API version : GL010106-beta

Doosan Robot

M0609 I M0617 I M1013 I M1509
A0509 I A0509s I A0912 I A0912s

API Manual

3. Function 62

History of Document Creation/Revision

Revision No.	Creation/Revision Pages and Contents	Revision Date	Amender
1.0	Initial Creation and Distribution	2018-06-29	Lee Jeong-woo
1.1	Update new commands	2020-05-20	Gong Jin-Hyuk
1.11	Update function name(DRL style)	2020-05-28	Gong Jin-Hyuk
1.12	Split the header(add DRCFEx class)	2020-06-08	Gong Jin-Hyuk

1. 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.

1.1 Installation Guide

1.1.1 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

1.1.2 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.
A) Setting of header file (include)
[Composition Attribute]->[C/C++]->[General]->[Additional Include Directories]
B) Setting of library (lib) path
[Composition Attribute]->[Linker]->[General]->[Additional Library Directories]

1.1.3 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.

1.1.4 Recommended Operational Specification

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

Classification	Name	Recommended Specification	Remarks
Windows	Operating System	Windows 7	
	CPU Acrchietecture	X86(64bi & 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)	
	Compliler	GNU GCC 4.x or higher	

1.2 Programming Instructions

1.2.1 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.

1.2.2 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.

1.2.3 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 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.

1.2.4 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.

1.2.5 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 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	Robot Operation State	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.

1.2.6 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.

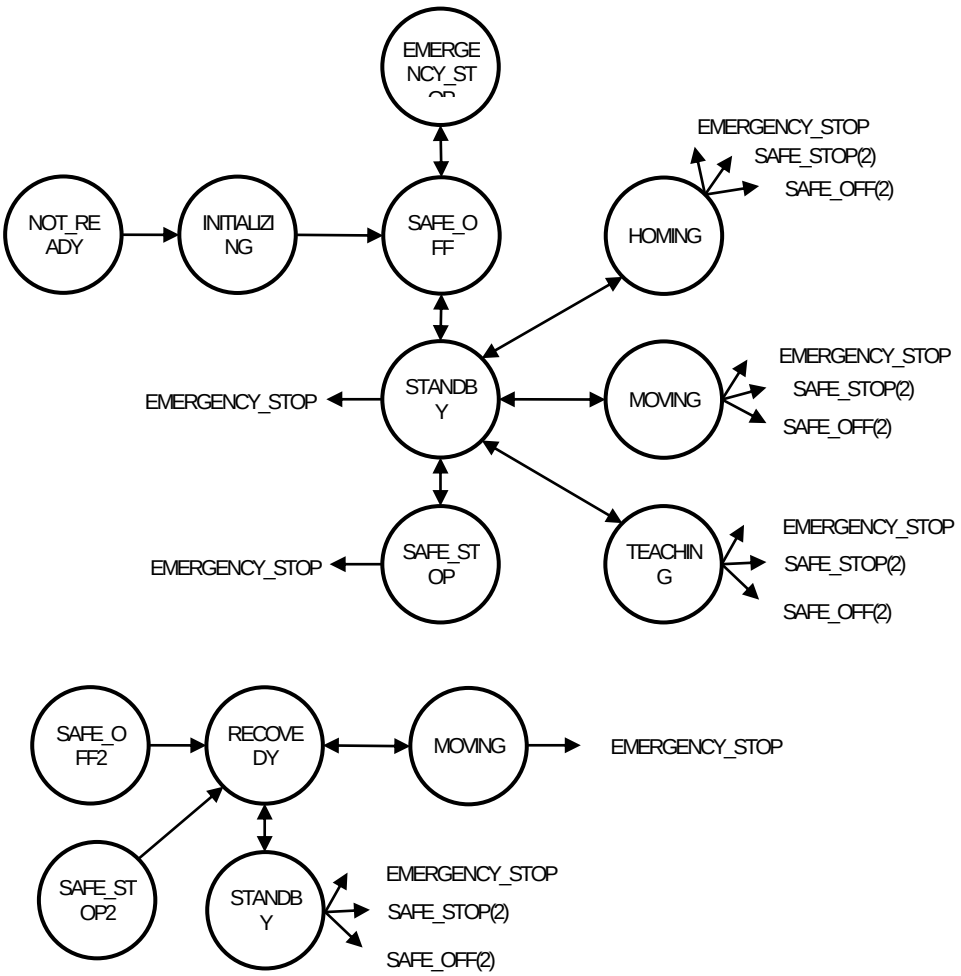
The EMERGENCY_STOP state is converted by the E/M button in whatever states excluding the initialization state (STATE_INITIALIZING) and robot stops. When an internal function or motion error of the robot controller occurs, it is converted into the SAFE_OFF state (motor and brake power cut-off) or SAFE_STOP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

Rank	Robot state control command	Description
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.



1.2.7 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.

1.2.8 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).

1.2.9 Limitation on Force Control Function

A force control function is provided using DRL, a robot programming language from our company (refer to the DRL programming guide document) and is not provided directly as an API function. As this API can be delayed due to the characteristics of TCP/IP communication, the force control logic may not be applied as intended by the user. Therefore, this is not provided due to the safety issue to protect personal and material property.

If the force control function is needed, DRL programming should be made and executed with the automatic mode using the programming control function that this API provides.

2. Definition

2.1 Constant and Enumeration Type

2.1.1 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
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

Rank	Constant Name	Description
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.

2.1.2 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.
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.

2.1.3 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

2.1.4 enum.SPEED_MODE

This is defined the same as the MONITORING_SPEED enumeration constant.

2.1.5 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

2.1.6 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)

2.1.7 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

2.1.8 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)

2.1.9 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_REQUEEST	Transmission of message for forced collection of control right
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

2.1.10 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

2.1.11 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

2.1.12 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

2.1.13 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

2.1.14 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
11	FORCE_AXIS_B	y axis rotation

Rank	Constant Name	Description
12	FORCE_AXIS_C	z axis rotation

2.1.15 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

2.1.16 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

2.1.17 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
1	FORCE_MODE_RELATIVE	Relative Coordinate

2.1.18 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

2.1.19 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

2.1.20 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
1	MOVEB_BLENDING_TYPE_CIRLCE	Circle

2.1.21 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

Rank	Constant Name	Description
1	SPLINE_VELOCITY_OPTION_CONST	Constant velocity motion

2.1.22 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.

Rank	Constant Name	Description
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
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

2.1.23 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

2.1.24 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

2.1.25 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
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

2.1.26 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

Rank	Constant Name	Description
2	MODBUS_REGISTER_TYPE_INPUT_REGISTER	Input Register
3	MODBUS_REGISTER_TYPE_HOLDING_REGISTER	Holding Register

2.1.27 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

2.1.28 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

2.1.29 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

2.1.30 enum.CONTRL_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

2.1.31 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
3	DATA _TYPE_STRING	string
4	DATA _TYPE_POSJ	posj
5	DATA _TYPE_POSX	posx
6	DATA _TYPE_UNKNOWN	unknown

2.1.32 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

2.1.33 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

2.1.34 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
2	SINGULARITY_AVOIDANCE_VEL	Variable Speed

2.1.35 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

2.1.36 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

2.1.37 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)

2.2 Definition of Structure

2.2.1 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)

2.2.2 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

BYTE#	Field Name	Data Type	Value	Remarks
				(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

BYTE#	Field Name	Data Type	Value	Remarks
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
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.

2.2.3 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)은 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

BYTE#	Field Name	Data Type	Value	Remarks
				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
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
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

BYTE#	Field Name	Data Type	Value	Remarks
				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
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

BYTE#	Field Name	Data Type	Value	Remarks
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.

2.2.4 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
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 (#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

2.2.5 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	4개의 Digital On/Off 정보

2.2.6 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

2.2.7 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.

BYTE#	Field Name	Data Type	Value	Remarks
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.

2.2.8 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.

2.2.9 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

2.2.10 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

2.2.11 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

BYTE#	Field Name	Data Type	Value	Remarks
				Stop : 0

2.2.12 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 바이트 문자열
256	Data Type	uchar	-	int : 0 float : 1 string : 2 bool : 3

2.3 Definition of Log and Alarm

2.3.1 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

2.3.2 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)

2.3.3 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.

2.4 Definition of Callback Function

2.4.1 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);
}
```

2.4.2 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);
}
```

2.4.3 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);
}
```

2.4.4 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)
}
```

2.4.5 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)
}
```

2.4.6 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)
}
```

2.4.7 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)
}
```

2.4.8 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);
}
```

2.4.9 TOnHomingCompletedCB

- **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);
}
```

2.4.10 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);
}
```

2.4.11 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);
}
```

2.4.12 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.Homme(True)
}

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

2.4.13 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);
}
```

2.4.14 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);
}
```

2.4.15 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 th at requires an excessive execution time (within 50 msec) within the callback function shoul d 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);
}
```

2.4.16 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);
}
```

2.4.17 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);
}
```

2.4.18 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. Function

3.1 Robot Connection Function

3.1.1 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;
}
```

3.1.2 CDRFLEx.close_connection

- **Features**

  This is a function for disconnecting the robot controller.

- **Parameter**

  None

- **Return**

  None.

- **Example**

```
drfl.close_connection();
```

3.2 Robot Property Function

3.2.1 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
1	Success

- **Example**

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

3.2.2 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;
```

3.2.3 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)
    }
}
```

3.2.4 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);
}
```

3.2.5 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);
    }
}
```

3.2.6 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);
}
```

3.2.7 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 …
}
```

3.2.8 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 …
}
```

3.2.9 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);
}
```

3.2.10 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);
}
```

3.2.11 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()
}
```

3.2.12 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
0	Success
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;
		//...
```

3.2.13 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;
```

3.2.14 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);
```

3.2.15 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;
}
```

3.2.16 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);
```

3.2.17 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;
}
```

3.2.18 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;
```

3.2.19 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;
```

3.2.20 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;
```

3.2.21 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;
}
```

3.3 Functions That Register the Callback Functions

3.3.1 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);
 }
```

3.3.2 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);
}
```

3.3.3 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);
}
```

3.3.4 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)
}
```

3.3.5 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)
}
```

3.3.6 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)
}
```

3.3.7 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)
}
```

3.3.8 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)
}
```

3.3.9 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)
}
```

3.3.10 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)
}
```

3.3.11 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)
}
```

3.3.12 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);
}
```

3.3.13 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	TOnHommingCompletedCB	-	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);
}
```

3.3.14 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	TOnTpInitializingCompletedCB	-	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);
}
```

3.3.15 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);
}
```

3.3.16 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);
}
```

3.3.17 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);
}
```

3.3.18 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);
}
```

3.4 Functions That Manage Control Right

3.4.1 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);

}
```

3.5 Basic Control Functions

3.5.1 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);
```

3.5.2 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);
```

3.6 Functions That Control Motion

3.6.1 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

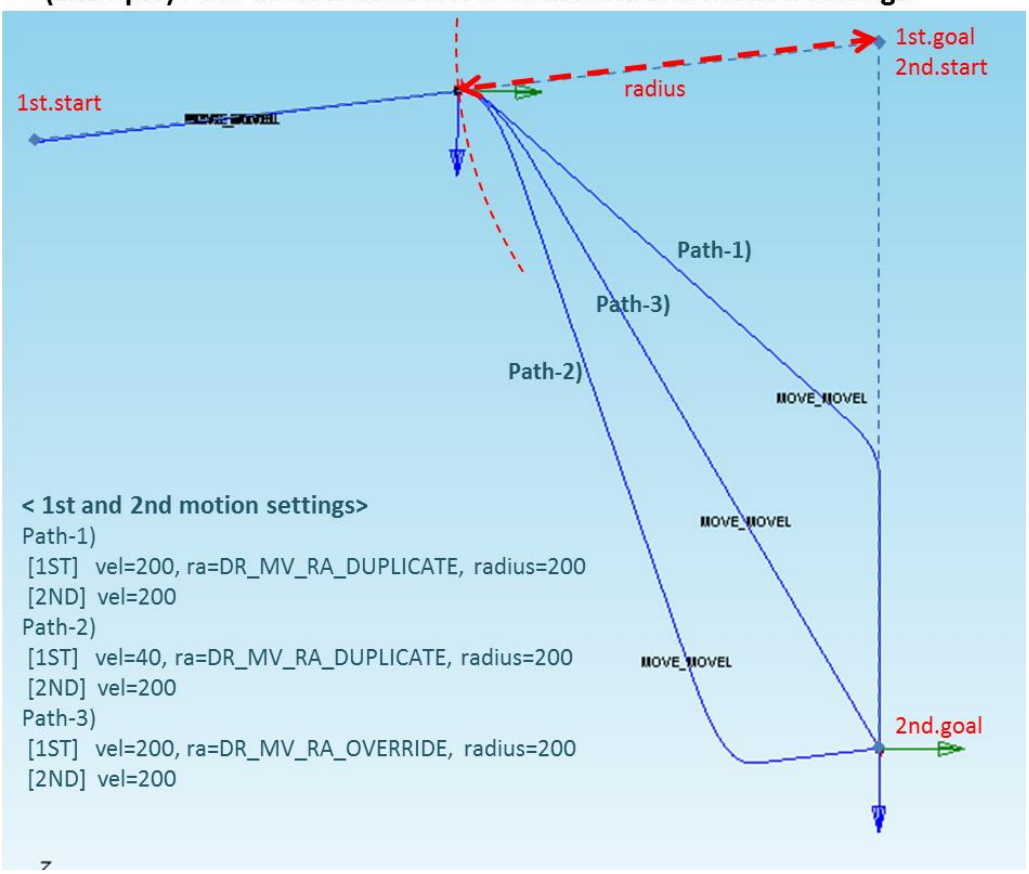
Note

- 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.

< (Example) Path differences accord. to 1st and 2nd motion settings>



- **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.
```

3.6.2 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
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

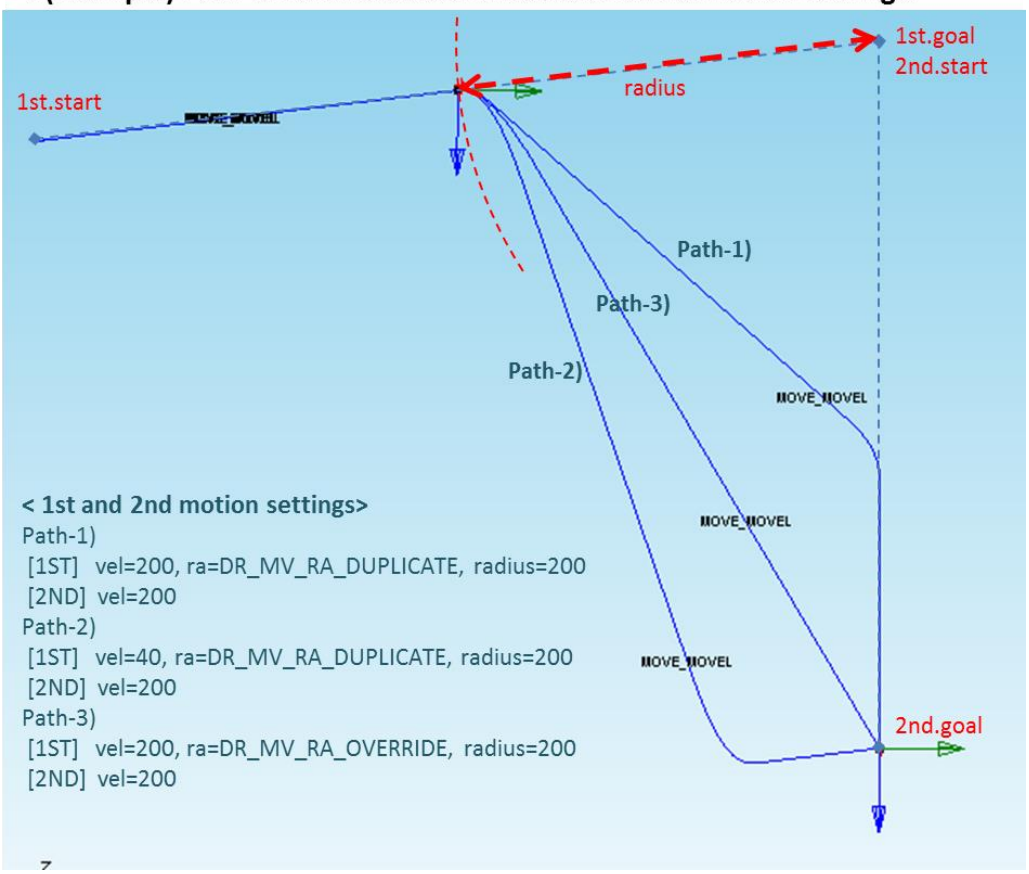
- If an argument is inputted to fTargetVel (e.g., fTargetVel ={30, 0}), the input argument corresponds to the linear velocity of the motion, while the angular velocity is determined proportionally to the linear velocity.
- If an argument is inputted to fTargetAcc (e.g., fTargetAcc ={60, 0}), the input 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.

< (Example) Path differences accord. to 1st and 2nd motion settings>



- **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);
```

3.6.3 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).

- **Parameter**

Parameter Name	Data Type	Default Value	Description
fTargetPos	float[6]	-	Target TCP Position for six axes
iSolutionSpace	unsigned char	-	joint combination shape (Refer to the below description)
fTargetVel	float	-	Velocity
fTargetAcc	floa	-	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
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

- **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.
```

3.6.4 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
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

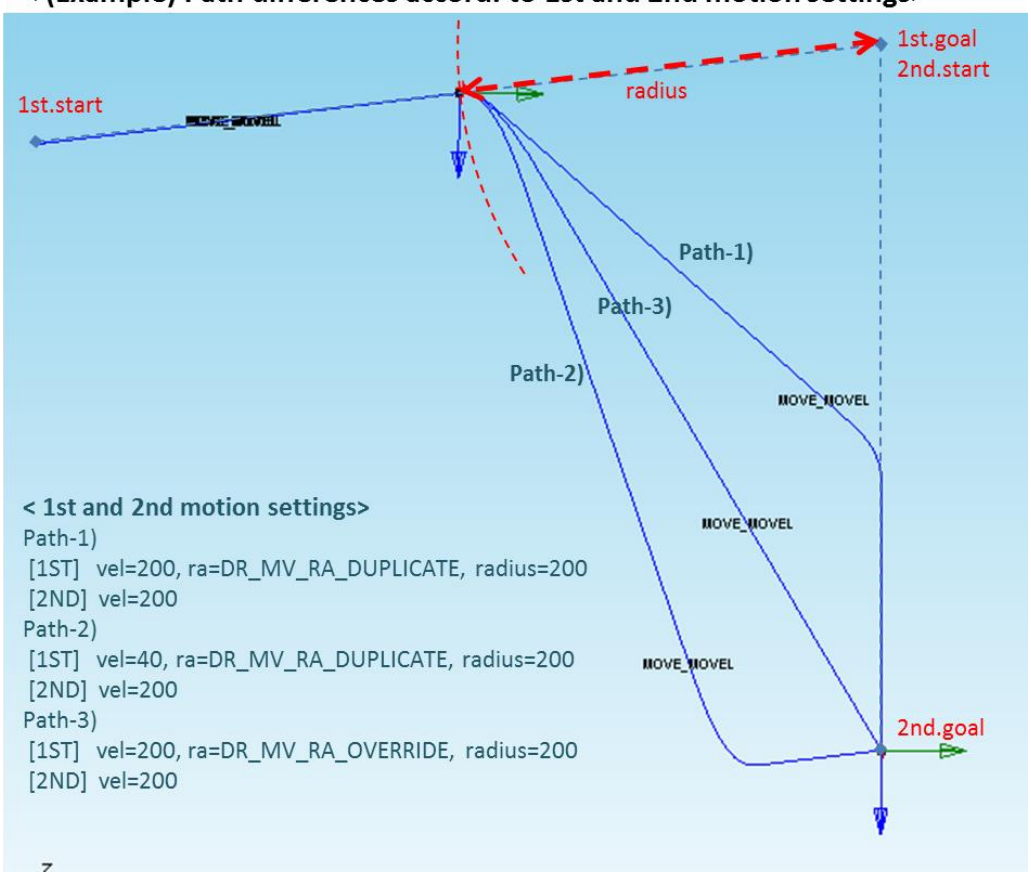
coordinate from the starting point, while fTargetPos[1] is the relative coordinate from fTargetPos[0].)

- 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.

< (Example) Path differences accord. to 1st and 2nd motion settings>



- **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);
```

3.6.5 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	-	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

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)
```

3.6.6 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]
eMoveMode	enum.MOVE_M ODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMove Reference	enum.MOVE_R EFERENCE	MOVE_REFERENCE _BASE	Refer to the Definition of Constant and Enumeration Type
eVelOpt	enum.SPLINE_ VELOCITY_OP TION	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)
```

3.6.7 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_REFERENCE	MOVE_REFERENCE_BASE	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.
```

3.6.8 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]
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

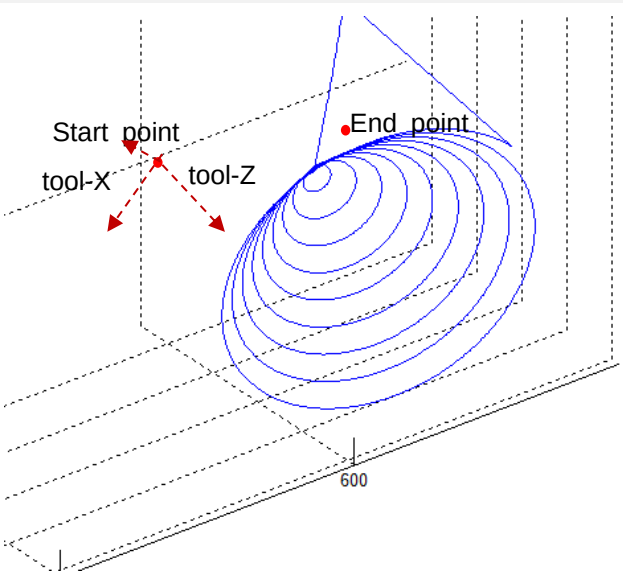
- If the rotating acceleration calculated by the spiral path is too great. an error can be generated to ensure safe motion. In this case, reduce fTargetVel, fTargetAcc or fTargetTime value.

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.
```



3.6.9 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**

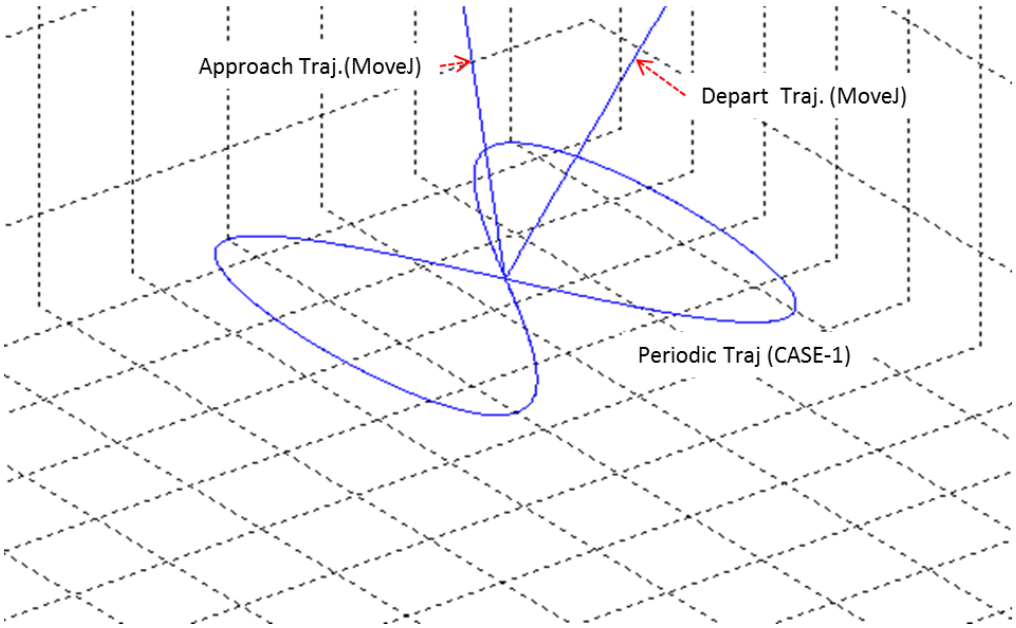
Parameter Name	Data Type	Default Value	Range	Description
fAmplitude	float[6]	-	>=0	Amplitude (motion between -amp and +amp) [mm] or [deg]
fPeriodic	float[6]	-	>=0	period(time for one cycle)[sec]
fAccelTime	float	-	>=0	Acc-, dec- time [sec]
nRepeat	unsigned char	-	> 0	Repetition count
eMoveReference	enum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

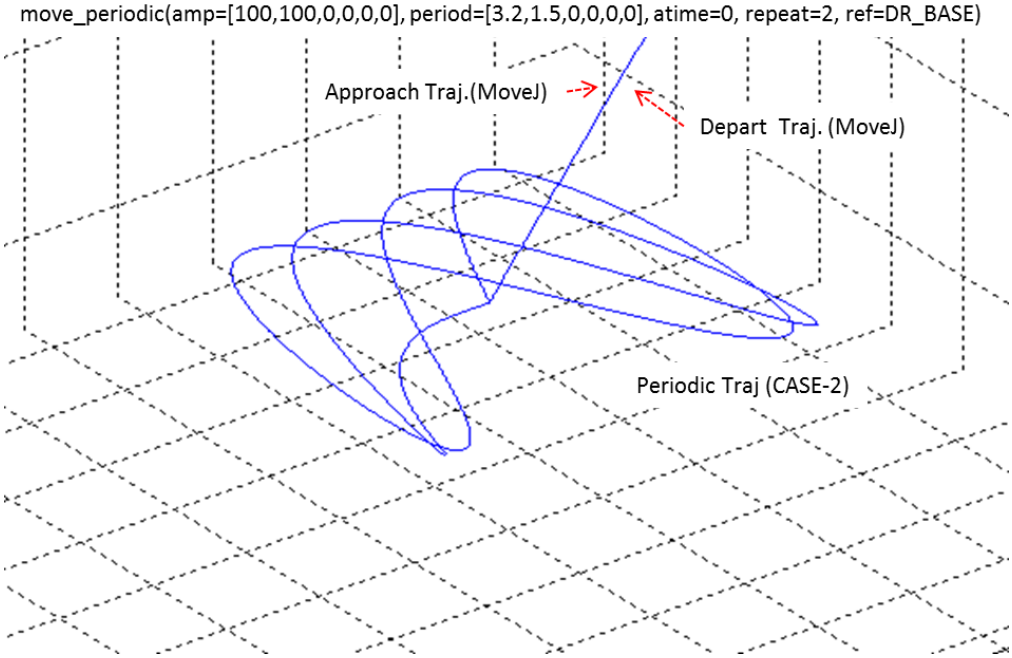
CASE-1) All-axis motions end at the same time

move_periodic(amp=[100,100,0,0,0,0], period=[3.2,1.6,0,0,0,0], atime=3.1, repeat=2, ref=DR_BASE)



CASE-2) Diff-axis motions end individually

move_periodic(amp=[100,100,0,0,0,0], period=[3.2,1.5,0,0,0,0], atime=0, repeat=2, ref=DR_BASE)



- 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.
  Max. 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
```

3.6.10 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	-	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
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.
- 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);
```

3.6.11 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
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);
```

3.6.12 CDRFLEx.amovejx

- **Features**

As an asynchronous movejx, it operates the same as the movejx 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
iSolutionSpace	unsigned char	-	joint combination shape (Refer to the description below)
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
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

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

Caution

Blending into the current active motion is disabled in the case of input with relative motion (eMoveMode = MOVE_MODE_RELATIVE), and it is recommended to blend using amovej() or moveLAsync().

- **Return**

Value	Description
0	Error
1	Success

- **Example**

```
// D-Out 2 seconds after the joint motion starts with x1
float x1[6] = { 559, 34.5, 651.5, 0, 180, 0 };
float sol=2;
float jvel=10;
float jacc=20;
drfl.amovejx(x1, sol, jvel, jacc);
Sleep(2000);
drfl.set_digital_output(GPIO_CTRLBOX_DIGITAL_INDEX_1, 1);
```

3.6.13 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);
```

3.6.14 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	-	Velocity
fTargetAcc	float	-	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 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.

- **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);
```

3.6.15 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_M ODE	MOVE_MODE_ ABSOLUTE	Refer to the Definition of Constant and Enumeration Type
eMove Reference	enum.MOVE_R EFERENCE	MOVE_REFERENCE _BASE	Refer to the Definition of Constant and Enumeration Type
eVelOpt	enum.SPLINE_ VELOCITY_OP TION	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 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);
```

3.6.16 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 fTargetAcc.
- 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);
```

3.6.17 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
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

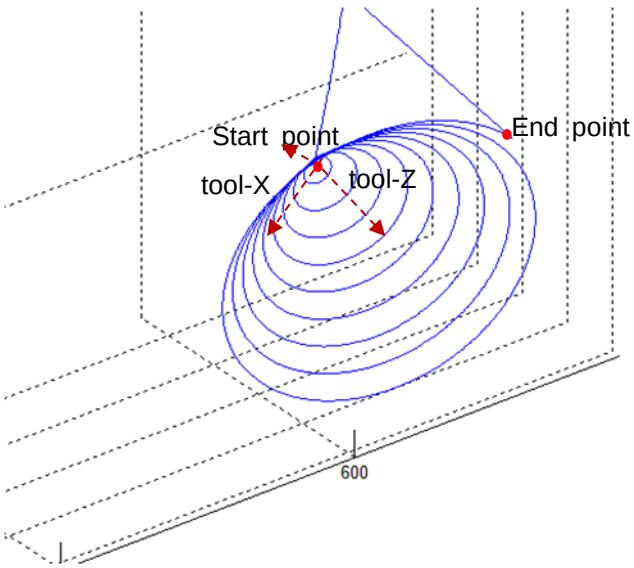
- If the rotating angular acceleration calculated by the spiral path is too great, an error can be generated to ensure safe motion.
  In this case, reduce the fTargetVel, fTargetAcc or fTargetTime value.

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);
```


Start point
End point
tool-X
tool-Z
600

3.6.18 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**

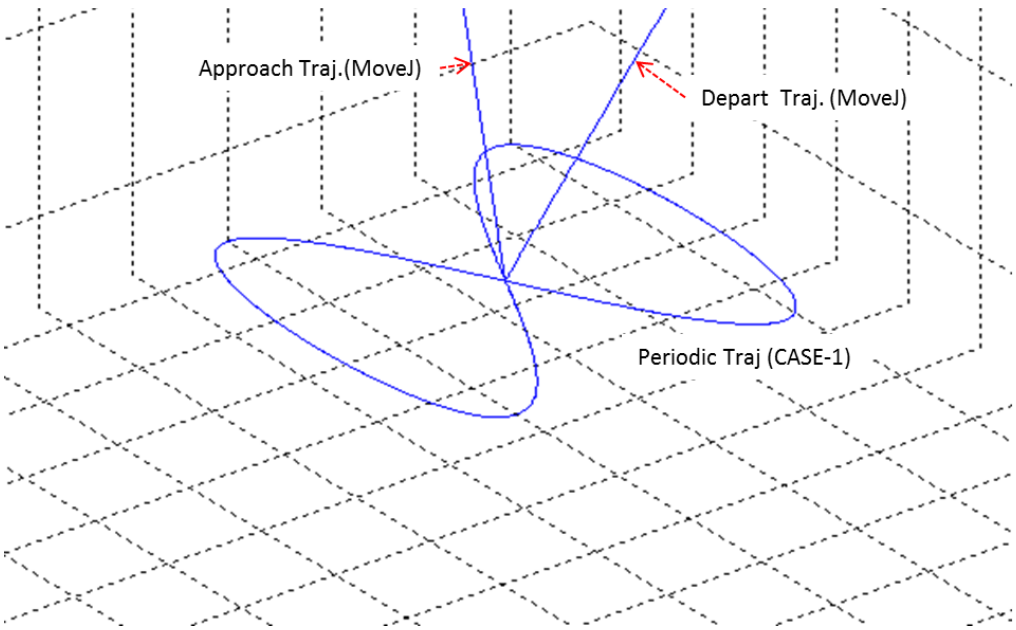
Parameter Name	Data Type	Default Value	Range	Description
fAmplitude	float[6]	-	>=0	Amplitude(motion between -fAmplitude and +fAmplitude) [mm] or [deg]
fPeriodic	float[6]	-	>=0	period(time for one cycle)[sec]
fAccelTime	float	-	>=0	Acc-, dec- time [sec]
nRepeat	unsigned char	-	> 0	Repetition count
eMoveReference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

CASE-1) All-axis motions end at the same time
move_periodic(amp=[100,100,0,0,0,0], period=[3.2,1.6,0,0,0,0], atime=3.1, repeat=2, ref=DR_BASE)



- 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.
  Max. 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();
```

3.6.19 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
```

3.6.20 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.
```

3.6.21 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.
```

3.6.22 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();
```

3.6.23 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;
}
```

3.6.24 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);
```

3.6.25 CDRFLEx.ikin

- **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};
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);
```

3.6.26 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);
```

3.6.27 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;
    }
}
```

3.6.28 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, li
mit_dPOS, limit_dPOS_per);
```

3.6.29 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);
}
```

3.6.30 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();.
}
```

3.7 Robot Setting Function

3.7.1 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
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");
```

3.7.2 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");
```

3.7.3 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

- **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");
```

3.7.4 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);
}
```

3.7.5 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);
```

3.7.6 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);
```

3.7.7 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);
```

3.7.8 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);
}
```

3.7.9 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”)
```

3.7.10 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();
```

3.7.11 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()
```

3.7.12 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 mo
de 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
```

3.7.13 CDRFLEx.set_up_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
```

3.7.14 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
```

3.8 I/O Control Function

3.8.1 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);
```

3.8.2 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
}
```

3.8.3 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
}
```

3.8.4 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 • ON: 1 • 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);
```

3.8.5 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
}
```

3.8.6 CDRFLEx.get_digital_output

- **Features**

This is a function for checking a signal at the digital contact point mounted on the control bo x 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
}
```

3.8.7 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);
```

3.8.8 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);
```

3.8.9 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 • In the case of current mode: 4.0~20.0 [mA] • 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);
```

3.8.10 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_CTRL BOX_ANALOG_IN DEX_	-	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);
```

3.8.11 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);
```

3.8.12 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
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
```

3.8.13 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);
```

3.8.14 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…
}
```

3.9 Program Control Function

3.9.1 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.

3.9.2 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 eStopType received as an argument and stops 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**

```
DRL_PROGRAM_STATE eProgramState = drfl. get_program_state ();
if ((eProgramState == DRL_PROGRAM_STATE_PLAY) ||
      (eProgramState == DRL_PROGRAM_STATE_HOLD)) {

      drfl.drl_stop(STOP_TYPE_QUICK);
      //…
}
```

Note

- When the program is terminated either normally or by error, the program stop command must be executed.

3.9.3 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();
}
```

3.9.4 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();
    //...
}
```

3.9.5 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(1~100)

- **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(10,10.1
0,10,10.10), 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);
      }
}
```

3.9.6 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());
```

3.9.7 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);
```

3.9.8 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");
```

3.10 Force/Stiffness Control and Other User-Friendly Features

3.10.1 CDRFLEx.parallel_axis

- **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	TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eSourceRef	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);
```

3.10.2 CDRFLEx.parallel_axis

- **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	TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	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);
```

3.10.3 CDRFLEx.align_axis

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	TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	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);
```

3.10.4 CDRFLEx.align_axis

- **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	TASK_AXIS	-	Refer to the Definition of Constant and Enumeration Type
eTargetRef	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);
```

3.10.5 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	FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
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;
}
```

3.10.6 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	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);
```

3.10.7 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();
```

3.10.8 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 rot ational 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	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();
```

3.10.9 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	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, p
os2, pos3, pos4);
for (int i=0; i<NUM_TASK; i++)
{
    cout << pose_user1->_fPosition[i] << endl;
}
```

3.10.10 CDRFLEx.set_user_cart_coord

- **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	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);
```

3.10.11 CDRFLEx.set_user_cart_coord

- **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	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);
```

3.10.12 CDRFLEx.set_user_cart_coord

- **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	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);
```

3.10.13 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	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;
```

3.10.14 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);
```

3.10.15 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	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	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);
```

3.10.16 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();
```

3.10.17 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	FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
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**

```
bool CON1 = Drfl.check_position_condition_abs(FORCE_AXIS_X, -5, 0, COORDINATE_SYS
TEM_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_SY
STEM_BASE);
bool CON6 = Drfl.check_position_condition_abs(FORCE_AXIS_Z, -10, -5);
```

3.10.18 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) shoul d 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	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	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, COORDINA
TE_SYSTEM_TOOL);
```

3.10.19 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) shoul d 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	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
eMode	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_rel(FORCE_AXIS_Z, -10, -5, posx1, MOVE_MOD
E_RELATIVE, COORDINATE_SYSTEM_TOOL);
```

3.10.20 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	FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float	-	Minimum value
fTargetMax	float	-	Maximum value
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**

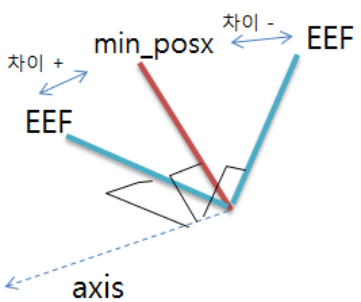
```
bool fcon1 = Drfl.check_force_condition(FORCE_AXIS_Z, 5, 10, COORDINATE_SYSTEM_W
ORLD);
bool fcon2;
bool pcon1;
while(true){
    fcon2 = Drfl.check_force_condition(FORCE_AXIS_C, 30, -10000);
    pcon1 = Drfl.check_position_condition_abs(FORCE_AXIS_X, 0, 0.1);
    if(fcon2 && pcon1)
    {
        break;
    }
}
```

3.10.21 CDRFLEx.check_orientation_condition

- **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	FORCE_AXIS	-	Refer to the Definition of Constant and Enumeration Type
fTargetMin	float[6]	-	Minimum value
fTargetMax	float[6]	-	Maximum value
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**

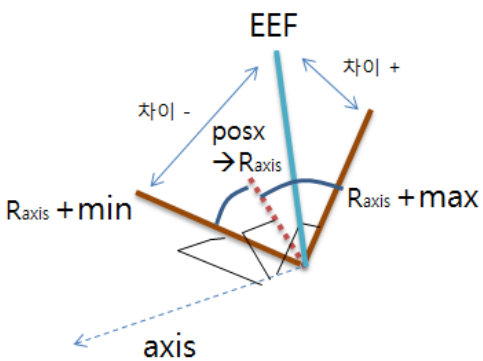
```
float posx1[6] = {400, 500, 800, 0, 180, 30};
float posx2[6] = {400, 500, 500, 0, 180, 60};
bool con1 = Drfl.check_orientation_condition(FORCE_AXIS_C, posx1, posx2);
```

3.10.22 CDRFLEx.check_orientation_condition

- **Features**

This function checks the difference between the current pose and the rotating angle range of th e 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” tech nique. 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 fro m 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	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
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, 30};
bool con1 = Drfl.check_orientation_condition(FORCE_AXIS_C, 0, 5, posx1);
```

3.10.23 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

Parameter Name	Data Type	Default Value	Description
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_S
YSTEM_TOOL);
```

Doosan Robotics

www.doosanrobotics.com