generated to prevent sudden acceleration.
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the motion through all paths of xb begins
MOVE_POSB xb[4];
memset(xb, 0x00, sizeof(xb));
int segNum = 4;
float tvel = { 50, 50 };
float tacc = { 100, 100 };
xb[0]._iBlendType = 0;  // line
xb[0]._fBlendRad  = 50;
xb[0]._fTargetPos[0][0] = 559;
xb[0]._fTargetPos[0][1] = 234.5;
xb[0]._fTargetPos[0][2] = 651.5;
xb[0]._fTargetPos[0][3] = 0;
xb[0]._fTargetPos[0][4] = 180;
xb[0]._fTargetPos[0][5] = 0;
xb[1]._iBlendType = 1;  // circle
xb[1]._fBlendRad  = 50;
xb[1]._fTargetPos[0][0] = 559;
xb[1]._fTargetPos[0][1] = 234.5;
xb[1]._fTargetPos[0][2] = 451.5;
xb[1]._fTargetPos[0][3] = 0;
xb[1]._fTargetPos[0][4] = 180;
xb[1]._fTargetPos[0][5] = 0;
xb[1]._fTargetPos[1][0] = 559;
xb[1]._fTargetPos[1][1] = 434.5;
xb[1]._fTargetPos[1][2] = 451.5;
xb[1]._fTargetPos[1][3] = 0;
xb[1]._fTargetPos[1][4] = 180;
xb[1]._fTargetPos[1][5] = 0;
xb[2]._iBlendType = 0;  // line
xb[2]._fBlendRad = 50;
xb[2]._fTargetPos[0][0] = 559;
xb[2]._fTargetPos[0][1] = 434.5;
xb[2]._fTargetPos[0][2] = 251.5;
xb[2]._fTargetPos[0][3] = 0;
xb[2]._fTargetPos[0][4] = 180;
xb[2]._fTargetPos[0][5] = 0;
xb[3]._iBlendType = 0;  // line
xb[3]._fBlendRad  = 50;
xb[3]._fTargetPos[0][0] = 559;
```

```
xb[3]._fTargetPos[0][1] = 234.5;
xb[3]._fTargetPos[0][2] = 251.5;
xb[3]._fTargetPos[0][3] = 0;
xb[3]._fTargetPos[0][4] = 180;
xb[3]._fTargetPos[0][5] = 0;
drfl.moveb(xb, segNum, tvel, tacc);
Sleep(3000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.6.16 CDRFLEx.amove_spiral

Features

As an asynchronous move_spiral, it operates the same as the move_spiral() function except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion.

A new motion command generated before the motion is terminated by amove_spiral() function causes errors for security reasons. Therefore, the termination of the amove_spiral() motion must be confirmed using the mwait() function between amove_spiral() and the following motion command before the new motion command is executed.

The radius increases in a radial direction and the robot moves in parallel with the rotating spiral motion in an axial direction. It moves the robot along the spiral trajectory on the surface that is perpendicular to the axis on the coordinate specified as eMoveReference and the linear trajectory in the axis direction.

Parameter

Parameter Name	Data Type	Default Value	Range	Description
eTaskAxis	enum.TASK_AXIS	-	-	Refer to the Definition of Constant and Enumeration Type
fRevolution	float	-	rev > 0	Total number of revolutions [revolution]
fMaximuRadius	float	-	rmax > 0	Final spiral radius [mm]
fMaximumLength	float	-		Distance moved in the axis direction [mm]
fTargetVel	float[2]	-		Linear Velocity, Angular Velocity

Parameter Name	Data Type	Default Value	Range	Description
fTargetAcc	float[2]	-		Linear Acceleration, Angular Acceleration
fTargetTime	float	0.0	time ≥ 0	Total execution time <sec>
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_TOOL		Refer to the Definition of Constant and Enumeration Type

Note

- fRevolution refers to the total number of revolutions of the spiral motion.
- fMaximuRadius refers to the maximum radius of the spiral motion.
- fMaximumLength refers to the parallel distance in the axis direction during the motion. A negative value means the parallel distance in the –axis direction.
- fTargetVel refers to the moving velocity of the spiral motion.
- fTargetAcc refers to the moving acceleration of the spiral motion.
- When fTargetTime is specified, values are processed based on fTargetTime , ignoring fTargetVel and fTargetAcc.
- eTaskAxis defines the axis that is perpendicular to the surface defined by the spiral motion.
- eMoveReference refers to the reference coordinate system defined by the spiral motion.
- This function does not support online blending of previous and subsequent motions.

Caution

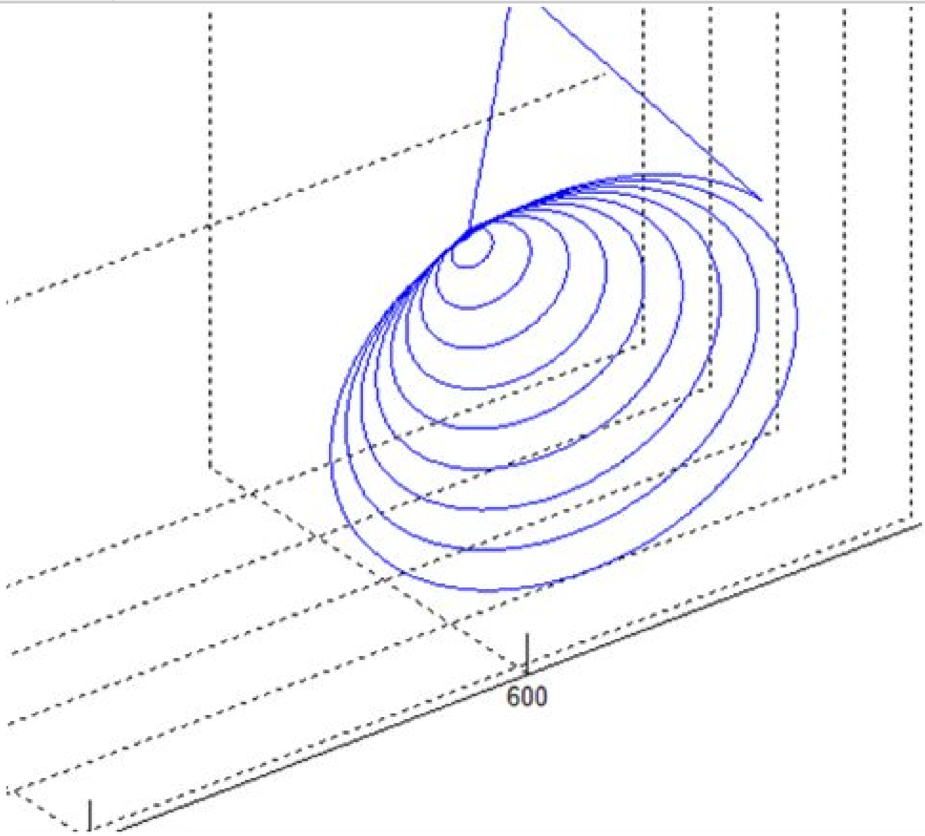
- If the rotating angular acceleration calculated by the spiral path is too great, an error can be generated to ensure safe motion.
  In this case, reduce the fTargetVel, fTargetAcc or fTargetTime value.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the spiral motion begins
float rev = 3;
float rmax = 50;
float lmax = 50;
float tvel = { 50, 50 };
float tacc = { 100, 100 };
Drfl.amove_spiral(TASK_AXIS_Z, rev, rmax, lmax, tvel, tacc);
Sleep(3000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```



4.6.17 CDRFLEx.amove_periodic

Features

As an asynchronous move_periodic, it operates the same as the move_periodic() function except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion. A new motion command generated before the motion is terminated by move_periodic() function causes errors for security reasons. Therefore, the termination of the amove_periodic() motion must be confirmed using mwait() function between the amove_periodic() function and the following motion command before the new motion command is executed.

This command performs a cyclic motion based on the sine function of each axis (parallel and rotation) of the reference coordinate (eMoveReference) input as a relative motion that begins at the current position. The attributes of the motion on each axis are determined by amplitude and period, and the acceleration/deceleration time and the total motion time are set by the interval and repetition count.

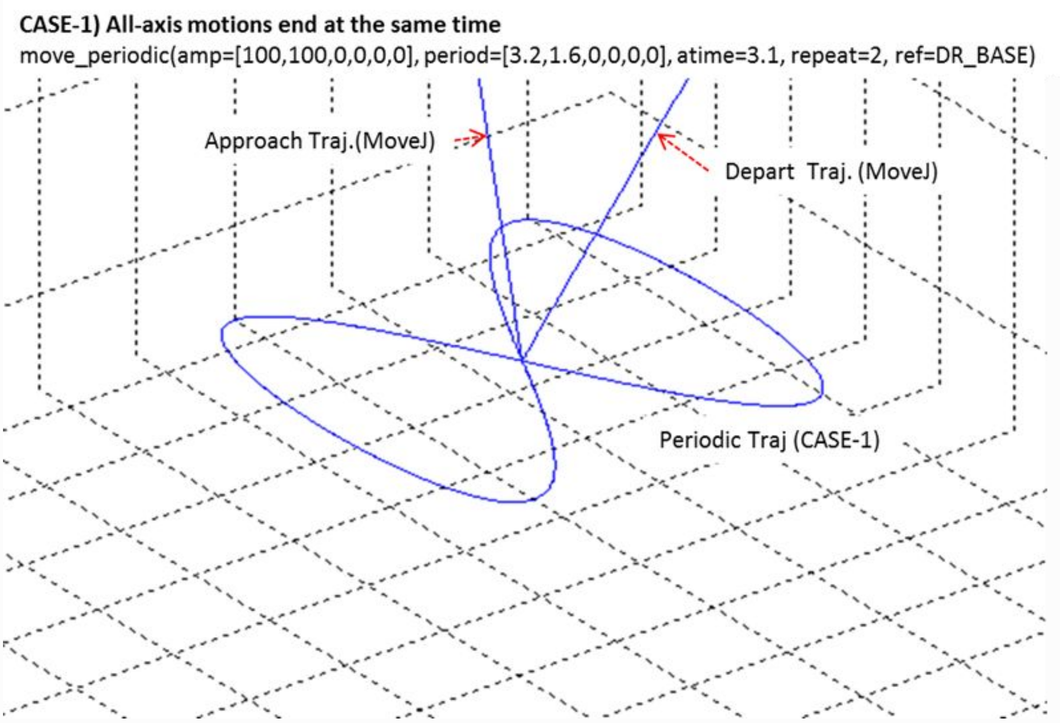
Parameter

Parameter Name	Data Type	Default Value	Range	Description
fAmplitude	float[6]	-	>=0	Amplitude(motion between -fAmplitude and +fAmplitude) [mm] or [deg]
fPeriodic	float[6]	-	>=0	period(time for one cycle) [sec]
fAccelTime	float	-	>=0	Acc-, dec- time [sec]
nRepeat	unsigned char	-	> 0	Repetition count
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_TOOL	-	Refer to the Definition of Constant and Enumeration Type

Note

- fAmplitude refers to the amplitude. The input is a list of six elements that are the amp values for the axes (x, y, z, rx, ry, and rz). However, the amp input on the axis that does not have a periodic motion must be 0.
- fPeriodic refers to the time needed to complete a motion in the direction, and the input is a list of six elements that are the amp values for the axes (x, y, z, rx, ry, and rz).
- fAccelTime refers to the acceleration and deceleration time at the beginning and end of the periodic motion. The largest of the inputted acceleration/deceleration times and maximum period*1/4 is applied. An error is generated when the inputted acceleration/deceleration time exceeds 1/2 of the total motion time.
- nRepeat refers to the number of repetitions of the axis (reference axis) that has the largest period value and determines the total motion time. The number of repetitions for each of the remaining axes is determined automatically according to the motion time.
- If the motion terminates normally, the motions for the remaining axes can be terminated before the reference axis's motion terminates so that the end position matches the starting

position. If the motions of all axes are not terminated at the same time, the deceleration section will deviate from the previous path. Refer to the following image for more information.

- 

- eMoveReference refers to the reference coordinate system of the repeated motion.
- If a maximum velocity error is generated during a motion, adjust the amplification and period using the following formula.
  velocity=Amplification(fAmplitude)*2*pi(3.14)/Period(fPeriodic**)**
  (i,e., Max. velocity=62.83 mm/sec if amp=10 mm, period=1)
- This function does not support online blending of previous and subsequent motions.

Return

Value	Description
0	Error
1	Success

Example

```
// Repeats the x-axis (10 mm amp and 1 sec. period) motion and y rotating axis (0.5 deg amp and 1 sec. period) motion in the tool coordinate system
// 5 times.
// SET(1) the Digital_Output channel no. 1, 1 second after the periodic motion begins.
```

```
float fAmplitude[NUM_TASK] = {10, 0, 0, 0, 0.5, 0};
float fPeriod[NUM_TASK] = { 1, 1, 1, 1, 1, 1};
Drfl.amove_periodic(fAmplitude, fPeriod, 0.5, 5, MOVE_REFERENCE_TOOL);
Sleep(1000);
Drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
Drfl.mwait();
```

4.6.18 CDRFLEx.stop

Features

This is a function for stopping the active motion in the robot controller. This function stops differently according to the eStopType received as an argument. All stop modes except Estop stop the motion in the currently active section.

Parameter

Parameter Name	Data Type	Default Value	Description
eStopType	enum.STOP_TYPE	STOP_TYPE_QUICK	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
# The motion is terminated with a soft stop 2 seconds after moving to x1
float x1[NUM_TASK] = {784, 543, 970, 0, 180, 0};
drfl.movel(x1, 100, 200)            // Executes the next command
immediately after the motion with x1.
Sleep(2000)           // Temporarily suspends the program for 2 seconds.
drfl.stop(STOP_TYPE_SLOW)    // Stops the motion with a soft Stop
```

4.6.19 CDRFLEx.move_pause

Features

This is a function for temporarily suspending the currently active robot motion in the robot controller. It is ignored if there is no active robot motion.

Parameter

None.

Return

Value	Description
0	Error
1	Success

Example

```
float j00[NUM_JOINT] = {0, 0, 90, 0, 90,};
drfl.amovej(j00, 20, 40);  // Starts asynchronous motion
while (1) {
// Checks the current joint angle
LPPOSITION pPose = drfl.get_current_pose(ROBOT_POSE_JOINT);
    if (pPose->_fTargetPos[2] >= 45) { // If the 3-axis angle is 45 degree
or more,
drfl.move_pause();            // it temporarily stops the motion.
Sleep(5000);                 // Waits for 5 seconds.
break;                      // Terminates while statement.
    }
}
drfl.move_resume();           // Resumes the motion.
```

4.6.20 CDRFLEx.move_resume

Features

This is a function for resuming the robot motion that was temporarily suspended by the move_pause function in the robot controller. It is ignored if there is no active robot path motion.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
float j00[NUM_JOINT] = {0, 0, 90, 0, 90,};
drfl.amovej(j00, 20, 40);  // Starts asynchronous motion
while (1) {
// Checks the current joint angle
LPPOSITION pPose = drfl.get_current_pose(ROBOT_POSE_JOINT);
    if (pPose->_fTargetPos[2] >= 45) { // If the 3-axis angle is 45 degree
or more,
        drfl.move_pause();            // it temporarily stops the motion.
        Sleep(5000);                 // Waits for 5 seconds.
        break;                      // Terminates while statement.
    }
}
drfl.move_resume();           // Resumes the motion.
```

4.6.21 CDRFLEx.mwait

Features

This is a function for waiting for the termination of the motion of the preceding motion command in the robot controller. If an asynchronous motion command is combined with this function, it can execute the same motion as a synchronous command.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
float point[6] = { 30, 30, 30, 30, 30, 30 };
drfl.amovej(point, 60, 120);
drfl.mwait();
```

4.6.22 CDRFLEx.trans

Features

Input parameter(fSourcePos) based on the ref coordinate is translated/rotated as fOffset based on the same coordinate and this function returns the result that is converted to the value based on the eTargetRef.

Parameter

Parameter Name	Data Type	Default Value	Description
fSourcePos	float[6]	-	Target joint location for six axes
fOffset	float[6]	-	Offset information for six axes
eSourceRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
float point[6] = { 30, 30, 30, 30, 30, 30 };
float offset[6] = { 100, 100, 100, 100, 100, 100};
LPROBOT_POSE res = Drfl.trans(point, offset);
for(int i=0; i<6; i++){
    cout << res->_fPosition[i] << endl;
}
```

4.6.23 CDRFLEx.fkin

Features

This function receives the input data of joint angles(fSourcePos) or equivalent forms (float[6]) in the joint space and returns the TCP (objects in the task space) based on the ref coordinate(eTargetRef).

Parameter

Parameter Name	Data Type	Default Value	Description
fSourcePos	float[6]	-	Target joint location for six axes
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
float q1[6] = {0,0,90,0,90,0};
Drfl.movej(q1, 60, 30);
float q2[6] = {30, 0, 90, 0, 90, 0};
LPROBOT_POSE res = Drfl.fkin(q2, COORDINATE_SYSTEM_WORLD);
float vel[2] = {100, 100};
float acc[2] = {200, 200};
float x2[6] = {0,};
for(int i=0; i<6; i++){
    x2[i] = res->_fPosition[i];
}
Drfl.MoveL(x2, vel, acc);
```

4.6.24 CDRFLEx.ikin

Features

This function receives the input data of joint angles(fSourcePos) or equivalent forms (float[6]) in the joint space and returns the TCP (objects in the task space) based on the ref coordinate(eTargetRef).

Parameter

Parameter Name	Data Type	Default Value	Description
fSourcePos	float[6]	-	Target joint location for six axes
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
float x1[6] = {370.9, 719.7, 651.5, 90, -180, 0};
LPROBOT_POSE res = Drfl.ikin(x1, 2);
float q1[6] = {0,};
```

```
for(int i=0; i<6; i++){
    q1[i] = res->_fPosition[i];
}
Drfl.movej(q1, 60, 30);
```

4.6.25 CDRFLEx.ikin(Extension)

Features

This function returns the joint position corresponding iSolutionSpace, which is equivalent to the robot pose in the operating space, among 8 joint shapes.

Parameter

Parameter Name	Data Type	Default Value	Description
fSourcePos	float[6]	-	Target task location for six axes
iSolutionSpace	uchar	-	solution space
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
float x1[6] = {370.9, 719.7, 651.5, 90, -180, 0};
LPINVERSE_KINEMATIC_RESPONSE res = Drfl.ikin(x1, 2, COORDINATE_BASE, 0);
float q1[6] = {0,};
for(int i=0; i<6; i++){
    q1[i] = res->_fPosition[i];
}
Drfl.movej(q1, 60, 30);
```

4.6.26 CDRFLEx.set_ref_coord

Features

This function sets the reference coordinate system.

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetCoordSystem	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float p1[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(p1, 60, 30);
float vec[2][3] = {{-1, 1, 1}, {1, 1, 0}};
float org[3] = {370.9, -419.7, 651.5};

int user = Drfl.set_user_cart_coord(vec, org);
Drfl.set_ref_coord((COORDINATE_SYSTEM)user);
```

4.6.27 CDRFLEx.check_motion

Features

This function checks the status of the currently active motion.

Parameter

None

Return

Value	Description
0	no motion in action
1	motion being calculated
2	motion in operation

Example

```
float q0[6] = {0, 0, 90, 0, 90, 0};
float q99[6] = {0, 0, 0, 0, 0, 0};
Drfl.amovej(q0, 60, 30); // Executes the next command immediately after the motion to q0.
while(true)
{
    if(Drfl.check_motion() == 0) // A motion is completed
    {
        Drfl.movej(q99, 60, 30); // Joint motion to q99.
        break;
    }
}
```

4.6.28 CDRFLEx.enable_alter_motion

Features

enable_alter_motion() and alter_motion() functions enable to alter motion trajectory.

This function sets the configurations for altering function and allows the input quantity of alter_motion() to be applied to motion trajectory. The unit cycle time of generating alter motion is 100msec. Cycle time(iCycleTime*100msec) can be changed through input parameter n. This function provide 2 modes(Accumulation mode, Increment mode). Input quantity of alter_motion() can be applied to motion trajectory in two ways as accumulated value or increment value. In accumulation mode, the input quantity means absolute altering amount(dX,dY,dZ,dRX,dRY,dRZ) from current motion trajectory.

On the contrary in increment mode, the quantity means increment value from the previous absolute altering amount. The reference coordinate can be changed through input parameter ref. Limitations of accumulation amout and increment amount can be set through input paramet fLimitDpos (accumulated limit) and fLimitDposPer(increment input limit during 1 cycle). The actual alter amount is limited to these limits.

This function is only available in M2.4 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
iCycleTime	int	-	Cycle time number
ePathMode	ePathMode	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	eTargetRef	-	Refer to the Definition of Constant and Enumeration Type
fLimitDpos	float[2]	-	First value : limitation of position[mm] Second value : limitation of orientation[deg]
fLimitDposPer	float[2]	-	First value : limitation of position[mm] Second value : limitation of orientation[deg]

Note

- alter_motion can be executed only in user thread.
- Accumulation amount or increment amout isn't be limited if limit_dPOS or limit_dPOS_per is None.

Return

Value	Description
0	Failed
1	Success

Example

```
float limit_dPOS[2] = {50, 90};
float limit_dPOS_per[2] = {50, 50};
```

```
Drfl.enable_alter_motion(5, PATH_MODE_DPOS, COORDINATE_SYSTEM_BASE,
limit_dPOS, limit_dPOS_per);
```

4.6.29 CDRFLEx.alter_motion

Features

This function applies altering amount of motion trajectory when the alter function is activated. The meaning of the input values is defined from enable_alter_motion().

This function is only available in M2.4 version or higher.

Caution

- alter_motion can be executed only in user thread.

Note

- alter_motion can be excuted only in user thread
- Alter motion can be adjusted through setting value fLimitDpos or fLimitDposPer in enable_alter_motion function
- Orientation of Input pose follows fixed XYZ notation.

Parameter

Parameter Name	Data Type	Default Value	Description
pos	float[6]	-	position list

Return

Value	Description
0	Failed
1	Success

Example

```
DWORD WINAPI ThreadFunc(void *arg){
    while(true){
```

```
        float pos[6] = {10, 0, 0, 10, 0, 0};
        Drfl.alter_motion(pos);
    }
}
…
int _tmain (int argc, _TCHAR* argv[]){
  HANDLE hThread;
  DWORD dwThreadID;
  hThread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, &dwThreadID);
  if (hThread == 0) {
    printf("Thread Error\n");
    return 0;
  }
  float limit_dPOS[2] = {50, 90};
  float limit_dPOS_per[2] = {50, 50};
  Drfl.enable_alter_motion(5, PATH_MODE_DPOS, COORDINATE_SYSTEM_BASE,
limit_dPOS, limit_dPOS_per);
}
```

4.6.30 CDRFLEx.disable_alter_motion

Features

This function deactivates alter motion.

This function is only available in M2.4 version or higher.

Parameter

None

Return

Value	Description
0	Failed
1	Success

Example

```
DWORD WINAPI ThreadFunc(void *arg){
while(true){
  float pos[6] = {10, 0, 0, 10, 0, 0};
  Drfl.alter_motion(pos);
  }
}
```

```
…
int _tmain (int argc, _TCHAR* argv[]){
  HANDLE hThread;
  DWORD dwThreadID;
  hThread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, &dwThreadID);
  if (hThread == 0) {
    printf("Thread Error\n");
    return 0;
  }
  float limit_dPOS[2] = {50, 90};
  float limit_dPOS_per[2] = {50, 50};
  Drfl.enable_alter_motion(5, PATH_MODE_DPOS, COORDINATE_SYSTEM_BASE,
limit_dPOS, limit_dPOS_per);
  Drfl.disable_alter_motion();.
}
```

4.6.31 CDRFLEx.servoj

Features

The command is the asynchronous motion command, and the next command is executed at the same time the motion begins. When the mode is set to Override mode, it follows the most recent target joint position in a motion within the maximum speed and acceleration among the continuously delivered commands. When the mode is set to Queue mode, it can store up to 100 previous commands and sequentially follow the path. When 100 waypoints have been stored in Queue mode, the command will not return until reaching the next waypoint. If an Override command is input during Queue mode, it ignores the previously stored waypoints and follows the most recent target joint position.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target joint location for six axes
fLimitVel	float[6]	-	Limit Velocity
fLimitAcc	float[6]	-	Limit Acceleration
fTargetTime	float	-	Time [sec]
eTargetMod	DR_SERVOJ_TYPE	DR_SERVO_QUEUE	Waypoint mode • DR_SERVO_OVERRIDE: Recent command • DR_SERVO_QUEUE: Queue mode (Keep waypoints)

Note

- If fTargetTime is entered, if the arrival time cannot be kept due to the maximum speed/ acceleration condition, it automatically adjusts and displays a notification message.

Caution

- Currently, it is not linked with the operation speed adjustment function of the change_operation_speed function.

Return

Value	Description
0	Failed
1	Success

Example

```
float q0[6] = { 0, 0, 90, 0, 90, 0 };
float jvel[6]={10, 10, 10, 10, 10, 10};
float jacc[6]= {10, 10, 10, 10, 10, 10};

drfl.servoj(q0, jvel, jacc, 0.02, DR_SERVO_OVERRIDE);
```

4.6.32 CDRFLEx.servol

Features

It is an asynchronous type motion command and executes the next command at the same time as the motion starts. It is a motion that follows the most recent target position among consecutive commands within the maximum speed and acceleration.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target location for six axes
fTargetVel	float[2]	-	Velocity
fTargetAcc	float[2]	-	Acceleration
fTargetTime	float	-	Time [sec]

Note

- If fTargetTime is entered, if the arrival time cannot be kept due to the maximum speed/ acceleration condition, it automatically adjusts and displays a notification message.

Caution

- Currently, it is not linked with the operation speed adjustment function of the change_operation_speed function.
- Currently, it is not linked with the DR_VAR_VEL option among the singularity processing options. When it is set as the DR_VAR_VEL option, it is automatically changed to the DR_AVOID option and a notification message is displayed.
- , it is not linked with the force/stiffness control command.

Return

Value	Description
0	Failed
1	Success

Example

```
float q0[6] = { 368, 34.5, 442, 50, -180, 50 };
float jvel[2]={10, 10};
float jacc[2]={20, 20};
```

```

drfl.servol(q0, jvel, jacc, -10000);
```

4.6.33 CDRFLEx.speedj

Features

It is an asynchronous type motion command and executes the next command at the same time as the motion starts. It is a motion that follows within the maximum speed and acceleration set for the most recent target joint speed among commands transmitted continuously.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetVel	float[6]	-	Velocity
fTargetAcc	float[6]	-	Acelerlation
fTargetTime	float	-	Time [Sec]

Note

- If fTargetTime is entered, if the arrival time cannot be kept due to the maximum speed/ acceleration condition, it automatically adjusts and displays a notification message.
- If you want to stop normally with speed, input vel as [0,0,0,0,0,0] or use the stop command.

For safety, if a new speedj command is not transmitted for 0.1 [sec] during movement, an error message is displayed and it stops.

Caution

- Currently, it is not linked with the operation speed adjustment function of the change_operation_speed function.

Return

Value	Description
0	Failed
1	Success

Example

```
float jvel={10, 10, 10, 10, 10, 10};
float jacc={20, 20, 20, 20, 20, 20};

drfl.speedj(jvel, jacc, -10000);
```

4.6.34 CDRFLEx.speedl

Features

It is an asynchronous type motion command and executes the next command at the same time as the motion starts. It is a motion that follows the most recent target position among consecutive commands within the maximum speed and acceleration.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetVel	float[6]	-	Velocity
fTargetAcc	float[2]	-	Acelerlation
fTargetTime	float	0.f	Time [Sec]

Note

- If fTargetTime is entered, if the arrival time cannot be kept due to the maximum speed/ acceleration condition, it automatically adjusts and displays a notification message.

For safety, if a new speedj command is not transmitted for 0.1 [sec] during movement, an error message is displayed and it stops.

Caution

- Currently, it is not linked with the operation speed adjustment function of the change_operation_speed function.
- Currently, it is not linked with the DR_VAR_VEL option among the singularity processing options. When it is set as the DR_VAR_VEL option, it is automatically changed to the DR_AVOID option and a notification message is displayed.

- , it is not linked with the force/stiffness control command.

Return

Value	Description
0	Failed
1	Success

Example

```
float jvel={10, 10, 10, 10, 10, 10};
float jacc={20, 20};

drfl.speedl(jvel, jacc, -10000);
```

4.6.35 CDRFLEx.amove_spiral(Extension)

Translation needed

The content of this page was copied from another page tree and needs to be translated or updated.

When you finish translation, make sure to

- Replace the label **NEEDS-TRANSLATING** with **TRANSLATED**
- Remove this macro from the page

Features

As an asynchronous move_spiral, it operates the same as the move_spiral() function except for the asynchronous process, and executes the next line by returning as soon as motion starts without waiting for the termination of motion.

A new motion command generated before the motion is terminated by amove_spiral() function causes errors for security reasons. Therefore, the termination of the amove_spiral() motion must be confirmed using the mwait() function between amove_spiral() and the following motion command before the new motion command is executed.

The radius increases in a radial direction and the robot moves in parallel with the rotating spiral motion in an axial direction. It moves the robot along the spiral trajectory on the surface that is perpendicular to the axis on the coordinate specified as eMoveReference and the linear trajectory in the axis direction.

Parameter

Parameter Name	Data Type	Default Value	Range	Description
eTaskAxis	enum.TASK_AXIS	-	-	Refer to the Definition of Constant and Enumeration Type
fRevolution	float	-	rev > 0	Total number of revolutions [revolution]
fTargetPos	float[3]	-	-	Target location
fTargetVel	float[2]	-		Linear Velocity, Angular Velocity
fTargetAcc	float[2]	-		Linear Acceleration, Angular Acceleration
fTargetTime	float	0.0	time ≥ 0	Total execution time <sec>
eMoveReference	enum.MOVE_REFERENCE	MOVE_REFERENCE_TOOL		Refer to the Definition of Constant and Enumeration Type
eMoveMode	enum.MOVE_MODE	MOVE_MODE_ABSOLUTE		Refer to the Definition of Constant and Enumeration Type
eSpiralDir	enum.SPIRAL_DIR	DR_SPIRAL_OUTWARD		Refer to the Definition of Constant and Enumeration Type
eRotDir	enum.ROT_DIR	DR_ROT_FORWARD		Refer to the Definition of Constant and Enumeration Type

Note

- fRevolution refers to the total number of revolutions of the spiral motion.
- fTargetVel refers to the moving velocity of the spiral motion.
- fTargetAcc refers to the moving acceleration of the spiral motion.
- When fTargetTime is specified, values are processed based on fTargetTime , ignoring fTargetVel and fTargetAcc.
- eTaskAxis defines the axis that is perpendicular to the surface defined by the spiral motion.
- eMoveReference refers to the reference coordinate system defined by the spiral motion.
- This function does not support online blending of previous and subsequent motions

Caution

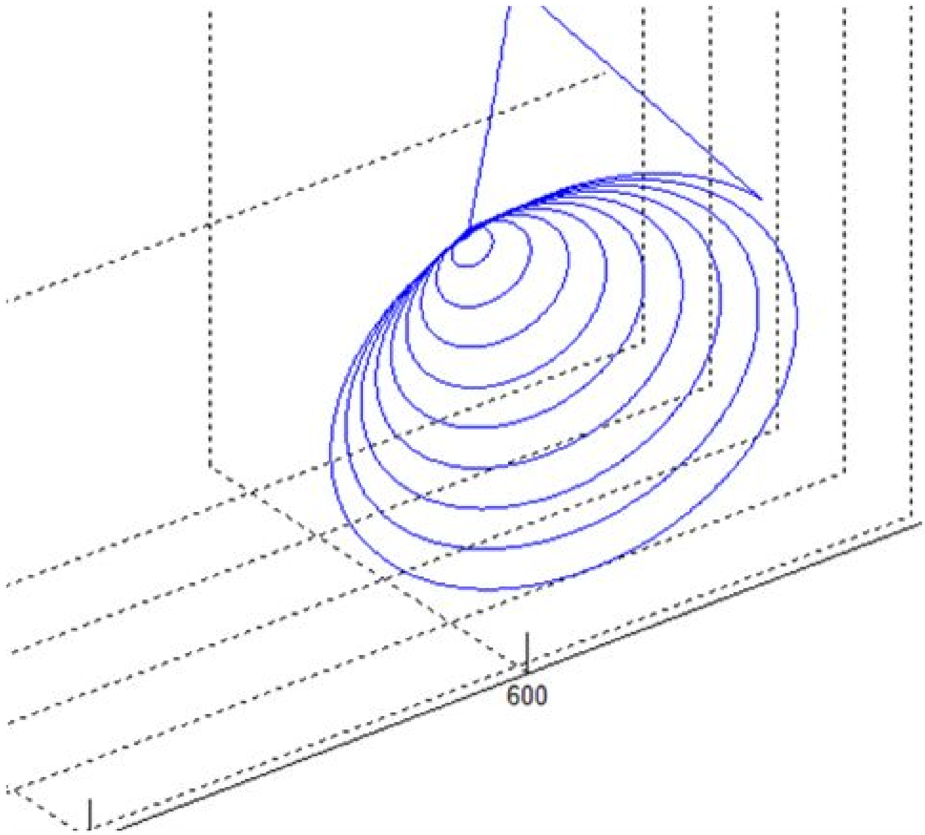
- If the rotating angular acceleration calculated by the spiral path is too great, an error can be generated to ensure safe motion.
  In this case, reduce the fTargetVel, fTargetAcc or fTargetTime value.

Return

Value	Description
0	Error
1	Success

Example

```
// D-Out 3 seconds after the spiral motion begins
float rev = 3;
float rmax = 50;
float lmax = 50;
float tvel = { 50, 50 };
float tacc = { 100, 100 };
Drfl.amove_spiral(TASK_AXIS_Z, rev, rmax, lmax, tvel, tacc);
Sleep(3000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```



4.7 Robot Setting Function

4.7.1 CDRFLEx.add_tool

Features

This is a function for using tool information that will be installed on the edge of the robot by registering it in advance for security reasons. The tool information registered using this function should be reset after rebooting, as it is stored in the memory. However, if it is registered the in T/P application, it can be reused, as it is added in the initialization process.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Tool Name
fCog	float[3]	-	Center of gravity

Parameter Name	Data Type	Default Value	Description
fWeight	float	-	Tool Weight
fInertia	float[6]	-	Inertia information

Return

Value	Description
0	Error
1	Success

Example

```
float fCog[3]  = {10, 10, 10};
float finertia[6] = { 0, 0, 0, 0, 0, 0 };
// Registers tool.
drfl.add_tool("tool#1", 5.3f, fCog, finertia);

// Selects currently mounted tool.
drfl.set_tool("tool#1");

// Releases registration of tool.
drfl.del_tool("tool#1");
```

4.7.2 CDRFLEx.del_tool

Features

This is a function for deleting information on the tool registered in the robot controller in advance.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Tool Name

Return

Value	Description
0	Error
1	Success

Example

```
float fCog[3]  = {10, 10, 10};
float finertia[6] = { 0, 0, 0, 0, 0, 0 };
// Registers tool.
drfl.add_tool("tool#1", 5.3f, fCog, finertia);

// Selects current tool.
drfl.set_tool("tool#1");

// Releases registration of tool.
drfl.del_tool("tool#1");
```

4.7.3 CDRFLEx.set_tool

Features

This is a function for setting the information on the tool currently mounted among the tool information registered in the robot controller in advance. When there is currently no tool mounted, if an empty character string is delivered, the currently set information is initialized.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Tool Name

cause

- If you use the set_tool and set_workpiece_weight functions consecutively, you must use the wait(transition_time) function as much as transition_time in between. Otherwise, the weight change may be erroneous.

Return

Value	Description
0	Error
1	Success

Example

```
float fCog[3]  = {10, 10, 10};
float finertia[6] = { 0, 0, 0, 0, 0, 0 };
// Registers tool.
drfl.add_tool("tool#1", 5.3f, fCog, finertia);

// Selects current tool.
drfl.set_tool("tool#1");

// Releases registration of tool.
drfl.del_tool("tool#1");
```

4.7.4 CDRFLEx.get_tool

Features

This is a function for retrieving the tool information currently set in the robot controller. When there is no currently set information on a tool, an empty character string is returned.

Parameter

None

Return

Value	Description
string	Tool Name

Example

```
string strTool = drfl.get_tool();
// Checks currently mounted tool.
```

```
if (strTool != “tool#1) {
drfl.set_tool(“tool#1);
}
```

4.7.5 CDRFLEx.add_tcp

Features

This is a function for using robot TCP information by registering it in advance for security reasons. The TCP information registered using this function should be reset after rebooting, as it is stored in the memory. However, if it is registered in the T/P application, it can be reused, as it is added in the initialization process.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	TCP Name
fPosition	float[6]	-	TCP Information

Return

Value	Description
0	Error
1	Success

Example

```
float fTCP[6] = { 10, 10, 10, 0, 0, 0 };
// Registers TCP.
drfl.add_tcp("tcp#1", fTCP);

// Sets current TCP
drfl.set_tcp("tcpl#1");

// Releases registration of TCP.
drfl.del_tcp(“tcp#1);
```

4.7.6 CDRFLEx.del_tcp

Features

This is a function for deleting information on the TCP registered in the robot controller in advance.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	TCP Name

Return

Value	Description
0	Error
1	Success

Example

```
float fTCP[6] = { 10, 10, 10, 0, 0, 0 };
// Registers TCP.
drfl.add_tcp("tcp#1", fTCP);

// Sets current TCP
drfl.set_tcp("tcpl#1");

// Releases registration of TCP.
drfl.del_tcp(“tcp#1);
```

4.7.7 CDRFLEx.set_tcp

Features

This is a function for setting the information on the TCP currently mounted among the TCP information registered in the robot controller in advance. When there is currently no TCP mounted, if an empty character string is delivered, the currently set information is initialized.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Tool Name

Return

Value	Description
0	Error
1	Success

Example

```
drfl.set_tcp(“tcp#1);

float point[6] = { 756.6f, 511.6f, 876.0f, 44.5f, 86.7f, 55.7f };
float vel[2] = {30, 0 };
float acc[2] = {30, 0 };

drfl.MoveL(point, vel, vel);
```

4.7.8 CDRFLEx.get_tcp

Features

This is a function for retrieving the TCP information currently set in the robot controller. When there is no currently set information on a tool, an empty character string is returned.

Parameter

None

Return

Value	Description
string	TCP Name

Example

```
string strTCP = drfl.get_tcp();
// Checks currently mounted TCP.
if (strTCP != “tcp#1) {
    drfl.set_tcp(“tcp#1);
}
```

4.7.9 CDRFLEx.set_tool_shape

Features

This function activates the tool shape information of the entered name among the tool shape information registered in the Teach Pendant.

This function is only available in M2.4 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	ToolShape name

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.set_tool_shape(“tool_shape1”)
```

4.7.10 CDRFLEx.get_workpiece_weight

Features

This function measures and returns the weight of the workpiece.

Parameter

None

Return

Value	Description
float	Mesured weight

Example

```
float weight = Drfl.get_workpiece_weight();
```

4.7.11 CDRFLEx.reset_workpiece_weight

Features

This function initializes the weight data of the material to initialize the algorithm before measuring the weight of the material.

Parameter

None

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.reset_workpiece_weight()
```

4.7.12 CDRFLEx.set_singularity_handling

Features

In case of path deviation due to the effect of singularity in task motion, user can select the response policy. The mode can be set as follows.

- Automatic avoidance mode(Default) : SINGULARITY_AVOIDANCE_AVOID
- Path first mode : SINGULARITY_AVOIDANCE_STOP
- Variable velocity mode : SINGULARITY_AVOIDANCE_VEL

The default setting is automatic avoidance mode, which reduces instability caused by singularity, but reduces path tracking accuracy. In case of path first setting, if there is possibility of instability due to singularity, a warning message is output after deceleration and then the corresponding task is terminated. In case of variable velocity mode setting, TCP velocity would be changed in singular region to reduce instability and maintain path tracking accuracy.

Parameter

Parameter Name	Data Type	Default Value	Description
eMode	SINGULARITY_AVOIDANCE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float p1[6] = {400, 500, 800, 0, 180, 0};
float p2[6] = {400, 500, 500, 0, 180, 0};
float p3[6] = {400, 500, 200, 0, 180, 0};
Drfl.set_singularity_handling(SINGULARITY_AVOIDANCE_AVOID); // Automatic avoidance mode for singularity
Drfl.movel(p1, 10, 20);
Drfl.set_singularity_handling(SINGULARITY_AVOIDANCE _STOP); // Task motion path first
Drfl.movel(p2, 30, 60);
```

```
Drfl.set_singularity_handling(SINGULARITY_AVOIDANCE_VEL); // Variable velocity mode for singularity
```

4.7.13 CDRFLEx.setup_monitoring_version

Features

This is a function for setting version information of monitoring information transmitted from the robot controller.

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
iVersion	int	-	Version information 0 : version 0 1 : version 1

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.set_up_monitoring_version(1); // Set monitoring data version information to 1
```

4.7.14 CDRFLEx.config_program_watch_variable

This function is used to set the variable name to be monitored in order to monitor the internal variables of the program when the program is executed in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eDivision	VARIABLE_TYPE	-	Refer to the Definition of Constant and Enumeration Type
eType	TYPE_TYPE	-	Refer to the Definition of Constant and Enumeration Type
strName	string	-	128-byte variable name string
strData	string	-	128-byte data string

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.config_program_watch_variable(VARIABLE_TYPE_INSTALL, DATA_TYPE_FLOAT, "var", "1.22"); // Save the installation variable “var” as float 1.22
```

4.7.15 CDRFLEx.set_user_home

Features

This is a function to set the user's home position in the robot controller.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_user_home();
Drfl.move_home(MOVE_HOME_USER, (unsigned char)1);
```

4.7.16 CDRFLEx.servo_off

Features

This is a function to set the motor and brake power (robot power off) in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eStopType	enum.STOP_TYPE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.servo_off(STOP_TYPE_QUICK);
```

4.7.17 CDRFLEx.release_protective_stop

Features

This is a function to release the protective stop state in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eReleaseMode	enum.RELEASE_MODE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.release_protective_stop(RELEASE_MODE_RELEASE);
```

4.7.18 CDRFLEx.change_collision_sensitivity

Features

This is a function to configure the collision sensitivity in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
fSensitivity	float	-	Collision Sensitivity(0~100)

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.change_collision_sensitivity(20);
```

4.7.19 CDRFLEx.set_safety_mode

Features

This is a function to set the safety state in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eSafetyMode	enum.SAFETY_MODE	-	Refer to the Definition of Constant and Enumeration Type
eSafetyEvent	enum.SAFETY_EVENT	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_safety_mode(SAFETY_MODE_MANUAL, SAFETY_EVENT_MOVE); // set the safety state to 'Manual', 'Operating"
```

4.7.20 CDRFLEx.set_auto_servo_off

Features

This is a function to set the automatic safe-off state transition function that is operated when a certain period of time has elapsed in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
bFuncEnable	bool	-	Disable : 0 Enable : 1
fElapseTime	float	-	Minute

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_auto_servo_off(1, 3);
```

4.7.21 CDRFLEx.set_workpiece_weight

Features

In addition to the tool weight/center of gravity at the end of the robot, set the weight/center of gravity of the work piece and other information. The weight and center of gravity of the entire payload is reflected by combining the set tool weight/center of gravity and the work piece's weight/center of gravity. It can be used in applications where the type of workpiece is frequently varied or the weight needs to be dynamically changed.

Caution

- Workpiece weight change is allowed only when both Collision Detection and TCP SLF Violation check are mute or deactivated during Auto Mode.
- In the current version, Collision Detection considers the function mute when Collision Sensitivity is overridden to 0 and TCP SLF considers the function mute when the TCP SLF Limit is overridden to the maximum. This override can be set using Collision Sensitivity Reduction Zone and Custom Zone.
- Otherwise, trigger an SS1 protective stop unless the workpiece weight is set to zero.
- If the robot stops due to an error and needs to be manually restored, place the robot in the desired position in the Recovery Mode and unload the workpiece through Servo On and I/O operation while the corresponding zones are activated in Auto Mode.
- When changing the set tool weight, the workpiece weight is initialized to 0.
- When using the set_tool and set_workpiece_weight functions in succession, you must use the wait(transition_time) function as much as transition_time between them. Otherwise, there may be errors in the weight change.

Parameter

Parameter Name	Data Type	Default Value	Description
fWiehgt	float	0	Weight [kg]
fCog	float[3]	[0, 0, 0]	Center of gravity of the workpiece (x, y, z) [mm]
eCogRef	enum.COG_REFERENCE	COG_REFERENCE_TCP	Reference coordinates of center of gravity position, DR_CUR_TCP : TCP coordinates, DR_FLANGE : FLANGE coordinates,

Parameter Name	Data Type	Default Value	Description
eAddUp	enum.ADD_UP	ADD_UP_REPLACE-	DR_REPLACE(0): Replace workpiece DR_ADD(1): Add workpiece DR_REMOVE(2): Remove workpiece
fStartTime	float	-10000	Starting time of changing workpiece weight [sec]
fTransitionTime	float	-10000	Transition time of changing workpiece weight [sec]

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_safety_mode(SAFETY_MODE_AUTONOMOUS, SAFETY_MODE_EVENT_MOVE);
Drfl.set_workpiece_weight();
Drfl.set_safety_mode(SAFETY_MODE_AUTONOMOUS, SAFETY_MODE_EVENT_STOP);
```

4.8 I/O Control Function

4.8.1 CDRFLEx.set_tool_digital_output

Features

This is a function for outputting a signal at the digital contact point mounted on the edge of the robot in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIndex	enum.GPIO_TOOL_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type
bOnOff	bool	-	Data to output • ON: 1 • OFF: 0

Return

Value	Description
0	Error
1	Success

Example

```
//Outputs the digital No. 1 output contact point on the robot arm
drfl.set_tool_digital_output(GPIO_TOOL_DIGITAL_INDEX_1, 1);
```

4.8.2 CDRFLEx.get_tool_digital_input

Features

This is a function for checking a signal at the digital contact point mounted on the edge of the robot in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_TOOL_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	OFF
1	ON

Example

```
//Checks the digital No. 1 input contact point on the robot arm
bool bSignal = drfl.get_tool_digital_input(GPIO_TOOL_DIGITAL_INDEX_1);
if (bSignal == True) {
  // do something
}
```

4.8.3 CDRFLEx.get_tool_digital_output

Features

This is a function for checking a signal at the digital contact point mounted on the edge of the robot in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_TOOL_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	OFF
1	ON

Example

```
//Checks the digital No. 1 output contact point on the robot arm
bool bSignal = drfl.get_tool_digital_output(GPIO_TOOL_DIGITAL_INDEX_1);
if (bSignal == True) {
  // do something
}
```

4.8.4 CDRFLEx.set_digital_output

Features

This is a function for outputting a signal at the digital contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type
bOnOff	bool	-	Data to output Ÿ ON: 1 Ÿ OFF: 0

Return

Value	Description
0	Error
1	Success

Example

```
//Outputs the digital No. 1 output contact point on the control box
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

4.8.5 CDRFLEx.get_digital_input

Features

This is a function for checking a signal at the digital contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	OFF
1	ON

Example

```
//Checks the digital No. 1 input contact point on the control box
bool bSignal = drfl.get_digital_input(GPIO_CTRLBOX_DIGITAL_INDEX_1);
if (bSignal ==TRUE) {
  // do something
}
```

4.8.6 CRDFLEx.get_digital_output

Features

This is a function for checking a signal at the digital contact point mounted on the control box in the robot controller.Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ DIGITAL_INDEX	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	OFF
1	ON

Example

```
// Checks the digital No. 1 output contact point on the control box
bool bSignal = drfl.get_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1);
if (bSignal ==TRUE) {
  // do something
}
```

4.8.7 CDRFLEx.set_mode_analog_input

Features

This is a function for setting the channel mode for the analog input contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ANALOG_INDEX_	-	Refer to the Definition of Constant and Enumeration Type
mod	enum.GPIO_ANALOG_TYPE	GPIO_ANALOG_TYPE_CURRENT	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
// Sets the analog No. 1 input contact point on the control box to current mode.
drfl.set_mode_analog_input(GPIO_CTRLBOX_ANALOG_INDEX_1, GPIO_ANALOG_TYPE_CURRENT);

// Sets the analog No. 2 input contact point on the control box to voltage mode.
drfl.set_mode_analog_input(GPIO_CTRLBOX_ANALOG_INDEX_2, GPIO_ANALOG_TYPE_VOLTAGE);
```

4.8.8 CDRFLEx.set_mode_analog_output

Features

This is a function for setting the channel mode for the analog output contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ANALOG_INDEX_	-	Refer to the Definition of Constant and Enumeration Type
mod	enum.GPIO_ANALOG_TYPE	GPIO_ANALOG_TYPE_CURRENT	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
// Sets the analog No. 1 output contact point on the control box to current mode.
drfl.set_mode_analog_output(pCtrl, GPIO_CTRLBOX_ANALOG_INDEX_1, GPIO_ANALOG_TYPE_CURRENT);

// Sets the analog No. 2 output contact point on the control box to voltage mode.
drfl.set_mode_analog_output(pCtrl, GPIO_CTRLBOX_ANALOG_INDEX_2, GPIO_ANALOG_TYPE_VOLTAGE);
```

4.8.9 CDRFLEx.set_analog_output

Features

This is a function for outputting a signal at the analog contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ANALOG_INDEX_	-	Refer to the Definition of Constant and Enumeration Type
fValue	float	-	Analog signal output Ÿ In the case of current mode: 4.0~20.0 [mA] Ÿ In the case of voltage mode: 0~10.0 [V]

Return

Value	Description
0	Error
1	Success

Example

```
// Sets the analog No. 1 output contact point on the control box to current mode.
drfl.set_mode_analog_output(GPIO_CTRLBOX_ANALOG_INDEX_1, GPIO_ANALOG_TYPE_CURRENT);

// Outputs 5.2mA on the analog No. 1 output contact point on the control box.
drfl.set_analog_output(GPIO_CTRLBOX_ANALOG_INDEX_1, 5.2);
```

4.8.10 CDRFLEx.get_analog_input

Features

This is a function for checking a signal at the analog contact point mounted on the control box in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
eGpioIdex	enum.GPIO_CTRLBOX_ANALOG_INDEX_	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
Analog signal input	In the case of current mode: 4.0~20.0 [mA] In the case of voltage mode: 0~10.0 [V]

Example

```
// Sets the analog No. 1 input contact point on the control box to current mode.
drfl.set_mode_analog_output(GPIO_CTRLBOX_ANALOG_INDEX_1, GPIO_ANALOG_TYPE_CURRENT);

// Checks the current value on the analog No. 1 input contact point on the control box.
float fCurrent = drfl.get_analog_input(GPIO_CTRLBOX_ANALOG_INDEX_1);
```

4.8.11 CDRFLEx.add_modbus_signal

Features

This is a function for using the Modbus I/O signal by registering it in advance. The modbus I/O signal registered using this function should be reset after rebooting, as it is stored in the memory. However, if it is registered in the T/P application, it can be reused, as it is added in the initialization process.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	modbus signal Name
strIpAddress	string	-	modbus module ip address
nPort	unsigned short	-	modbus module port
eRegType	enum	-	Refer to the Definition of Constant and Enumeration Type
iRegIndex	unsigned short	-	Index of Modbus signal
nRegValue	unsigned short	0	Output value when type is MODBUS_REGISTER_TYPE_COILS or MODBUS_REGISTER_TYPE_HOLDING_REGISTER (ignored in other cases)

Return

Value	Description
0	Error
1	Success

Example

```
/*
Example of connecting Modbus IO and allocating contact point
Modbus IO IP: 192.168.127.254
input 2 points: “di1”,”di2”
*/

// set <modbus> input : “di1”, “di2”
drfl.add_modbus_signal("di1", "192.168.127.254" , 502,
MODBUS_REGISTER_TYPE_DISCRETE_INPUTS, 0, 0);

drfl.add_modbus_signal("di2", "192.168.127.254" , 502,
MODBUS_REGISTER_TYPE_DISCRETE_INPUTS, 0, 0);
```

4.8.12 CDRFLEx.del_modbus_signal

Features

This is a function for deleting information on the Modbus I/O signal registered in the robot controller in advance.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Name of registered modbus signal

Return

Value	Description
0	Error

Value	Description
1	Success

Example

```
/*
 If the Modbus IO signal is registered as “di1” and ”do1,”
 and you want to delete this signal registration.
*/
drfl.del_modbus_signal("di1")   // Deletes “di1” contact point registration
drfl.del_modbus_signal("do1")   // Deletes “do1” contact point registration
```

4.8.13 CDRFLEx.set_modbus_output

Features

This is a function for outputting a signal at the modbus I/O contact point in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Modbus name
nValue	unsigned short	-	• In the case of Modbus digital I/O: 0 or 1 • In the case of Modbus analog: Data

Return

Value	Description
0	Error
1	Success

Example

```
//When the Modbus digital I/O is connected and the signal is registered as "di1" and "do1,"
drfl.set_modbus_output("do1",1);
drfl.set_modbus_output("do2",0);
```

4.8.14 CDRFLEx.get_modbus_input

Features

This is a function for checking a signal at the modbus I/O contact point in the robot controller.

Parameter

Parameter Name	Data Type	Default Value	Description
strSymbol	string	-	Modbus name

Return

Value	Description
unsigned shor	• In the case of Modbus digital I/O: 0 or 1 • In the case of Modbus analog: Data

Example

```
//When the Modbus digital I/O is connected and the signal is registered as "di1" and "di2,"
unsigned short siganl1 = drfl.get_modbus_input("di1");
unsigned short signal2 = drfl.get_modbus_input("di2");
if ( signal1 == 1 && signal2 == 1) {

    // do something…
}
```

4.8.15 CDRFLEx.flange_serial_open

Features

This is a function for open flange serial transfer port in the robot controller.

This command is not available in the new flange version (v2).

Parameter

Parameter Name	Data Type	Default Value	Description
Baudrate	Int	115200	Baudrate 2400, 4800, 9600, 19200, 38400, 57600, 115200 etc
Bytesize	enum.BYTE_SIZE	BYTE_SIZE_EIGHTBITS	Refer to the Definition of Constant and Enumeration Type
Parity	enum.PARITY_CHECK	PARITY_CHECK_NONE	Refer to the Definition of Constant and Enumeration Type
stopbits	enum.STOP_BITS	STOPBITS_ONE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.flange_serial_open(115200);
Drfl.flange_serial_write(8, "test1234");
LPFLANGE_SER_RXD_INFO temp = Drfl.flange_serial_read(4.5);
cout << "serial read : " << temp->_cRxd << endl;
cout << "serial cnt : " << (int)temp->_iSize << endl;
Drfl.flange_serial_close();
```

4.8.16 CDRFLEx.flange_serial_close

Features

This is a function for close flange serial transfer port in the robot controller.

This command is not available in the new flange version (v2).

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.flange_serial_open(115200);
Drfl.flange_serial_write(8, "test1234");
LPFLANGE_SER_RXD_INFO temp = Drfl.flange_serial_read(4.5);
cout << "serial read : " << temp->_cRxd << endl;
cout << "serial cnt : " << (int)temp->_iSize << endl;
Drfl.flange_serial_close();
```

4.8.17 CDRFLEx.flange_serial_write

Features

This is a function for send serial data in the robot controller.

The port parameter is only available in the new flange version (v2)

Parameter

Parameter Name	Data Type	Default Value	Description
nSize	Int	-	RX_Data Size
pSendData	char	-	Data value(Maximum 256bytes)
nPort	int	1	port number 1 : X1 2 : X2(support M/H model only)

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.flange_serial_open(115200);
Drfl.flange_serial_write(8, "test1234");
LPFLANGE_SER_RXD_INFO temp = Drfl.flange_serial_read(4.5);
cout << "serial read : " << temp->_cRxd << endl;
cout << "serial cnt : " << (int)temp->_iSize << endl;
Drfl.flange_serial_close();
```

4.8.18 CDRFLEx.get_tool_analog_input

Features

This is a function for load the channel value corresponding to the analog input from the robot flange.

The port parameter is only available in the new flange version (v2)

Parameter

Parameter Name	Data Type	Default Value	Description
nCh	int	-	1 : channel 1 2 : channel 2 3 : channel 3 (support M/H model only) 4 : channel 4 (support M/H model only)

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_mode_tool_analog_input(1, GPIO_ANALOG_TYPE_CURRENT);
Drfl.set_mode_tool_analog_input(2, GPIO_ANALOG_TYPE_VOLTAGE);
float cur = Drfl.get_tool_analog_input(1);
float vol = Drfl.get_tool_analog_input(2);
```

4.8.19 CDRFLEx.set_tool_digital_output_level

Features

This is a function for set the output strength for digital output on the robot flange.

The port parameter is only available in the new flange version (v2)

Parameter

Parameter Name	Data Type	Default Value	Description
nLv	Int	12	The intensity of the voltage to be output to the digital output installed in the flange I/O - 0, 12, 24 [V]

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_tool_digital_output_level(24);
```

4.8.20 CDRFLEx.set_tool_digital_output_type

Features

This is a function for set the output type for digital output on the robot flange.

The port parameter is only available in the new flange version (v2)

Parameter

Parameter Name	Data Type	Default Value	Description
nPort	int	1	port number
eOutputType	enum.OUTPUT_TYPE	OUTPUT_TYPE_PNP	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.set_tool_digital_output_type(1, OUTPUT_TYPE_PNP);
```

4.8.21 CDRFLEx.set_mode_tool_analog_input

Features

is a function for set the input type for analog input on the robot flange.

The port parameter is only available in the new flange version (v2)

Parameter

Parameter Name	Data Type	Default Value	Description
nCh	int	-	1 : channel 1 2 : channel 2 3 : channel 3 (support M/H model only) 4 : channel 4 (support M/H model only)
eAnalogType	enum.GPIO_ANALOG_TYPE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Error

Value	Description
1	Success

Example

```
Drfl.set_mode_tool_analog_input(1, GPIO_ANALOG_TYPE_CURRENT);
Drfl.set_mode_tool_analog_input(2, GPIO_ANALOG_TYPE_VOLTAGE);
float cur = Drfl.get_tool_analog_input(1);
float vol = Drfl.get_tool_analog_input(2);
```

4.9 Program Control Function

4.9.1 CDRFLEx.drl_start

Features

This is a function for executing the program (task) consisting of the DRL language in the robot controller.

Parameter

Parameter Name	**Data Type**	**Default Value**	**Description**
eRobotSystem	enum.ROBOT_SYSTEM		Refer to the Definition of Constant and Enumeration Type
strDrlProgram	string		Program character string to execute

Return

Value	**Description**
0	Error
1	Success

Example

```
string strDrlProgram = "\r\n\
loop = 0\r\n\
while loop < 3:\r\n\
 movej(posj(10,10.10,10,10.10), vel=60, acc=60)\r\n\
 movej(posj(00,00.00,00,00.00), vel=60, acc=60)\r\n\
 loop+=1\r\n\
movej(posj(10,10.10,10,10.10), vel=60, acc=60)\r\n";

if (drfl.get_robot_state() == eSTATE_STANDBY) {
    if (drfl.get_robot_mode() == ROBOT_MODE_AUTONOMOUS) {
    // Automatic Mode
    ROBOT_SYSTEM eTargetSystem = ROBOT_SYSTEM_VIRTUAL;
    drfl.drl_start(eTargetSystem, strDrlProgram)
    }
}
```

Note

- The robot operation state should be STATE_STANDBY, and the robot motions normally when the robot mode is automatic mode.
- The DRL program should be made by referring to the programming manual document in the appendix.

4.9.2 CDRFLEx.drl_stop

Features

This is a function for stopping the DRL program (task) currently executed in the robot controller. This function stops differently according to the iStopType received as an argument and stops the motion in the currently active section.

Parameter

Parameter Name	Data Type	Default Value	Description
iStopType	unsigned char	1	0 : Slow Stop 1 : Quick Stop

Return

Value	Description
0	Error
1	Success

Example

```
DRL_PROGRAM_STATE eProgramState = drfl. get_program_state();
if ((eProgramState == DRL_PROGRAM_STATE_PLAY) ||
    (eProgramState == DRL_PROGRAM_STATE_HOLD)) {

    drfl.drl_stop(1);
    //…
}
```

4.9.3 CDRFLEx.drl_pause

Features

This is a function for temporarily suspending the DRL program (task) currently executed in the robot controller.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
if (drfl.get_program_state() == DRL_PROGRAM_STATE_PLAY) {
    drfl.drl_pause();
}
```

4.9.4 CDRFLEx.drl_resume

Features

This is a function for resuming the DRL program (task) currently temporarily suspended in the robot controller.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
if (drfl. get_program_state() == DRL_PROGRAM_STATE_HOLD) {
    bool bResult = drfl.drl_resume();
    //...
}
```

4.9.5 CDRFLEx.change_operation_speed

Features

This function adjusts the operation velocity. The argument is the relative velocity in a percentage of the currently set velocity and has a value from 1 to 100. Therefore, a value of 50 means that the velocity is reduced to 50% of the currently set velocity.

Parameter

Parameter Name	Data Type	Default Value	Description
speed	float	-	operation speed(10~100)%

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.change_operation_speed(10);
string strDrlProgram = "loop = 0\nwhile loop < 3:\n movej(posj(10,10.10,10,10.10), vel=60, acc=60)\n movej(posj(00,00.00,00,00.00), vel=60, acc=60)\n  loop+=1\n movej(posj(10,10.10,10,10.10), vel=60, acc=60)";

if (Drfl.get_robot_state() == STATE_STANDBY) {
    Drfl.set_robot_mode(ROBOT_MODE_AUTONOMOUS);
    if (Drfl.get_robot_mode() == ROBOT_MODE_AUTONOMOUS) {
    // Autonomous Mode
    ROBOT_SYSTEM eTargetSystem = ROBOT_SYSTEM_VIRTUAL;
    Drfl.drl_start(eTargetSystem, strDrlProgram);
    }
}
```

4.9.6 CDRFLEx.save_sub_program

Features

The created DRL language program (task) is stored as a sub-program.

Parameter

Parameter Name	Data Type	Default Value	Description
iTargetSystem	SUB_PROGRAM	-	Refer to the Definition of Constant and Enumeration Type
strFileName	string	-	256-byte file name information
strDrlProgram	string	-	string buffer

Return

Value	Description
0	Failed
1	Success

Example

```
string drl_string= "movej([0,0,0,0,0,0], 60, 30)";

Drfl.save_sub_program(SUB_PROGRAM_SAVE, "sub_test", drl_string.c_str());
```

4.9.7 CDRFLEx.tp_popup_response

Features

This function controls the next operation (stop and resume of a paused program) according to the user's response after outputting a type message (Popup) during DRL program operation.

Parameter

Parameter Name	Data Type	Default Value	Description
eRes	POPUP_RESPONSE	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
// When executing DRL command tp_popup
```

```
Drfl.tp_popup_response(POPUP_RESPONSE_RESUME);
```

4.9.8 CDRFLEx.tp_get_user_input_response

Features

This function transfers user input information when a user input is requested during DRL program operation.

Parameter

Parameter Name	Data Type	Default Value	Description
strUserInput	string	-	256-byte user input string

Return

Value	Description
0	Failed
1	Success

Example

```
// When executing DRL command tp_get_user_input
Drfl.tp_get_user_input_response("tp get user input response");
```

4.10 Force/Stiffness Control and Other User-Friendly Features

4.10.1 CDRFLEx.parallel_axis

CDRFLEx.parallel_axis(fTargetPos1, fTargetPos2, fTargetPos3, eTaskAxis, eSourceRef)

Features

This function matches the normal vector of the plane consists of points(fTargetPos1, fTargetPos2, fTargetPos2) based on the ref coordinate(eTargetRef) and the designated axis(eTaskAxis) of the tool frame. The current position is maintained as the TCP position of the robot.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos1	float[6]	-	Target task location for six axes
fTargetPos2	float[6]	-	Target task location for six axes
fTargetPos3	float[6]	-	Target task location for six axes
eTaskAxis	enum.TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eSourceRef	enum.COORDINATE_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float x0[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(x0, 60, 30);
float x1[6] = {0, 500, 700, 30, 0, 90};
float x2[6] = {500, 0, 700, 0, 0, 45};
float x3[6] = {300, 100, 500, 45, 0, 45};
Drfl.parallel_axis(x1, x2, x3, TASK_AXIS_X);
```

CDRFLEx.parallel_axis(eSourceVec, eTaskAxis, eTargetRef)

Features

This function matches the given vect direction(eSourceVec) based on the ref coordinate(eTargetRef) and the designated axis of the tool frame. The current position is maintained as the TCP position of the robot.

Parameter

Parameter Name	Data Type	Default Value	Description
eSourceVec	float[3]	-	vector
eTaskAxis	enum.TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	enum.COORDINATE_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float v[3] = {1000, 700, 300};

Drfl.parallel_axis(v, TASK_AXIS_X, COORDINATE_SYSTEM_BASE);
```

4.10.2 CDRFLEx.align_axis

CDRFLEx.align_axis(fTargetPos1, fTargetPos2, fTargetPos3, fSourceVec, eTaskAxis, eTargetRef)

Features

This function matches the normal vector of the plane consists of points(fTargetPos1, fTargetPos2, fTargetPos3) based on the ref coordinate(eTargetRef)) and the designated axis of the tool frame. The robot TCP moves to the pos position.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos1	float[6]	-	Target task location for six axes
fTargetPos2	float[6]		Target task location for six axes
fTargetPos3	float[6]		Target task location for six axes
fSourceVec	float[3]		vector
eTaskAxis	enum.TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	enum.COORDINATE_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float x1[6] = {0, 500, 700, 30, 0, 0};
float x2[6] = {500, 0, 700, 0, 0, 0};
float x3[6] = {300, 100, 500, 0, 0, 0};
float pos[3] = {400, 400, 500};
Drfl.align_axis(x1, x2, x3, pos, TASK_AXIS_X, COORDINATE_SYSTEM_BASE);
```

CDRFLEx.align_axis(eTargetVec, eSourceVec, eTaskAxis, eTargetRef)

Features

This function matches the given vect direction(eTargetVec) based on the ref coordinate(eTargetRef) and the designated axis of the tool frame. The robot TCP moves to the pos position.

Parameter

Parameter Name	Data Type	Default Value	Description
eTargetVec	float[3]	-	vector
eSourceVec	float[3]		vector
eTaskAxis	enum.TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	enum.COORDINATE_SYSTEM	-	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
float v1[3] = {400, 400, 500};
float v2[3] = {350, 37, 430};
Drfl.align_axis(v1, v2, TASK_AXIS_X, COORDINATE_SYSTEM_BASE);
```

4.10.3 CDRFLEx.is_done_bolt_tightening

Features

This function monitors the tightening torque of the tool and returns True if the set torque (m) is reached within the given time and False if the given time has passed.

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type

Parameter Name	Data Type	Default Value	Description
fTargetTor	float	0	Target torque
fTimeout	float	0	Monitoring duration [sec]

Return

Value	Description
0	Failed
1	Success

Example

```
float p0[6] = {0,0,90,0,90,0};
Drfl.movej(p0, 60, 30);
float stx[6] = {3000, 3000, 3000, 200, 200, 200};
Drfl.task_compliance_ctrl(stx);
float x1[6] = {559, 34.5, 651.5, 0, 180, 60};
float velx[2] = {50, 50};
float accx[2] = {50, 50};
Drfl.amovel(x1, velx, accx);
bool res = Drfl.is_done_bolt_tightening(FORCE_AXIS_Z, 10, 5);
int x = 0;
if(res){
    x = 1;
}
else{
    x = 2;
}
```

4.10.4 CDRFLEx.task_compliance_ctrl

Features

This function begins task compliance control based on the preset reference coordinate system.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetStiffness	float[6]	[3000, 3000, 3000, 200, 200, 200]	Three translational stiffnesses Three rotational stiffnesses
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type
fTargetTime	float	0	Stiffness varying time [sec] Range: 0 - 1.0 * Linear transition during the specified time

Return

Value	Description
0	Failed
1	Success

Example

```
float p0[6] = {0,0,90,0,90,0};
Drfl.movej(p0, 60, 30);
float stx[6] = {3000, 3000, 3000, 200, 200, 200};
Drfl.task_compliance_ctrl(stx);
```

4.10.5 CDRFLEx.release_compliance_ctrl

Features

This function terminates compliance control and begins position control at the current position.

Parameter

None

Return

Value	Description
0	Failed
1	Success

Example

```
float p0[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(p0, 60, 30);
float stx[6] = {3000, 3000, 3000, 200, 200, 200};
Drfl.task_compliance_ctrl(stx);
float stx2[6] = {1, 2, 3, 4, 5, 6};
Drfl.set_stiffnessx(stx2);
Drfl.release_compliance_ctrl();
```

4.10.6 CDRFLEx.set_stiffnessx

Features

This function sets the stiffness value based on the global coordinate (refer to set_ref_coord). The linear transition from the current or default stiffness is performed during the time given as STX. The user-defined ranges of the translational stiffness and rotational stiffness are 0-20000N/m and 0-400Nm/rad, respectively.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetStiffness	float[6]	-	Three translational stiffnesses Three rotational stiffnesses
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type
fTargetTime	float	0	Stiffness varying time [sec] Range: 0 - 1.0 * Linear transition during the specified time

Return

Value	Description
0	Failed
1	Success

Example

```
float p0[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(p0, 60, 30);
float stx[6] = {3000, 3000, 3000, 200, 200, 200};
Drfl.task_compliance_ctrl(stx);
float stx2[6] = {1, 2, 3, 4, 5, 6};
Drfl.set_stiffnessx(stx2);
Drfl.release_compliance_ctrl();
```

4.10.7 CDRFLEx.calc_coord

Features

This function returns a new user cartesian coordinate system by using up to 4 input poses ([x1]~[x4]), input mode [mod] and the reference coordinate system [ref]. The input mode is only valid when the number of input robot poses is 2.

In the case that the number of input poses is 1, the coordinate system is calculated using the position and orientation of x1.

In the case that the number of input poses is 2 and the input mode is 0, X-axis is defined by the direction from x1 to x2, and Z-axis is defined by the projection of the current Tool-Z direction onto the plane orthogonal to the x-axis. The origin is the position of x1.

In the case that the number of input poses is 2 and the input mode is 1, X-axis is defined by the direction from x1 to x2, and Z-axis is defined by the projection of the z direction of x1 onto the plane orthogonal to the X-axis. The origin is the position of x1.

In the case that the number of input poses is 3, X-axis is defined by the direction from x1 to x2. If a vector v is the direction from x1 to x3, Z-axis is defined by the cross product of X-axis and v (X-axis cross v). The origin is the position of x1.

In the case that the number of input poses is 4, the definition of axes is identical to the case that the number of input poses is 3, but the origin is the position of x4.

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
nCnt	unsigned short	-	input count
nInputMode	unsigned short	-	input mode (only valid when the number of input poses is 2) 0: defining z-axis based on the current Tool-z direction 1: defining z-axis based on the z direction of x1

Parameter Name	Data Type	Default Value	Description
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type
fTargetPos1	float[6]		Target task location for six axes
fTargetPos2	float[6]		Target task location for six axes
fTargetPos3	float[6]		Target task location for six axes
fTargetPos4	float[6]		Target task location for six axes

Return

Value	Description
enum.ROBOT_POSE	Refer to the Definition of Constant and Enumeration Type

Example

```
float pos1[6] = {500, 30, 500, 0, 0, 0};
float pos2[6] = {400, 30, 500, 0, 0, 0};
float pos3[6] = {500, 30, 600, 45, 180, 45};
float pos4[6] = {500, -30, 600, 0, 180, 0};
ROBOT_POSE* pose_user1 = Drfl.calc_coord(4, 0, COORDINATE_SYSTEM_BASE, pos1, pos2, pos3, pos4);
for (int i=0; i<NUM_TASK; i++)
{
    cout << pose_user1->_fPosition[i] << endl;
}
```

4.10.8 CDRFLEx.set_user_cart_coord

CDRFLEx.set_user_cart_coord(iReqId, fTargetPos, eTargetRef)

Features

This function set a new user cartesian coordinate system using input pose [fTargetPos] and reference coordinate system [eTargetRef]. Up to 20 user coordinate systems can be set including the coordinate systems set within Workcell Item. Since the coordinate system set by this function is removed when the program is terminated, setting new coordinate systems within Workcell Item is recommended for maintaining the coordinate information.

Parameter

Parameter Name	Data Type	Default Value	Description
iReqId	int	-	Coordinate ID 0 : Auto creation 101 ~ 120 : Manual Creation
fTargetPos	float[6]	-	Target task location for six axes
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
int	Successful coordinate setting Set coordinate ID (101 - 200)

Example

```
float pos1[6] = {500, 30, 500, 0, 0, 0};
int id = Drfl.set_user_cart_coord(0, pos1, COORDINATE_SYSTEM_BASE);
```

CDRFLEx.set_user_cart_coord(fTargetPos, fTargetOrg, eTargetRef)

Features

This function sets a new user cartesian coordinate system using [fTargetPos[0]], [fTargetPos[1]]], and [fTargetPos[2]] based on ref coordinate system [eTargetRef]. Creates a user coordinate system with ux, uy, and uz as the vector for each axis and origin position is the position of [pos] based on [ref]. 1)ux is defined as the unit vector of x1x2 , uz is defined as the unit vector defined by the cross product of x1x2 and x1x3 (x1x2 cross x1x3). uy is can be determined by right hand rule (uz cross ux). Up to 20 user coordinate systems can be set including the coordinate systems set within Workcell Item. Since the coordinate system set by this function is removed when the program is terminated, setting new coordinate systems within Workcell Item is recommended for maintaining the coordinate information.

Before M2.0.2 software version, ux is the unit vector of x2x1

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[3][6]	-	Target task location for six axes #1 Target task location for six axes #2 Target task location for six axes #3
fTargetOrg	float[3]	-	origin position
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
int	Successful coordinate setting Set coordinate ID (101 - 200)

Example

```
float x1[6] = {0,500,700,0,0,0};
float x2[6] = {500,0,700,0,0,0};
float x3[6] = {300,100,500,0,0,0};
```

```
float org[3] = {10, 20, 30};
float x[3][6] = {{0,500,700,0,0,0}, {500,0,700,0,0,0},
{300,100,500,0,0,0}};
Drfl.set_user_cart_coord(x, org, COORDINATE_SYSTEM_BASE);
```

CDRFLEx.set_user_cart_coord(fTargetVec, fTargetOrg, eTargetRef)

Features

This function sets a new user cartesian coordinate system using [fTargetVec[0]] and [fTargetVec[1]] based on [eTargetRef] coordinate system. The origin position the position of [fTargetOrg] based on the [eTargetRef] coordinate while the direction of x-axis and y-axis bases are given in the vectors u1 and v1, respectively. Other directions are determined by u1 cross v1. If u1 and v1 are not orthogonal, v1', that is perpendicular to u1 on the surface spanned by u1 and v1, is set as the vector in the y-axis direction. Up to 20 user coordinate systems can be set including the coordinate systems set within Workcell Item. Since the coordinate system set by this function is removed when the program is terminated, setting new coordinate systems within Workcell Item is recommended for maintaining the coordinate information.

This function is only available in M2.5 hot fix version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetVec	float[2][3]	-	vector #1 vector #2
fTargetOrg	float[3]	-	origin position
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
int	Successful coordinate setting Set coordinate ID (101 - 200)

Example

```
float vec[2][3] = {{-1, 1, 1}, {1, 1, 0}};
```

```
float org[3] = {370.9, -419.7, 651.5};

int user = Drfl.set_user_cart_coord(vec, org);
```

4.10.9 CDRFLEx.overwrite_user_cart_coord

Features

This function changes the pose and reference coordinate system of the requested user coordinate system [iReqId] with the [fTargetPos] and [eTargetRef], respectively.

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
bTargetUpdate	bool	-	0 : local update 1 : global update(with TP)
iReqId	int	-	coordinate ID
fTargetPos	float[6]	-	Target task location for six axes
eTargetRef	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_BASE	Refer to the Definition of Constant and Enumeration Type

When the coordinate system change (bTargetUpdate) variable is 0, the user coordinate system must be changed and maintained only when the program is executed.

Return

Value	Description
int	Successful coordinate setting Set coordinate ID (101 - 200)

Example

```
float pos1[6] = {30, 40, 50, 0, 0, 0};
```

```
int pose_user1 = Drfl.set_user_cart_coord(0, pos1, COORDINATE_SYSTEM_BASE);

float pos2[6] = {100, 150, 200, 45, 180, 0};
int result = Drfl.overwrite_user_cart_coord(0, pose_user1, pos2);

cout << result << endl;
```

4.10.10 CDRFLEx.get_user_cart_coord

Features

This function returns the pose and reference coordinate system of the requested user coordinate system [iReqId].

This function is only available in M2.5 version or higher.

Parameter

Parameter Name	Data Type	Default Value	Description
iReqId	int	-	coordinate ID

Return

Value	Description
USER_COORDINATE	Refer to definition of structure

Example

```
float pos[6] = {10, 20, 30, 0, 0, 0};
int id = Drfl.set_user_cart_coord(0, pos);
USER_COORDINATE *temp = Drfl.get_user_cart_coord(id);
```

4.10.11 CDRFLEx.set_desired_force

Features

This function defines the target force, direction, translation time, and mode for force control based on the global coordinate(refer to set_ref_coord).

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetForce	float[6]	-	Three translational target forces Three rotational target moments
iTargetDirection	unsigned char[6]	-	Force control in the corresponding direction if 1 Compliance control in the corresponding direction if 0
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type
fTargetTime	float	0	Transition time of target force to take effect [sec] Range: 0 - 1.0
eForceMode	enum.FORCE_MODE	FORCE_MODE_ABSOLUTE	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.set_ref_coord(COORDINATE_SYSTEM_TOOL);
float x0[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(x0, 60, 30);
float stx[6] = {500, 500, 500, 100, 100, 100};
```

```
Drfl.task_compliance_ctrl(stx);
float fd[6] = {0, 0, 0, 0, 0, 10};
unsigned char fctrl_dir[6] = {0, 0 ,1, 0, 0, 1};
Drfl.set_desired_force(fd, fctrl_dir);
```

4.10.12 CDRFLEx.release_force

Features

This function reduces the force control target value to 0 through the time value and returns the task space to adaptive control.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetTime	float	0	Time needed to reduce the force Range: 0 - 1.0

Return

Value	Description
0	Failed
1	Success

Example

```
Drfl.set_ref_coord(COORDINATE_SYSTEM_TOOL);
float x0[6] = {0, 0, 90, 0, 90, 0};
Drfl.movej(x0, 60, 30);
float stx[6] = {500, 500, 500, 100, 100, 100};
Drfl.task_compliance_ctrl(stx);
float fd[6] = {0, 0, 0, 0, 0, 10};
unsigned char fctrl_dir[6] = {0, 0 ,1, 0, 0, 1};
Drfl.set_desired_force(fd, fctrl_dir);
float x1[6] = {0, 500, 700, 0, 180, 0};
float velx[2] = {60, 60};
float accx[2] = {30, 30};
Drfl.MoveL(x1, velx, accx);
```

```
Drfl.release_force(0.5);
Drfl.release_compliance_ctrl();
```

4.10.13 CDRFLEx.check_position_condition_abs

Features

This function checks the status of the given position. This condition can be repeated with the while or if statement. Axis and pos of input paramets are based on the ref coordinate(eTargetRef).

In case of eForceReference = COORDINATE_SYSTEM_TOOL, pos(fTargetPos) should be defined in BASE coordinate.

→ Refer to the check_position_condition_rel function for relative movement criteria

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	the condition is False

Example

```
bool CON1 = Drfl.check_position_condition_abs(FORCE_AXIS_X, -5, 0, COORDINATE_SYSTEM_WORLD);
bool CON2 = Drfl.check_position_condition_abs(FORCE_AXIS_Y, -10000, 700);
bool CON3 = Drfl.check_position_condition_abs(FORCE_AXIS_Z, -10, -5);
bool CON4 = Drfl.check_position_condition_abs(FORCE_AXIS_Z, 30, -10000);
bool CON5 = Drfl.check_position_condition_abs(FORCE_AXIS_Z, -10, -5, COORDINATE_SYSTEM_BASE);
bool CON6 = Drfl.check_position_condition_abs(FORCE_AXIS_Z, -10, -5);
```

4.10.14 CDRFLEx.check_position_condition_rel

Features

This function checks the status of the given position. This condition can be repeated with the while or if statement. Axis and pos of input paramets are based on the ref coordinate(eTargetRef).

In case of eForceReference = COORDINATE_SYSTEM_TOOL, pos(fTargetPos) should be defined in BASE coordinate.

→ Refer to check_position_condition_abs function for absolute movement criteria

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
fTargetPos	float[6]	-	Target task location for six axes
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
float posx1[6] = {400, 500, 800, 0, 180, 0};
bool CON7 = Drfl.check_position_condition_rel(FORCE_AXIS_Z, -10, -5, posx1, COORDINATE_SYSTEM_TOOL);
```

4.10.15 CDRFLEx.check_position_condition

Features

This function checks the status of the given position. This condition can be repeated with the while or if statement. Axis and pos of input paramets are based on the ref coordinate(eTargetRef).

In case of eForceReference = COORDINATE_SYSTEM_TOOL, pos(fTargetPos) should be defined in BASE coordinate.

If eMode is MOVE_MODE_RELATIVE, check_position_condition_rel is called, and if MOVE_MODE_ABSOLUTE, check_position_condition_abs is called.

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target task location for six axes
eMode	enum.MOVE_MODE	MOVE_MODE_ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eForceReference	COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
float posx1[6] = {400, 500, 800, 0, 180, 0};
bool CON7 = Drfl.check_position_condition(FORCE_AXIS_Z, -10, -5, posx1, MOVE_MODE_ABSOLUTE);
```

4.10.16 CDRFLEx.check_force_condition

Features

This function checks the status of the given force. It disregards the force direction and only compares the sizes. This condition can be repeated with the while or if statement. Measuring the force, eForceAxis is based on the ref coordinate(eTargetRef) and measuring the moment, eForceAxis is based on the tool coordinate.

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
bool fcon1 = Drfl.check_force_condition(FORCE_AXIS_Z, 5, 10, COORDINATE_SYSTEM_WORLD);
bool fcon2;
bool pcon1;
while(true){
    fcon2 = Drfl.check_force_condition(FORCE_AXIS_C, 30, -10000);
    pcon1 = Drfl.check_position_condition_abs(FORCE_AXIS_X, 0, 0.1);
    if(fcon2 && pcon1)
    {
        break;
    }
}
```

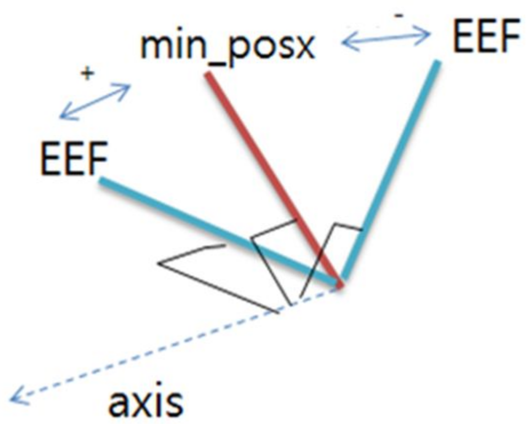
4.10.17 CDRFLEx.check_orientation_condition

CDRFLEx.check_orientation_condition(eForceAxis, fTargetMin, fTargetMax, eForceReference)

Features

This function checks the difference between the current pose and the specified pose of the robot end effector. It returns the difference between the current pose and the specified pose in rad with the algorithm that transforms it to a rotation matrix using the “AngleAxis” technique. It returns True if the difference is positive (+) and False if the difference is negative (-). It is used to check if the difference between the current pose and the rotating angle range is + or -. For example, the function can use the direct teaching position to check if the difference from the current position is + or - and then create the condition for the orientation limit. This condition can be repeated with the while or if statement.

- Setting Min only: True if the difference is + and False if -
- Setting Min and Max: True if the difference from min is - while the difference from max is + and False if the opposite.
- Setting Max only: True if the maximum difference is + and False otherwise



Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float[6]	-	Minimum value
fTargetMax	float[6]	-	Maximum value

Parameter Name	Data Type	Default Value	Description
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	The condition is False

Example

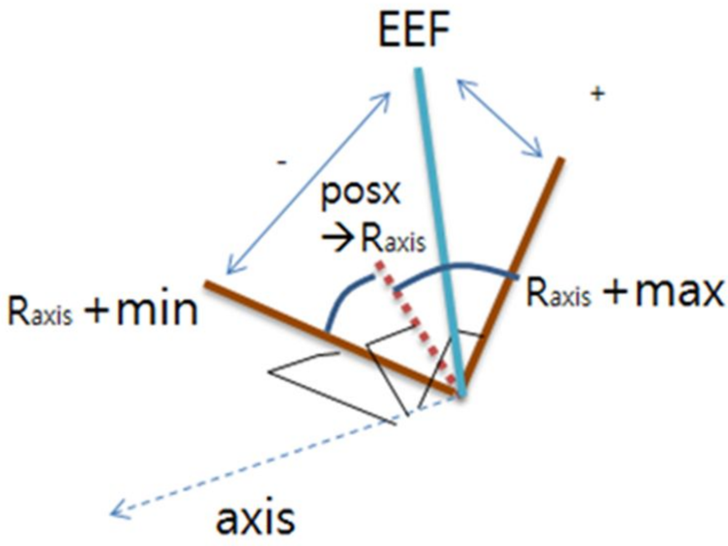
```
float posx1[6] = {400, 500, 800, 0, 180, 30};
float posx2[6] = {400, 500, 500, 0, 180, 60};
bool con1 = Drfl.check_orientation_condition(FORCE_AXIS_C, posx1, posx2);
```

CDRFLEx.check_orientation_condition(eForceAxis, fTargetMin, fTargetMax, fTargetPos, eForceReference)

Features

This function checks the difference between the current pose and the rotating angle range of the robot end effector. It returns the difference (in rad) between the current pose and the rotating angle range with the algorithm that transforms it to a rotation matrix using the “AngleAxis” technique. It returns True if the difference is positive (+) and False if the difference is negative (-). It is used to check if the difference between the current pose and the rotating angle range is + or -. For example, the function can be used to set the rotating angle range to min and max at any reference position, and then determine the orientation limit by checking if the difference from the current position is + or -. This condition can be repeated with the while or if statement.

- Setting Min only: True if the difference is + and False if -
- Setting Min and Max: True if the difference from min is - while the difference from max is + and False if the opposite.
- Setting Max only: True if the maximum difference is + and False otherwise



> **Note**
>
> Range of rotating angle: This means the relative angle range (min, max) based on the specified axis from a given position based on the ref coordinate.

Parameter

Parameter Name	Data Type	Default Value	Description
eForceAxis	enum.FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
fTargetPos	float[6]	-	Target task location for six axes
eForceReference	enum.COORDINATE_SYSTEM	COORDINATE_SYSTEM_TOOL	Refer to the Definition of Constant and Enumeration Type

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
float posx1[6] = {400, 500, 800, 0, 180, 30};
bool con1 = Drfl.check_orientation_condition(FORCE_AXIS_C, 0, 5, posx1);
```

4.10.18 CDRFLEx.coord_transform

Features

This function transforms given task position expressed in reference coordinate, 'eInCoordSystem' to task position expressed in reference coordinate, 'eOutCoordSystem'. It returns transformed task position. It supports calculation of coordinate transformation for the following cases.

- (eInCoordSystem) world reference coordinate → (eOutCoordSystem) world reference coordinate
- (eInCoordSystem) world reference coordinate → (eOutCoordSystem) base reference coordinate
- (eInCoordSystem) world reference coordinate → (eOutCoordSystem) tool reference coordinate
- (eInCoordSystem) world reference coordinate → (eOutCoordSystem) user reference coordinate
- (eInCoordSystem) base reference coordinate → (eOutCoordSystem) world reference coordinate
- (eInCoordSystem) base reference coordinate → (eOutCoordSystem) base reference coordinate
- (eInCoordSystem) base reference coordinate → (eOutCoordSystem) tool reference coordinate
- (eInCoordSystem) base reference coordinate → (eOutCoordSystem) user reference coordinate
- (eInCoordSystem) tool reference coordinate → (eOutCoordSystem) world reference coordinate
- (eInCoordSystem) tool reference coordinate → (eOutCoordSystem) base reference coordinate
- (eInCoordSystem) tool reference coordinate → (eOutCoordSystem) tool reference coordinate
- (eInCoordSystem) tool reference coordinate → (eOutCoordSystem) user reference coordinate
- (eInCoordSystem) user reference coordinate → (eOutCoordSystem) world reference coordinate
- (eInCoordSystem) user reference coordinate → (eOutCoordSystem) base reference coordinate
- (eInCoordSystem) user reference coordinate → (eOutCoordSystem) tool reference coordinate
- (eInCoordSystem) user reference coordinate → (eOutCoordSystem) user reference coordinate

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetpos	float[6]	-	Target task location for six axes
eInCoordSystem	float	-	Minimum value
eOutCoordSystem	float	-	Maximum value

Return

Value	Description
float[6]	Target task location for six axes

Example

```
float base_pos[6] = {400, 500, 800, 0, 180, 15};
ROBOT_POSE* tool_pos;
tool_pos = Drfl.coord_transform(base_pos, COORDINATE_SYSTEM_BASE,
COORDINATE_SYSTEM_TOOL);
```

4.10.19 CDRFLEx.set_palletizing_mode

Features

During palletizing application motion, path tracking and velocity can be maintained around the wrist singularity point using this function. there is no instability in wrist singular region when B in motion command is set to 0deg or 180deg.

Parameter

Parameter Name	Data Type	Default Value	Description
iMode	unsigned char	-	0 : disable 1 : enable

Return

Value	Description
0	fail
1	success

Example

```
Drfl.set_palletizing_mode(1);
Drfl.set_palletizing_mode(0);
```

4.10.20 CDRFLEx.query_modbus_data_list

Features

This function query all currently registered Modbus data (TCP / RTU) information.

Parameter

None

Return

Value	Description
struct.MODBUS_DATA_LIST	Refer to definition of structure

Example

```
LPMODBUS_DATA_LIST tParam = Drfl.query_modbus_data_list();
for (int i = 0; i < tParam->_nCount; i++) {
    if (tParam->_tRegister[i]._iType == 0) // TCP : 0
    {
        cout << tParam->_tRegister[i]._tData._tcp._iRegValue << endl;
    }
    else { // RTU : 1
        cout << tParam->_tRegister[i]._tData._rtu._iRegValue << endl;
    }
}
```

5 Realtime Control

5.1 Realtime Control Introduction

Real-time External Control is a UDP/IP-based communication api for users who want to directly control the robot arm from an external PC through the Doosan controller. It operates independently of the existing TCP/IP-based communication api, and sends and receives input data (external controller → robot controller) and output data (robot controller → external controller) required for real-time control up to 1 kHz at a set period, and separately You can send servo control commands (servoj_rt, servol_rt, speedj_rt, speedl_rt, torque_rt).

5.1.1 Version

The real-time external control version is managed according to the input/output data structure version. Although each input/output data structure may change as the SW version increases, backward compatibility is supported by matching the I/O data version developed by the external controller.

5.2 Robot Connection Function 1

5.2.1 CDRFLEx.connect_rt_control

Features

This function connects to robot controller via Real-time External Control.

Parameter

Parameter Name	Data Type	Default Value	Description
strIPAddr	string	192.168.137.100	IP Address
usPort	unsigned int	12345	Port Number

Note

- Real-time external control uses udp/ip communication.
- This channel is independent from the original tcp/ip api. It doesn't care control authority.
- For the integrated controller (v3), versions v3.2.2 or below can be connected to 192.168.137.50.

Caution

- Currently, 1 and 1 communication is allowed.

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
Drfl.connect_rt_control(); //connect udp. if your controller version is below v3.2.2, ip address will be 192.168.137.50.
```

5.2.2 CDRFLEx.disconnect_rt_control

Features

This function disconnects real-time external control.

Note

- Reset all settings when disconnecting.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
Drfl.disconnect_rt_control(); //disconnect udp
```

5.3 Information Lookup Function

5.3.1 CDRFLEx.get_rt_control_input_version_list

Features

This function provides real-time external control version list for about input data.

Parameter

None

Return

Value	Description
String	Input Data Version List. Comma(,) separated.

Version List

Input Data Version	First Updated Controller Version
v1.0	V2.9

Example

```
string version_list = Drfl.get_rt_control_input_version_list();
cout << version_list << endl;
```

5.3.2 CDRFLEx.get_rt_control_output_version_list

Features

This function provides real-time external control version list for about output data.

Parameter

None

Return

Value	Description
String	Output Data Version List. Comma(,) separated.

Version List

Input Data Version	First Updated Controller Version
v1.0	V2.9

Example

```
string version_list = Drfl.get_rt_control_output_version_list();
cout << version_list << endl
```

5.3.3 CDRFLEx.get_rt_control_input_data_list

Features

Returns a list of input data supported by a specific version.

Parameter

Parameter Name	Data Type	Default Value	Description
strVersion	String	-	Input Data Version

Return

Value	Description
string	Input Data List. Comma(,) separated.

Input Data List

Parameter Name	Data Type	Description	First Updated Controller Version
external_force_torque	float[6]	external force/torque sensor	v1.0
external_digital_input	uint16	external digital input(16 channel)	v1.0
external_digital_output	uint16	external digital output(16 channel)	v1.0
external_analog_input	float[6]	external analog input(6 channel)	v1.0
external_analog_output	float[6]	external analog output(6 channel)	v1.0

Example

```
string data_list = Drfl.get_rt_control_input_data_list();
cout << data_list << endl;
```

5.3.4 CDRFLEx.get_rt_control_output_data_list

Features

Returns a list of output data supported by a specific version.

Parameter

Parameter Name	Data Type	Default Value	Description
strVersion	string	-	Output Data Version

Return

Value	Description
string	Output Data List. Comma(,) separated.

Output Data List

Parameter Name	Data Type	Description	First Updated Controller Version
time_stamp	double	timestamp at the data of data acquisition [s]	v1.0
actual_joint_position	float[6]	actual joint position from incremental encoder at motor side(used for control) [deg]	v1.0
actual_joint_position_abs	float[6]	actual joint position from absolute encoder at link side (used for exact link position) [deg]	v1.0
actual_joint_velocity	float[6]	actual joint velocity from incremental encoder at motor side [deg/s]	v1.0
actual_joint_velocity_abs	float[6]	actual joint velocity from absolute encoder at link side [deg/s]	v1.0
actual_tcp_position	float[6]	actual robot tcp position w.r.t. base coordinates: (x, y, z, a, b, c), where (a, b, c) follows Euler ZYZ notation [mm, deg]	v1.0
actual_tcp_velocity	float[6]	actual robot tcp velocity w.r.t. base coordinates [mm, deg/s]	v1.0
actual_flange_position	float[6]	actual robot flange position w.r.t. base coordinates: (x, y, z, a, b, c), where (a, b, c) follows Euler ZYZ notation [mm, deg]	v1.0
actual_flange_velocity	float[6]	robot flange velocity w.r.t. base coordinates [mm, deg/s]	v1.0
actual_motor_torque	float[6]	actual motor torque applying gear ratio = gear_ratio * current2torque_constant * motor current [Nm]	v1.0
actual_joint_torque	float[6]	estimated joint torque by robot controller [Nm]	v1.0

Parameter Name	Data Type	Description	First Updated Controller Version
raw_joint_torque	float[6]	calibrated joint torque sensor data	v1.0
raw_force_torque	float[6]	calibrated force torque sensor data w.r.t. flange coordinates [N, Nm]	v1.0
external_joint_torque	float[6]	estimated joint torque [Nm]	v1.0
external_tcp_force	float[6]	estimated tcp force w.r.t. base coordinates [N, Nm]	v1.0
target_joint_position	float[6]	target joint position [deg]	v1.0
target_joint_velocity	float[6]	target joint velocity [deg/s]	v1.0
target_joint_acceleration	float[6]	target joint acceleration [deg/s^2]	v1.0
target_motor_torque	float[6]	target motor torque [Nm]	v1.0
target_tcp_position	float[6]	target tcp position w.r.t. base coordinates: (x, y, z, a, b, c), where (a, b, c) follows Euler ZYZ notation [mm, deg]	v1.0
target_tcp_velocity	float[6]	target tcp velocity w.r.t. base coordinates [mm, deg/s]	v1.0
jacobian_matrix	float[6][6]	jacobian matrix=J(q) w.r.t. base coordinates	v1.0
gravity_torque	float[6]	gravity torque=g(q) [Nm]	v1.0
coriolis_matrix	float[6][6]	coriolis matrix=C(q) [Nm.s]	v1.0
mass_matrix	float[6][6]	mass matrix=M(q)+B [Nm.s^2]	v1.0
solution_space	uint8	robot configuration	v1.0
singularity	float	minimum singular value	v1.0

Parameter Name	Data Type	Description	First Updated Controller Version
operation_speed_rate	float	current operation speed rate(1~100 %)	v1.0
joint_temperature	float[6]	joint temperature(celsius)	v1.0
controller_digital_input	uint16	controller digital input(16 channel)	v1.0
controller_digital_output	uint16	controller digital output(16 channel)	v1.0
controller_analog_input_type	uint8	controller analog input type(2 channel)	v1.0
controller_analog_input	float[2]	controller analog input(2 channel)	v1.0
controller_analog_output_type	uint8	controller analog output type(2 channel)	v1.0
controller_analog_output	float[2]	controller analog output(2 channel)	v1.0
flange_digital_input	uint8	flange digital input (A-Series: 2 channel, M/H-Series: 6 channel)	v1.0
flange_digital_output	uint8	flange digital input (A-Series: 2 channel, M/H-Series: 6 channel)	v1.0
flange_analog_input	float[4]	flange analog input (A-Series: 2 channel, M/H-Series: 4 channel)	v1.0
external_encoder_strobe_count	uint8[2]	strobe count (increased by 1 when detecting setting edge)	v1.0
external_encoder_count	uint32[2]	external encoder count	v1.0
goal_joint_position	float[6]	final goal joint position (reserved)	v1.0
goal_tcp_position	float[6]	final goal tcp position (reserved)	v1.0

Parameter Name	Data Type	Description	First Updated Controller Version
robot_mode	uint8	ROBOT_MODE_MANUAL(0), ROBOT_MODE_AUTONOMOUS(1), ROBOT_MODE_MEASURE(2)	v1.0
robot_state	uint8	STATE_INITIALIZING(0), STATE_STANDBY(1), STATE_MOVING(2), STATE_SAFE_OFF(3), STATE_TEACHING(4), STATE_SAFE_STOP(5), STATE_EMERGENCY_STOP(6), STATE_HOMMING(7), STATE_RECOVERY(8), STATE_SAFE_STOP2(9), STATE_SAFE_OFF2(10)	v1.0
control_mode	uint16	position control mode, torque mode	v1.0

Example

```
string data_list = Drfl.get_rt_control_output_data_list();
cout << data_list << endl;
```

5.4 Configuration Function

5.4.1 CDRFLEx.set_rt_control_input

Features

Set the input data (external controller → robot controller) communication configuration supported by real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
strVersion	string	-	Input Data Version

Parameter Name	Data Type	Default Value	Description
fPeriod	float	-	Communication Period (sec). Range: 0.001~1 [sec]
nLossCnt	int	-	In succession, if the input data or the servo control command is lost due to over the set count, the real-time control connection is disconnected. However, if set to -1, Loss Count is not checked.

Note

- fPeriod currently supports 0.001 to 1 sec.

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
// Connect and configure input data
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
drfl.set_rt_control_input(version, period, losscount);
```

5.4.2 CDRFLEx.set_rt_control_output

Features

Set the output data (robot controller → external controller) communication configuration supported by real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
strVersion	string	-	Output Data Version
fPeriod	float	-	Communication Period (sec). Range: 0.001~1 [sec]
nLossCnt	int	-	Loss Count

Note

- fPeriod currently supports 0.001 to 1 sec.
- Currently, nLossCnt is not used.

Return

Value	Description
1	The condition is True
0	The condition is False

Example

```
// Connect and configure output data
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
drfl.set_rt_control_output(version, period, losscount);
```

5.5 Operation Function

5.5.1 CDRFLEx.start_rt_control

Features

Starts sending/receiving the set input/output data.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
// Connect and start
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();
```

5.5.2 CDRFLEx.stop_rt_control

Features

Finishes sending/receiving the set input/output data.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
// Connect and start/stop
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();
// do something
drfl.stop_rt_control();
```

5.5.3 CDRFLEx. set_on_rt_monitoring_data

Features

This is a callback function called when the robot controller output data is received from the external controller.

Note

- Non real-time functions such as printf must not be called faster than 30Hz inside this function.

Parameter

None

Return

Value	Description
0	Error
1	Success

Example

```
// callback function
void OnRTMonitoringData(LPRT_OUTPUT_DATA_LIST tData)
{
    return;
    static int td = 0;
    if (td++==1000) {
```

```
        td = 0;
        printf("timestamp : %.3f\n", tData->time_stamp);
        printf("actual_joint_position : %f %f %f %f %f %f\n", tData->actual_joint_position[0], tData->actual_joint_position[1], tData->actual_joint_position[2], tData->actual_joint_position[3], tData->actual_joint_position[4], tData->actual_joint_position[5]);
        printf("actual_motor_torque : %f %f %f %f %f %f\n", tData->actual_motor_torque[0], tData->actual_motor_torque[1], tData->actual_motor_torque[2], tData->actual_motor_torque[3], tData->actual_motor_torque[4], tData->actual_motor_torque[5]);
        printf("actual_grav_torque : %f %f %f %f %f %f\n", tData->gravity_torque[0], tData->gravity_torque[1], tData->gravity_torque[2], tData->gravity_torque[3], tData->gravity_torque[4], tData->gravity_torque[5]);
        printf("target torque : %f %f %f %f %f %f\n", tData->target_motor_torque[0], tData->target_motor_torque[1], tData->target_motor_torque[2], tData->target_motor_torque[3], tData->target_motor_torque[4], tData->target_motor_torque[5]);
    }
}

// main.cpp
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
Drfl.set_on_rt_monitoring_data(OnRTMonitoringData);
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();
```

5.5.4 CDRFLEx.read_data_rt

Features

This function reads the real-time output data from the robot controller.

Parameter

None

Return

Value	Description
LPRT_OUTPUT_DATA_LIST	Output Data List Pointer.

Example

```cpp
// main.cpp
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
Drfl.set_on_rt_monitoring_data(OnRTMonitoringData);
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();

float q[NUMBER_OF_JOINT] = {0.0, };
float q_dot[NUMBER_OF_JOINT] = {0.0, };
float trq_g[NUMBER_OF_JOINT] = {0.0, };
while (1)
{
    time=(++count)*st;

    memcpy(q, drfl.read_data_rt()->actual_joint_position, sizeof(float)*6);
    memcpy(q_dot, drfl.read_data_rt()->actual_joint_velocity, sizeof(float)*6;
    memcpy(trq_g, drfl.read_data_rt()->gravity_torque, sizeof(float)*6);
    memcpy(trq_d, trq_g, sizeof(float)*6);

    drfl.torque_rt(trq_d, 0);

    if(time > plan1.time)
        {
            time=0;
            Drfl.stop(STOP_TYPE_SLOW);
            break;
        }
        rt_task_wait_period(NULL);
}
```

5.5.5 CDRFLEx.write_data_rt

Features

This function writes the real-time input data. (external controller → robot controller) It is for the purpose of receiving sensors, DIO, AIO, and other commands connected to an external controller, and although there is no current use, it will be utilized through collaboration with external corporations.

Parameter

Parameter Name	Data Type	Default Value	Description
fExternalForceTorque	float[6]	-	External Force Torque
iExternalDI	unsigned short	-	External Digital Input
iExternalDO	unsigned short	-	External Digital Output
fExternalAnalogInput	float[6]	-	External Analog Input(6 channel)
fExternalAnalogOutput	float[6]	-	external analog output(6 channel)

Return

Value	Description
0	Error
1	Success

Example

```
fExternalForceTorque[6] = {100, 100, 100, 100, 100, 100};
int iExternalDI = 1;
int iExternalDO = 2;
float fExternalAnalogInput[6] = {100, 100, 100, 100, 100, 100};
float fExternalAnalogOutput[6] = {0, 0, 0, 0, 0, 0};

Drfl.write_data_rt(fExternalForceTorque, iExternalDI, iExternalDO,
fExternalAnalogInput, fExternalAnalogOutput);
```

5.6 Servo Motion Function

The servo motion function does not use the motion provided by Doosan. It is a function that can be used by implementing the motion/force control algorithm in an external controller. When a real-time external controller system is built and an algorithm is implemented in the controller, the robot controller is a concept that processes it by simple transmission or interpolation.

5.6.1 Note

- Servo motion functions of real-time external control (ex. servoj_rt, servol_rt, speedj_rt, speedl_rt) must be built in a real-time environment by using real-time OS in the external controller to ensure real-time performance of the entire system. It also makes sense if the intermediate profile can be calculated externally. If a real-time environment cannot be built or the intermediate profile cannot be calculated externally, use the servo motion functions of general motion (ex. servoj, servol, speedj, speedl) in the existing api based on TCP/IP. In the servo motion function of general motion, if only the target value is changed in real time, the robot controller calculates the intermediate profile.

5.6.2 CDRFLEx.set_velj_rt

Features

Set the global joint speed limit value used in the servo motion of real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
vel	float[6]	-	Joint Velocity Limit [deg/s]

Note

- If the velocity while moving using the servo motion function is greater than the global velocity limit, Info is generated.

Return

Value	Description
0	Error
1	Success

Example

```
float vel[6] = {100, 100, 100, 100, 100, 100};
Drfl.set_velj_rt(vel);
```

5.6.3 CDRFLEx.set_accj_rt

Features

Set the global joint acceleration limit value used in the servo motion of real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
acc	float[6]	-	Joint Acceleration Limit [deg/s]

Note

- If the acceleration while moving using the servo motion function is greater than the global acceleration limit, Info is generated.

Return

Value	Description
0	Error
1	Success

Example

```
float acc[6] = {100, 100, 100, 100, 100, 100};
Drfl.set_accj_rt(acc);
```

5.6.4 CDRFLEx.set_velx_rt

Features

Set the global task velocity limit value used in the servo motion of real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
fTransVel	float	-	Task Linear Velocity Limit [mm/s]
fRotationVel	float	-	Task Rotational Velocity Limit [deg/s], if None Value(-10000) entered, auto-calculated by Task Linear Velocity Limit

> **Note**
>
> - If the velocity while moving using the servo motion function is greater than the global velocity limit, Info is generated.

Return

Value	Description
0	Error
1	Success

Example

```
float vel[6] = {100, 100, 100, 100, 100, 100};
Drfl.set_velx_rt(vel);
```

5.6.5 CDRFLEx.set_accx_rt

Features

Set the global task acceleration limit value used in the servo motion of real-time external control.

Parameter

Parameter Name	Data Type	Default Value	Description
fTransAcc	float	-	Task Linear Acceleration Limit [mm/s2]

Parameter Name	Data Type	Default Value	Description
fRotationAcc	float	-	Task Rotational Acceleration Limit [deg/s2]. if None Value(-10000) entered, auto-calculated by Task Linear Acceleration Limit

Note

- If the acceleration while moving using the servo motion function is greater than the global acceleration limit, Info is generated.

Return

Value	Description
0	Error
1	Success

Example

```
float acc[6] = {100, 100, 100, 100, 100, 100};
Drfl.set_accx_rt(acc);
```

5.6.6 CDRFLEx.servoj_rt

Features

This is a function that controls the joint position from an external controller.

Caution

- **The current servoj_rt command is premature.** Jerky motion may appear depending on the communication situation, so it is recommended to tune the target time (fTargetTime) and use it with a slow response speed, or use the speedj_rt command to control the position.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target Joint Position [deg]
fTargetVel	float[6]	-	Target Joint Velocity [deg/s]. If None Values(-10000) entered, it is automatically calculated based on the target joint position you entered.
fTargetAcc	float[6]	-	Target Joint Acceleration [deg/s2]. If None Values(-10000) entered, it is automatically calculated based on the target joint position you entered.
fTargetTime	float	-	Target Time [s]

Note

- Asnyc command.
- The inner profile is interpolated to arrive at (fTargetPos, fTargetVel, fTargetAcc) at fTargetTime .
- If fTargetTime <= controller's control period (=1ms) is entered, it controls to the corresponding position without interpolation.
- If the next command is not received until arriving at fTargetPos, it decelerates based on the set global acceleration value.

Caution

- **The current servoj_rt command is premature, so jerky motion may come out. Depending on the system situation, set fTargetTime to be larger than the communication period and use it. In the current version, it is recommended to use a setting larger than fTargetTime >= 20 [ms] or use speedj_rt to control the position based on calling the servoj_rt command at an interval of 1 [ms].**
- In the current version, it is not linked with Operation Speed [%].

Return

Value	Description
0	Error
1	Success

Example

```
// main.cpp
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
Drfl.set_on_rt_monitoring_data(OnRTMonitoringData);
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();

float time = 0.0;
int count = 0;
float ratio = 20.0;
float q[NUMBER_OF_JOINT] = {0.0, };
float q_dot[NUMBER_OF_JOINT] = {0.0, };
float q_d[NUMBER_OF_JOINT] = {0.0, };
float q_dot_d[NUMBER_OF_JOINT] = {0.0, };
float integral_v_error[NUMBER_OF_JOINT] = {0.0, };
while (1)
{

    time=(++count)*period;

    // get current state
    memcpy(q, drfl.read_data_rt()->actual_joint_position, sizeof(float)*6);
    memcpy(q_dot, drfl.read_data_rt()->actual_joint_velocity, sizeof(float)*6);

    // make trajectory
    TrajectoryGenerator(q_d, q_dot_d, q_ddot_d); // Custom Trajectory Generation Function

    // or make target position and target velocity, acceleration can be omitted
    TrajectoryGenerator(q_d);
    for (int i=0; i<NUMBER_OF_JOINT; i++) {
        q_dot_d[i] = -10000;
        q_ddot_d[i] = -10000;
    }

    drfl.servoj_rt(q_d, q_dot_d, q_ddot_d, ratio*period); // Currently tuning ratio

    if(time > plan1.time)
    {
        time=0;
        Drfl.stop(STOP_TYPE_SLOW);
        break;
```

```
    }

    rt_task_wait_period(NULL); // RTOS function
}
```

5.6.7 CDRFLEx.servol_rt

Features

This is a function that controls the task position from an external controller.

Caution

- **The current servol_rt command is premature.** Jerky motion may appear depending on the communication situation, so it is recommended to tune the target time (fTargetTime) and use it with a slow response speed, or use the speedl_rt command to control the position.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target Task Position [mm, deg] (rotation: euler zyz)
fTargetVel	float[6]	-	Target Task Velocity. [mm/s, deg/s]. If None values(-10000) entered, it is automatically calculated based on the target task position you entered.
fTargetAcc	float[6]	-	Target Task Acceleration. [mm/s, deg/s] If None values(-10000) entered, it is automatically calculated based on the target task position you entered.
fTargetTime	float	-	Target Time [s]

Note

- Asnyc Command.
- The inner profile is interpolated to arrive at (fTargetPos, fTargetVel, fTargetAcc) at fTargetTime .
- If fTargetTime <= controller's control period (=1ms), control is performed at the corresponding velocity without interpolation.

- If the next command is not received until arriving at fTargetPos, it decelerates based on the set global acceleration limit value.

Caution

- **The current servol_rt command is premature, so jerky motion may come out. Depending on the system situation, set fTargetTime to be larger than the communication period and use it. In the current version, it is recommended to use a setting larger than fTargetTime >= 20 [ms] or use speedl_rt to control the position based on calling the servol_rt command at an interval of 1 [ms].**
- In the current version, it is not linked with Operation Speed [%].
- In the current version, it does not work with the force/compliance control function.
- In the current version, it does not work with DR_VAR_VEL among the singularity options. If set, it is automatically set to the DR_AVOID option.

Return

Value	Description
0	Error
1	Success

Example

```
float fTargetPos[6] = {1500, 3, 100, 0, 0, 0};
float fTargetVel[6] = {100, 100, 100, 100, 100, 100};
float fTargetAcc[6] = {100, 100, 100, 100, 100, 100};
float fTargetTime = 6;
Drfl.servol_rt(fTargetPos, fTargetVel, fTargetAcc, fTargetTime);
```

5.6.8 CDRFLEx.speedj_rt

Features

This is a function that controls the joint velocity from an external controller.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetVel	float[6]	-	Target Joint Velocity. [deg/s]
fTargetAcc	float[6]	-	Target Joint Acceleration [deg/s2]. If None Values(-10000) entered, it is automatically calculated based on the target joint velocity you entered.
fTargetTime	float	-	Target Time [s]

Note

- Asnyc Command.
- The inner profile is interpolated to reach (fTargetVel, fTargetAcc) at fTargetTime .
- If fTargetTime <= controller's control period (=1ms), control is performed at the corresponding speed without interpolation.
- If the next command is not received until arriving at fTargetVel, the last input velocity is maintained.
- However, for safety, if the next command is not received for 0.1[s], an error is generated as a time-out and stops.
- If the acceleration limit set globally during motion is exceeded, the motion is not stopped and an Info message is generated.

Caution

- In the current version, it is not linked with Operation Speed [%].

Return

Value	Description
0	Error
1	Success

Example

```
// main.cpp
CDRFLEx drfl;
```

```
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
Drfl.set_on_rt_monitoring_data(OnRTMonitoringData);
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();

float time = 0.0;
int count = 0;
float q[NUMBER_OF_JOINT] = {0.0, };
float q_dot[NUMBER_OF_JOINT] = {0.0, };
float q_d[NUMBER_OF_JOINT] = {0.0, };
float q_dot_d[NUMBER_OF_JOINT] = {0.0, };
float vel_d[NUMBER_OF_JOINT] = {0.0, };
float acc_d[NUMBER_OF_JOINT] = {0.0, };
float kv[NUMBER_OF_JOINT] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0}; // have to tune
float kp[NUMBER_OF_JOINT] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0}; // have to tune
float ki[NUMBER_OF_JOINT] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0}; // have to tune
float integral_v_error[NUMBER_OF_JOINT] = {0.0, };
while (1)
{
    time=(++count)*period;
    // get current state
    memcpy(q, drfl.read_data_rt()->actual_joint_position, sizeof(float)*6);
    memcpy(q_dot, drfl.read_data_rt()->actual_joint_velocity, sizeof(float)*6);

    // make trajectory
    TrajectoryGenerator(q_d, q_dot_d); // Custom Trajectory Generation Function

    // velocity feedforward + pi controller
    for (int i=0; i<6; i++) {
        q_dot_v[i] = kv[i] * (q_d[i] - q[i]);
        integral_v_error[i] += q_dot_v[i] - q_dot[i];
        vel_d[i] = q_dot_d[i] + kp[I;]*(q_dot_v[i] - q_dot[i]) + ki[i]*integral_v_error[i];
        acc_d[i] = -10000;
    }

    drfl.speedj_rt(vel_d, acc_d[i], period);

    if(time > plan1.time)
    {
        time=0;
        Drfl.stop(STOP_TYPE_SLOW);
        break;
    }

```

```
    rt_task_wait_period(NULL); // RTOS function
}
```

5.6.9 CDRFLEx.speedl_rt

Features

This is a function that controls the task velocity from an external controller.

Parameter

Parameter Name	Data Type	Default Value	Description
fTargetVel	float[6]	-	Target Task Velocity. [mm/s, deg/s]
fTargetAcc	float[6]	-	Target Task Acceleration. [mm/s, deg/s] If None Values(-10000) entered, it is automatically calculated based on the target task velocity you entered.
fTargetTime	float	-	Target Time [s]

Note

- Asnyc Command.
- The inner profile is interpolated to reach (fTargetVel, fTargetAcc) at fTargetTime .
- If fTargetTime <= controller's control period (=1ms), control is performed at the corresponding speed without interpolation.
- If the next command is not received until arriving at fTargetVel, the last input velocity is maintained.
- However, for safety, if the next command is not received for 0.1[s], an error is generated as a time-out and stops.
- If the acceleration limit set globally during motion is exceeded, the motion is not stopped and an Info message is generated.

Caution

- In the current version, it is not linked with Operation Speed [%].
- In the current version, it does not work with the force/compliance control function.
- In the current version, it does not work with DR_VAR_VEL among the singularity options. If set, it is automatically set to the DR_AVOID option.

Return

Value	Description
0	Error
1	Success

Example

```
float fTargetPos[6] = {1500, 3, 100, 0, 0, 0};
float fTargetVel[6] = {100, 100, 100, 100, 100, 100};
float fTargetAcc[6] = {100, 100, 100, 100, 100, 100};
float fTargetTime = 6;
Drfl.servol_rt(fTargetPos, fTargetVel, fTargetAcc, fTargetTime);
```

5.6.10 CDRFLEx.torque_rt

Features

This is a function that controls the motor torque from an external controller.

Parameter

Parameter Name	Data Type	Default Value	Description
fMotorTor	float[6]	-	Target Motor Torque [Nm]
fTargetTime	float	-	Target Time [s]

Note

- Asnyc Command
- Interpolate profile to be fMotorTor at fTargetTime.
- If fTargetTime <= controller's control period (=1ms), control is performed with the corresponding motor torque without interpolation.
- If the next command is not received until arriving at fMotorTor, the last input torque is maintained.
- However, for safety, if the next command is not received for 0.1[s], an error is generated as a time-out and stops.

> **Caution**
>
> In the case of the H series, there is a gravity compensator on the second axis, so the input motor torque and the torque generated by the gravitational compensator are added to move the robot. The gravity_torque of the external controller output data means (=link gravity torque – the torque of the gravity compensator)

Return

Value	Description
0	Error
1	Success

Example

```
// main.cpp
CDRFLEx drfl;
drfl.connect_rt_control();
string version = “v1.0”;
float period = 0.001; // 1 msec
int losscount = 4;
Drfl.set_on_rt_monitoring_data(OnRTMonitoringData);
drfl.set_rt_control_output(version, period, losscount);
drfl.start_rt_control();

float time = 0.0;
int count = 0;
float q[NUMBER_OF_JOINT] = {0.0, };
float q_dot[NUMBER_OF_JOINT] = {0.0, };
float q_d[NUMBER_OF_JOINT] = {0.0, };
float q_dot_d[NUMBER_OF_JOINT] = {0.0, };
float trq_g[NUMBER_OF_JOINT] = {0.0, };
float trq_d[NUMBER_OF_JOINT] = {0.0, };
float kp[NUMBER_OF_JOINT] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0}; // have to tune
float kd[NUMBER_OF_JOINT] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0}; // have to tune
while (1)
{
    time=(++count)*period;
    // get current state
    memcpy(q, drfl.read_data_rt()->actual_joint_position, sizeof(float)*6);
    memcpy(q_dot, drfl.read_data_rt()->actual_joint_velocity, sizeof(float)*6);
    memcpy(trq_g, drfl.read_data_rt()->gravity_torque, sizeof(float)*6);
```

```
    // make trajectory
    TrajectoryGenerator(q_d, q_dot_d); // Custom Trajectory Generation Function

    // gravity compensation + pd control
    for (int i=0; i<6; i++) {
        trq_d[i] = trq_g[i] + kp[i]*(q_d[i] - q[i]) + kd[i]*(q_dot_d[i] - q_dot[i]);
    }
    drfl.torque_rt(trq_d, 0);

    if(time > plan1.time)
    {
        time=0;
        Drfl.stop(STOP_TYPE_SLOW);
        break;
    }

    rt_task_wait_period(NULL); // RTOS function
}
```