DENSO
b-CAP Communication Specifications
for RC8

Version 1.0.5

July 16, 2021

[Remarks]

[Revision history]

Date	Revision	Content
2013-2-12	1.0.0	First edition
2013-8-20	1.0.1	Added the reference of sample libraries for ANSI-C and Java. Added sample programs (variable access, task control, and robot control) Added the explanation of the Slave Mode instruction speed/accel limit. Added b-CAP Tester Raw packet mode. Added the correspondence table about b-CAP function ID and CAO interface.
2013-10-08		Added SlvSendFormat and SlvRecvFormat to SlvMode.
2014-10-13	1.0.2	Added descriptions of communication speed limit in Slave Mode. Added descriptions of command restriction in Slave Mode.
2015-04-02		Changed sample programs. Added electric current value obtainment in Slave Mode
2017-02-07	1.0.3	Added descriptions of extended-joint in Slave Mode.
2017-10-19	1.0.4	Added descriptions of the Service_Start option character string.
2021-07-16	1.0.5	Changed the sample program of b-CAP Slave Mode. Added about unsuported functions with b-CAP Slave Mode.

[Devices]

Device	Version	Notes
RC8	V1.4.0 or later	

Contens

1. Introduction

This document describes specifications of b-CAP communication protocol for RC8.

b-CAP is a communication protocol that has been developed based on the concept of CAP to improve communication speed; therefore, b-CAP has the same features as CAP series, as follows.

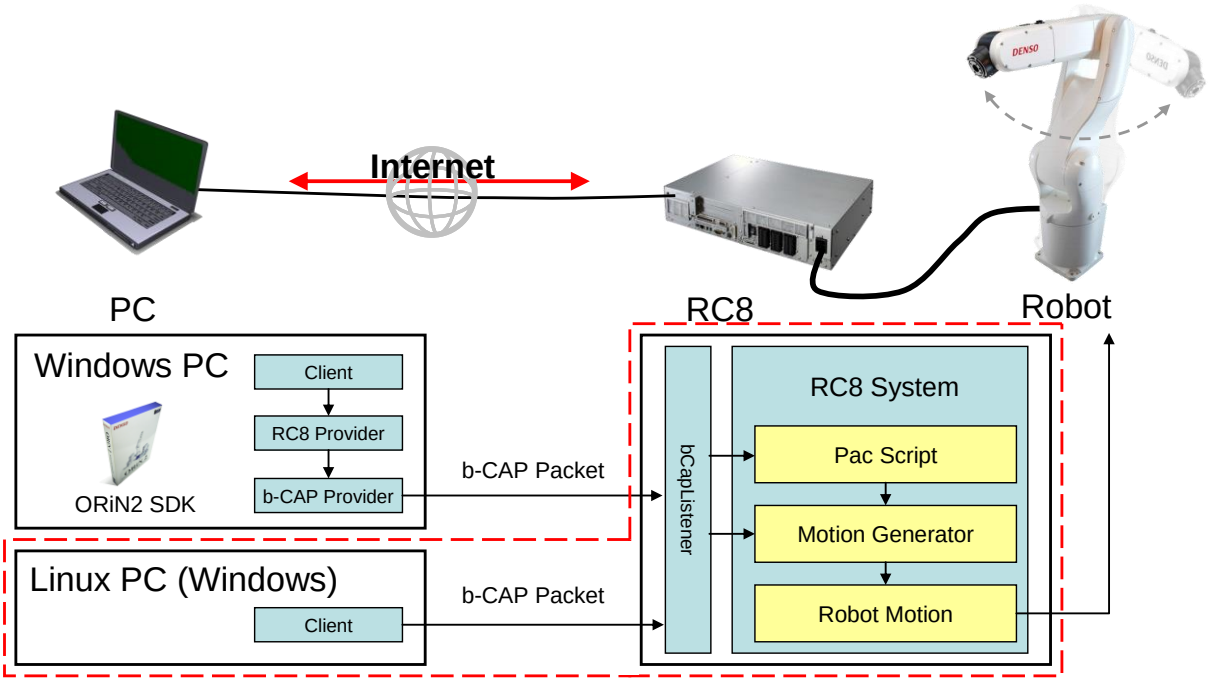
- Having the same service structure as the object model of CAO provider.
- To invoke function, specify an object by the object ID.
- Event occurrence on the server side is detected by polling.

For more detailed information about CAP series, please refer to "CAP provider User's guide" (CAP_ProvGuide_en.pdf) included in ORiN2 SDK.

1.1. System Configuration

This document targets that the following operation environment.

- Client software runs on Linux, or, client software runs on Windows with or without ORiN2 SDK preinstalled.
- The version of RC8 is Version 1.4.0 and the higher.



Inside of RC8, bCapListner receives b-CAP packet, then assigns commands according to the contents of the packets. RC8 includes PacScript which is an interpreter of a robot language, MotionGenerator which generates trajectory when a robot motion command is issued, and RobotMotion which controls robot in real time.

A client that runs on ORiN2 SDK-installed Windows can operate RC8 by using RC8 Provider. RC8 Provider

converts commands to b-CAP packets through b-CAP Provider, then transmits it to RC8.

Clients that cannot use RC8 Provider, such as Linux or Windows without ORiN2 SDK preinstalled, can control RC8 by transmitting b-CAP packet individually.

This document describes how to operate RC8 by transmitting and receiving b-CAP packets with concrete examples.

1.2. Reference information

This document includes examples of b-CAP packet transmission to perform basic operation of RC8. If you require more detailed operations, refer to the following files.

Regarding to the basic structure of b-CAP, refer to the following files.

- b-CAP Specifications
ORiN2\CAP\b-CAP\Doc\b-CAP_Spec_en.pdf

A command supported by RC8 specification b-CAP complies with RC8 Provider. For arguments of commands, refer to the following files.

- RC8 Provider Guide
ORiN2\CAO\ProviderLib\DENSO\RC8\Doc\RC8_ProvGuide_en.pdf

You can use sample libraries to create b-CAP packets and then send the packets to the Controller.

- Sample library for ANSI-C
ORiN2\CAP\b-CAP\CapLib\DENSO\RC8\Include\C
- Sample library for Java
ORiN2\CAP\b-CAP\CapLib\DENSO\RC8\Include\Java

1.3. b-CAP Slave Mode

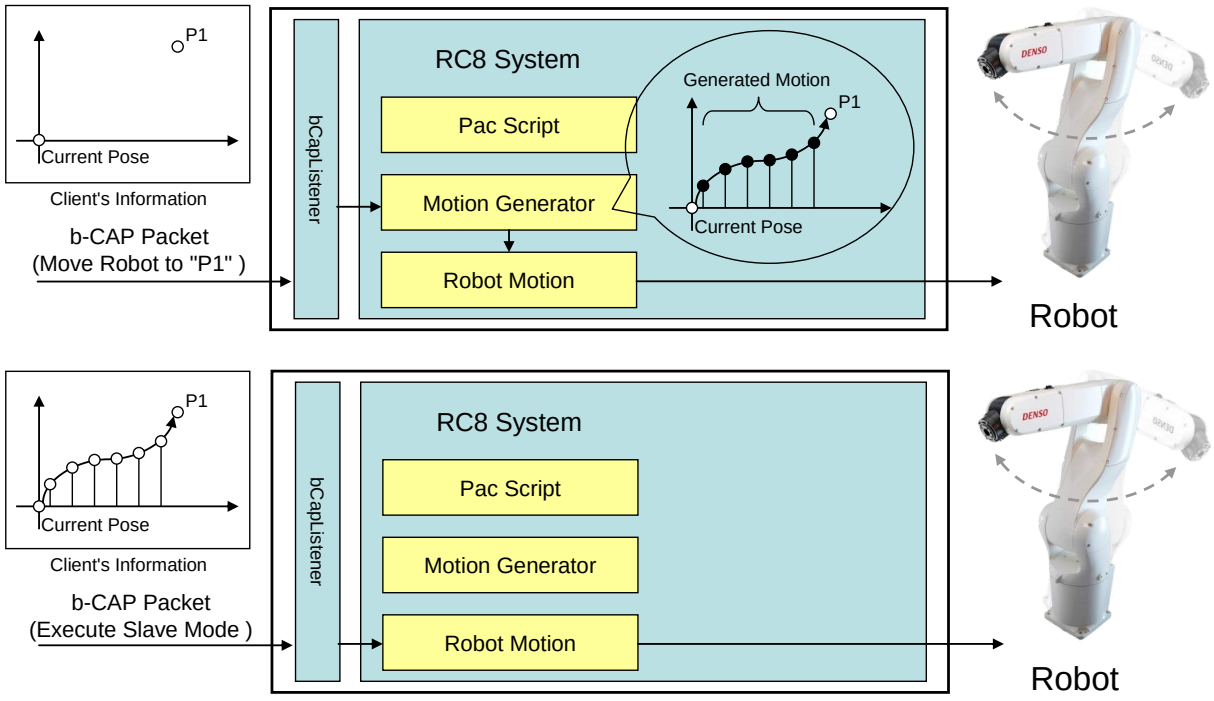
Slave Mode is function to control a robot periodically in very short intervals by sending data of robot position and posture.

In normal robot motion commands (e.g, "Move 1", "P1", etc.), RC8 controls a robot by generating trajectory in real time in order to achieve the target posture specified by client. On the other hand, in Slave Mode, a client repeatedly specifies a robot posture in short intervals in order to control robot motion in real time. Using this function, the client can control trajectory of the robot freely.

Slave Mode is an optional function of RC8 robot controller. Please add the license key according to your needs. You can confirm the license key on the member site. Please select [RC8 Free License Confirmation], and input the serial number printed in the chassis of RC8[1].

Note: Registration and login to a member site are required to confirm the license key.

http://www.denso-wave.com/en/robot/index.html



[1] A robot controller that the b-CAP Slave license is enabled restricts communication speed for other operations. This is because the controller places a priority on ensuring the communication for the b-CAP Slave operation. If time-out occurs before other operations are not completed, take any prevention measures, such as setting longer time-out period.

2. Setup of RC8

This section describes the necessary setup of RC8 in order to operate the controller by transmitting and receiving b-CAP packet. For details about setup, please refer to RC8 Provider Guide.

2.1. Setup of system parameters

Before operating a robot controller by b-CAP packet, you need to setup the robot controller which is to be controlled. With a teach pendant (TP) or a mini-pendant (MiniTP), configure the communication permission and executable token.

Communication permission is a setting that gives permission of the reading (or writing) data in (or from) the robot controller to the communication devices. When you write variable data in the robot controller, or control the robot, make sure to give the communication permission to the communication device.

Executable token is a setting that assigns a communication device the rights to activate (or perform) a program task to the robot controller, to turn the motor power ON, and to control the robot (motion instruction). You can select any value from 1) TP, 2) I/O, 3) Ethernet, and 4) Any. If "Any" is selected, robot controller can be operated regardless of the communication path, so make sure to perform exclusive control between communication devices so as not to cause inconsistency between client PC and PLC.

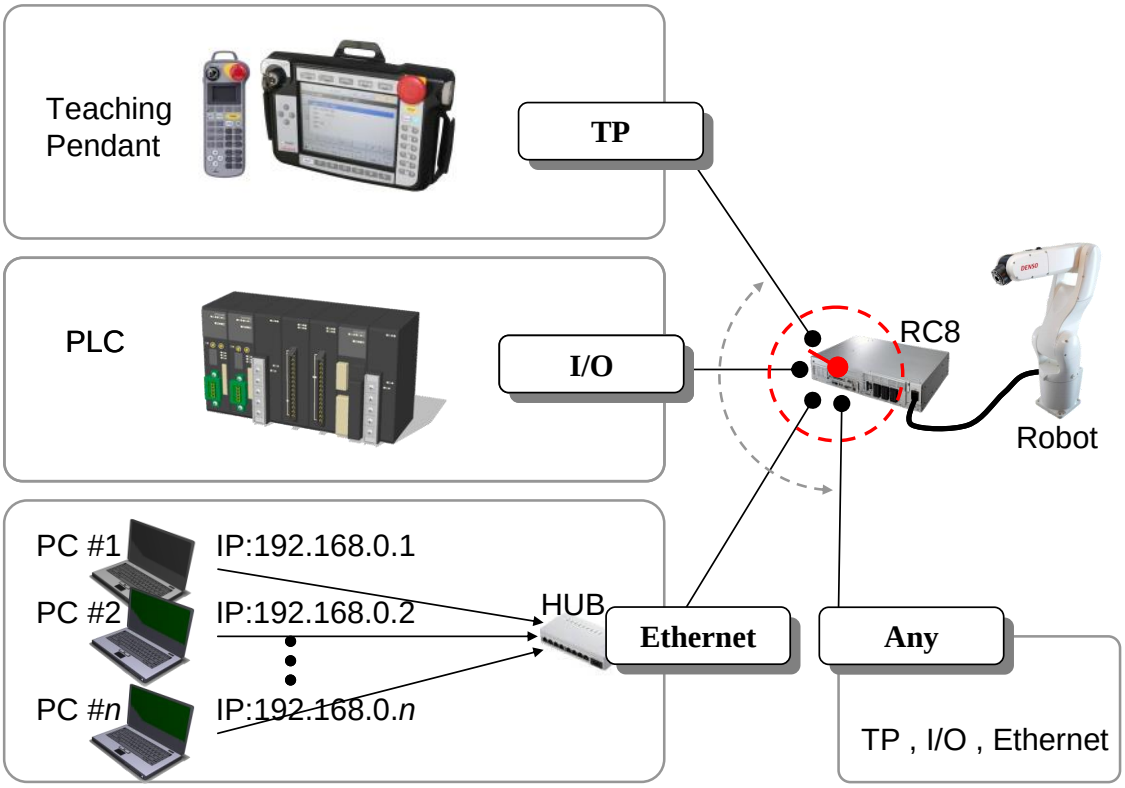


Figure 2-1 Setting of the device with executable token

When using Ethernet as a connection method, you have to setup the IP address of the client PC. This setup enables the robot controller to run the program task or control the robot only from the designated client PC.

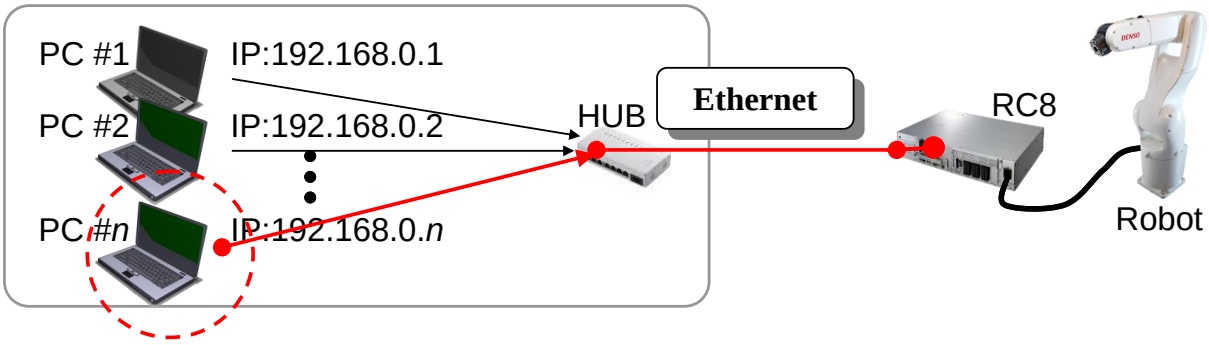


Figure 2-2 Set executable token to Ethernet

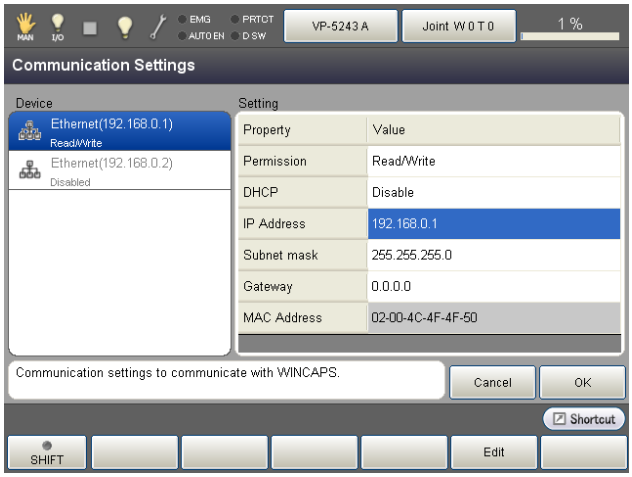


Figure 2-3 Communication permission setting

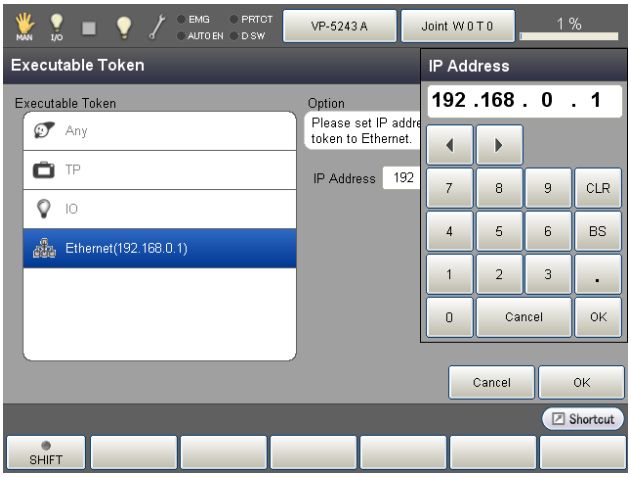


Figure 2-4 Executable token setting

2.2. Changing to the AutoMode

In order to run (execute) a program task of the robot controller, turn on the motor, or control the robot (motion command) from outside client, the robot controller needs to be set to the Auto mode.

To set the robot controller to Auto mode, you need to set the mode selector switch of the Teach pendant (or the Mini-pendant) to the position which is described in Figure 2-5.

Figure 2-5 Setting of Auto mode

3. Communication procedure

This chapter describes how to communicate with a robot controller by using b-CAP for RC8.

3.1. Variable access

To access variables, follow the procedure described in Figure 3-1. Each step is described in more detail below.

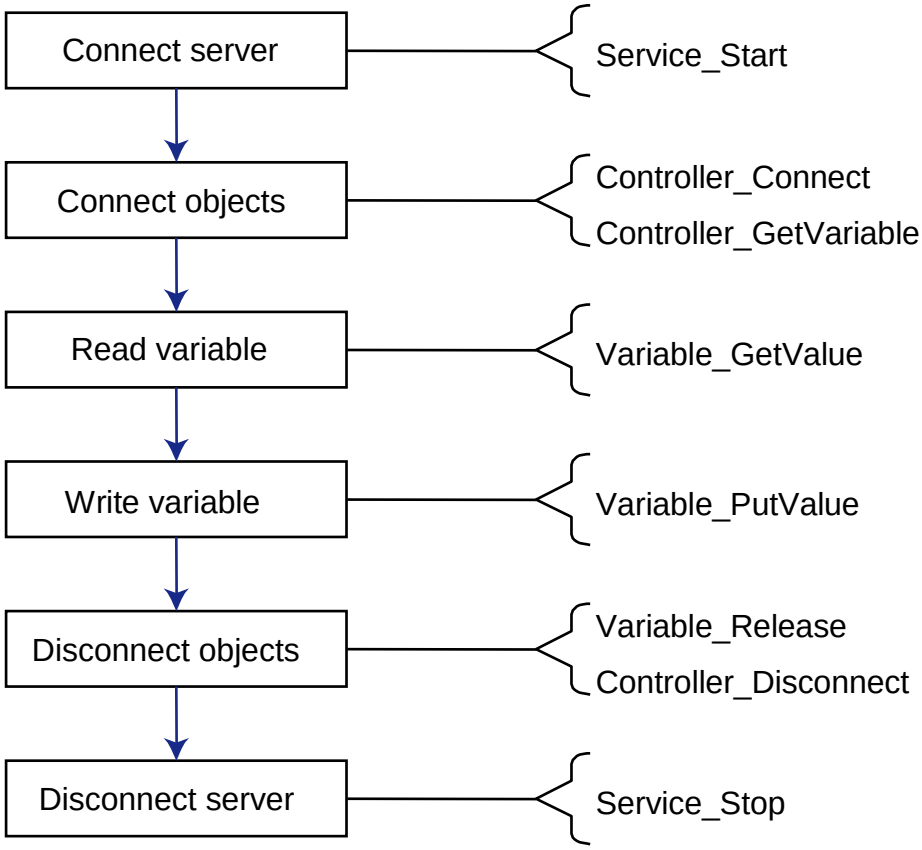


Figure 3-1 Flow of the variable access

3.1.1. Connecting to the server

To start the server service of RC8, send a "Service_Start" packet.

Service_Start				
Packet TX	Client -> Server: 01 10 00 00 00 00 00 00 00 00 01 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-

		-		
Packet RX	Server -> Client: 01 10 00 00 00 00 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

The "Service_Start" packet can be specified option character string as the arguments. The list specified in the option character string is shown as follows.

Table 3-1 Option character string of Service_Start

Option	Meaning
WDT=<Watch dog timer interval>	Watch dog timer interval (ms) of b-CAP server. When you invoke a command which takes longer time than the b-CAP client's timeout,you can avoid raising timeout error before completing the command by specifying this parameter.If this option is set and a certain command took a long time, b-CAP server sends a special packet (executing notify packet) to b-CAP client every specified interval for resetting the timeout count. The value should be more than 80ms and less than the b-CAP client's timeout.

If you specify the option character string, then you send the following packet.

Service_Start				
Packet TX	Client -> Server 01 2c 00 00 00 01 00 00 00 00 00 00 01 00 18 00 00 00 08 00 01 00 00 00 00 0e 00 00 00 57 00 44 00 54 00 3d 00 34 00 30 00 30 00 04			
	Argument	Description	Data Type	Value
		Binary		
	bstrOption	Option character string	VT_BSTR	"WDT=400"
		0e 00 00 00 57 00 44 00 54 00 3d 00 34 00 30 00 30 00		
Packet RX	Server -> Client: 01 10 00 00 00 00 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.1.2. Connecting to the objects

Connect to each object to acquire the handle of the controller object. Controller handle is necessary to connect variable objects of the controller. To connect to the controller, use Contoller_Connect (3) as function ID. To connect to the variable object of the controller, use Controller_GetVariable (9) as function ID. The following table shows the example of each packet. In this example, connect to the controller of IP: 192.168.0.1, then acquire a handle which system variable is "IO150". Use IP address that is set to each controller.

<table>
<tr><th colspan="4">Controller_Connect</th></tr>
<tr><td colspan="4">This function connects to the controller, and then returns the handle of the controller object (hController).</td></tr>
<tr><td>Packet
TX</td><td colspan="3">Client -> Server:
01 8A 00 00 00 01 00 00 00 03 00 00 00 04 00 14
00 00 00 08 00 01 00 00 00 0A 00 00 00 62 00 2D
00 43 00 41 00 50 00 2C 00 00 00 08 00 01 00 00
00 22 00 00 00 43 00 61 00 6F 00 50 00 72 00 6F
00 76 00 2E 00 44 00 45 00 4E 00 53 00 4F 00 2E
00 56 00 52 00 43 00 20 00 00 00 08 00 01 00 00
00 16 00 00 00 31 00 39 00 32 00 2E 00 31 00 36
00 38 00 2E 00 30 00 2E 00 31 00 0A 00 00 00 08
00 01 00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="8"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>bstrCtrlName</td><td>Controller name</td><td>VT_BSTR</td><td>"b-CAP"</td></tr>
<tr><td></td><td colspan="3">14
00 00 00 08 00 01 00 00 00 0A 00 00 00 62 00 2D
00 43 00 41 00 50 00</td></tr>
<tr><td>bstrProvName</td><td>Provider name</td><td>VT_BSTR</td><td>"CaoProv.DENSO.VRC"</td></tr>
<tr><td></td><td colspan="3">2C 00 00 00 08 00 01 00 00
00 22 00 00 00 43 00 61 00 6F 00 50 00 72 00 6F
00 76 00 2E 00 44 00 45 00 4E 00 53 00 4F 00 2E
00 56 00 52 00 43 00</td></tr>
<tr><td>bstrPcName</td><td>Client PC name</td><td>VT_BSTR</td><td>"192.168.0.1"</td></tr>
<tr><td></td><td colspan="3">20 00 00 00 08 00 01 00 00
00 16 00 00 00 31 00 39 00 32 00 2E 00 31 00 36
00 38 00 2E 00 30 00 2E 00 31 00</td></tr>
<tr><td>bstrOption</td><td>Connection option</td><td>VT_BSTR</td><td>Null String</td></tr>
<tr><td></td><td colspan="3">0A 00 00 00 08
00 01 00 00 00 00 00 00 00</td></tr>
<tr><td>Packet
RX</td><td colspan="3">Server -> Client:
01 1E 00 00 00 01 00 00 00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 04</td></tr>
<tr><td rowspan="4"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hController</td><td>Controller handle</td><td>VT_I4</td><td>0x00000002</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 02 00 00 00</td></tr>
</table>

<table>
<tr><th colspan="5">Controller_GetVariable</th></tr>
<tr><td colspan="5">This function returns the handle of the variable object (hVariable).</td></tr>
<tr><td rowspan="8">Packet
TX</td><td colspan="4">Client -> Server:
01 44 00 00 00 03 00 00 00 09 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 14 00 00
00 08 00 01 00 00 00 0A 00 00 00 49 00 4F 00 31
00 35 00 30 00 0A 00 00 00 08 00 01 00 00 00 00
00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hController</td><td>Controller handle</td><td>VT_I4</td><td>0x00000002</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 02 00 00 00 0A</td></tr>
<tr><td>bstrName</td><td>Variable name</td><td>VT_BSTR</td><td>"IO150"</td></tr>
<tr><td></td><td colspan="3">00 08 00 01 00 00 00 0A 00 00 00 49 00 4F 00 31
00 35 00 30 00 14 00 00</td></tr>
<tr><td>bstrOption</td><td>Option string</td><td>VT_BSTR</td><td>Null String</td></tr>
<tr><td></td><td colspan="3">00 00 00 00 08 00 01 00 00 00 00 0A 00 00</td></tr>
<tr><td rowspan="6">Packet
RX</td><td colspan="4">Server -> Client:
01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hVariable</td><td>Variable handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
</table>

3.1.3. Reading and writing a variable

Read and write values of the variables to be connected. To acquire the value, use "Variable_GetValue (101)" as function ID. To set the value, use "Variable_PutValue (102)" as function ID. The following table shows the example of each packet.

<table>
<tr><th colspan="5">Variable_GetValue</th></tr>
<tr><td colspan="5">This function gets the value of the variable specified by "hVariable".</td></tr>
<tr><td rowspan="6">Packet
TX</td><td colspan="4">Client -> Server:
01 1E 00 00 00 04 00 00 00 65 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hVariable</td><td>Variable handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
</table>

Packet RX	Server -> Client: 01 1C 00 00 00 04 00 00 00 00 00 00 00 01 00 08 00 00 00 0B 00 01 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	pVal	The value of variable "IO150"	VT_BOOL	0x0000 (FALSE)
		00 00 00 0B 00 01 00 00 00 00 00 08		

Variable_PutValue			
This function puts the value to the variable specified by "hVariable".			
Packet TX	Client -> Server: 01 2A 00 00 00 05 00 00 00 66 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 08 00 00 00 0B 00 01 00 00 00 FF FF 04		
	Argument	Description	Data Type	Value
		Binary		
	hVariable	Variable handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
	newVal	The value to put	VT_BOOL	0xFFFF (TRUE)
		00 0B 00 01 00 00 00 FF FF 08 00 00		
Packet RX	Server -> Client: 01 10 00 00 00 05 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.1.4. Disconnecting from an object

Disconnect from an connected object. To disconnect the variable object, use Variable_Release (111) as function ID. To disconnect the controller object, use Controller_Disconnect (4) as function ID. The following table shows the example of each packet.

Variable_Release			
This function disconnects the client from the variable object specified by the handle of the variable "hVariable".			
Packet TX	Client -> Server: 01 1E 00 00 00 06 00 00 00 6F 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04		
	Argument	Description	Data Type	Value

		Binary		
	hVariable	Variable handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
Packet RX	Server -> Client: 01 10 00 00 00 06 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

Controller_Disconnect				
This function disconnects the client from the controller object specified by the handle of the controller "hController".				
Packet TX	Client -> Server: 01 1E 00 00 00 07 00 00 00 04 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hController	Controller handle	VT_I4	0x00000002
		00 00 00 03 00 01 00 00 00 02 00 00 00 0A		
Packet RX	Server -> Client: 01 10 00 00 00 07 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.1.5. Disconnecting from the server

Sending the "Service_Stop" packet stops a server service of RC8.

Service_Stop				
Packet TX	Client -> Server: 01 10 00 00 00 08 00 00 00 02 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		
Packet	Server -> Client: 01 10 00 00 00 08 00 00 00 00 00 00 00 00 00 04			

RX	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.1.6. Access to other variables

RC8 has various variables, such as I type variables, I/O variables. For details about variables supported by RC8, refer to the "RC8 Provider Guide 5.3. Variable list." This section describes how to translate variable access procedures written in "RC8 Provider Guide" into b-CAP's expression with concrete examples.

In "RC8 Provider Guide", the way to access variable "@MODE", which is a controller class system variable, is expressed as follows..

```
---------------------------------------------------------------------------------------------------------------------------
        Dim caoVar as CaoVariable
        Set caoVar = caoCtrl.AddVariable("@MODE","")    'Specify a system variable @MODE
---------------------------------------------------------------------------------------------------------------------------
```

In b-CAP, the above code is expressed as follows.

Packet TX	Client -> Server: 01 44 00 00 00 02 00 00 00 09 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 14 00 00 00 08 00 01 00 00 00 0A 00 00 00 40 00 4D 00 4F 00 44 00 45 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04		
	Argument	Description	Data Type	Value
		Binary		
	hController	Controller handle	VT_I4	0x00000002
		00 00 00 03 00 01 00 00 00 02 00 00 00 0A		
	bstrName	Variable name	VT_BSTR	"@MODE"
		00 08 00 01 00 00 00 0A 00 00 00 40 00 4D 00 4F 14 00 00 00 44 00 45 00		
	bstrOption	Option string	VT_BSTR	Null String
		00 00 00 0A 00 00 00 08 00 01 00 00 00 00		
Packet RX	Server -> Client: 01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hVariable	Variable handle	VT_I4	0x00000003

		0A 00 00 00 03 00 01 00 00 00 03 00 00 00

The b-CAP funciont ID corresponds to the Method of RC8 provider. In this case, Controller_GetVariable (9) corresponds to caoCtrl.AddVariable. Since CAO class objects, such as caoCtrl or caoVar, correspond to the b-CAP object handles, in this sample, caoCtrl corresponds to the controller handle, and caoVar corresponds to the variable handle.

3.1.7. Sample program

The following shows the sample program of variable acces, using ANSI-C sample library.

This sample program reads/writes the variable IO150 (the 150th I/O variable). IP should be set to the value for the target controller. This sample program uses the following value.

IP:192.168.0.1

List 3-1 bCapVariable.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM           5007


int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hVar;
	HRESULT hr;

	/* Open socket */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
	bstrName = SysAllocString(L"");
	bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
	bstrMachine = SysAllocString(L"localhost");
	bstrOpt = SysAllocString(L"");
	hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
	SysFreeString(bstrName);
	SysFreeString(bstrProv);
	SysFreeString(bstrMachine);
	SysFreeString(bstrOpt);
	if FAILED(hr) return (hr);

	/* Get variable handle */
	BSTR bstrVarName, bstrVarOpt;
	bstrVarName = SysAllocString(L"IO150");
```

```
        bstrVarOpt = SysAllocString(L"");
        hr = bCap_ControllerGetVariable(fd, hCtrl, bstrVarName, bstrVarOpt, &hVar);
        SysFreeString(bstrVarName);
        SysFreeString(bstrVarOpt);
        if FAILED(hr) return (hr);

        /* Read variable */
        bCap_VariableGetValue(fd, hVar, &vntResult);

        /* Write variable */
        vntResult.lVal = -1L;
        vntResult.vt = VT_I4;
        bCap_VariablePutValue(fd, hVar, vntResult);

        /* Release variable handle */
        bCap_VariableRelease(fd, &hVar);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

3.2. Task control

Figure 3-2 shows the flow of task control. To start a task, the controller should be set to Auto mode. Executable token of the controller should be set to IP address of the client PC. For details, refer to "2.Setup of RC8". Details of each steps are described below.

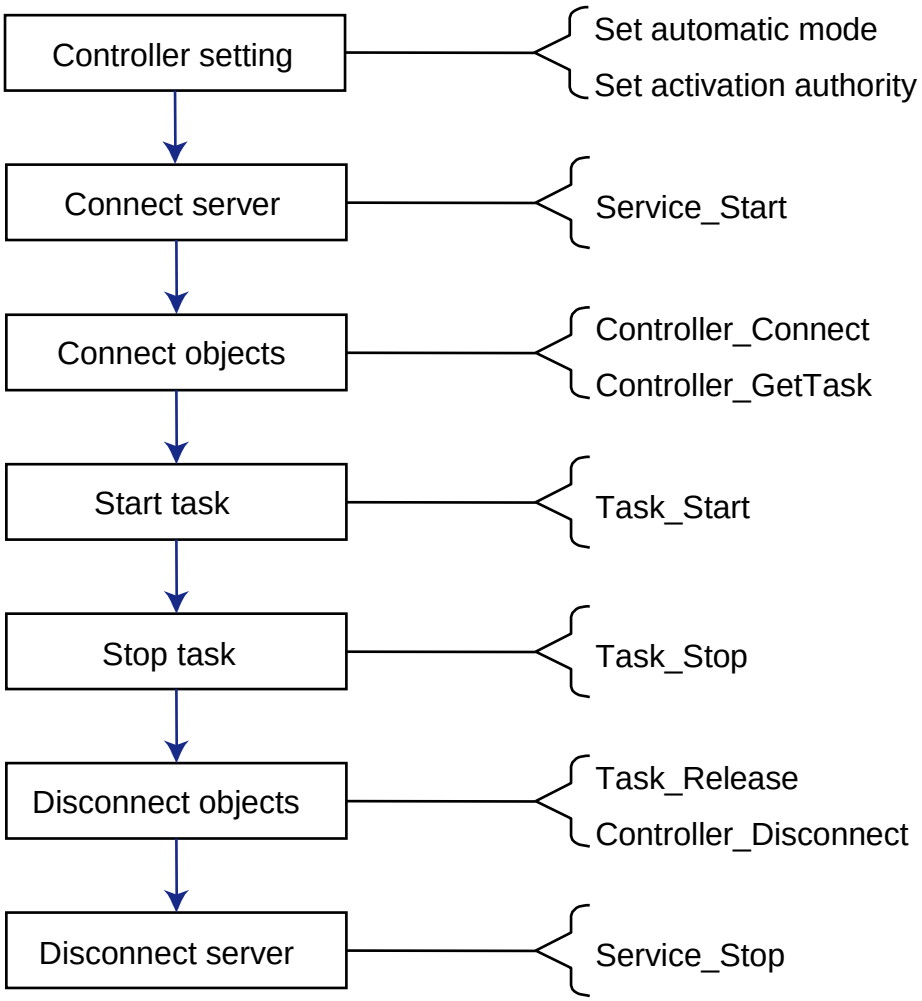


Figure 3-2 Flow of the task control

3.2.1. Connecting to the object

With regards to the procedures until connecting Controller object, refer to "3.1Variable access". To connect to a Task object, use Controller_GetTask (8) as function ID. A controller handle is required as argument as well. The following table shows a packet to acquire a handle of the task "PRO1".

Controller_GetTask	
This function gets the handle of the task object- hTask.	
Packet TX	Client -> Server: 01 42 00 00 00 03 00 00 00 08 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 12 00 00 00 08 00 01 00 00 00 08 00 00 00 50 00 72 00 6F 00 31 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04

	Argument	Description	Data Type	Value
		Binary		
	hController	Controller handle	VT_I4	0x00000002
		00 00 00 03 00 01 00 00 00 02 00 00 00 0A		
	bstrName	Task name	VT_BSTR	"Pro1"
		00 08 00 01 00 00 00 08 00 00 00 50 00 72 00 6F 00 31 00 12 00 00		
	bstrOption	Option string	VT_BSTR	Null String
		00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 00		
Packet RX	Server -> Client: 01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hTask	Task handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		

3.2.2. Start and Stop of a task

Start and stop a connected task. To start a task, use Task_Start (88) as function ID. To stop a task, use Task_Stop (89) as function ID. The following table shows the example of each packet.

Task_Start				
This function starts the task in one cycle execution.				
Packet TX	Client -> Server: 01 3A 00 00 00 04 00 00 00 00 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hTask	Task handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
	lMode	Start mode (=One cycle execution)	VT_I4	0x00000002
		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		
	bstrOption	Option string	VT_BSTR	Null String
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		

Packet RX	Server -> Client: 01 10 00 00 00 04 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

Task_Stop				
This function stops the task in cycle stop.				
Packet TX	Client -> Server: 01 3A 00 00 00 05 00 00 00 59 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hTask	Task handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 0A 00 03 00 00 00		
	lMode	Stop mode (3:Cycle stop)	VT_I4	0x00000003
		00 03 00 01 00 00 00 03 0A 00 00 00		
	bstrOption	Option string	VT_BSTR	Null String
		00 01 00 00 00 00 00 00 0A 00 00 00 08 00		
Packet TX	Server -> Client: 01 10 00 00 00 05 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.2.3. Disconnecting from an object

Disconnect from an object connected. With regards to the procedures after a task object disconnection, refer to "3.1Variable access". To disconnect a task object, use Task_Release (99) as function ID. The following table shows a packet to disconnect a task object.

Task_Release
This function disconnects the client from the task object specified by the handle of the task - "hTask".

<table>
<tr><td rowspan="5">Packet
TX</td><td colspan="4">Client -> Server:
01 1E 00 00 00 06 00 00 00 63 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 04</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>hTask</td><td>Task handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
<tr><td rowspan="5">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 06 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

3.2.4. Sample program

Following is a sample program for task control with using ANSI-C sample library.

The sample program controls a task "PRO01" (continuous execution and cycle stop).

List 3-2 bCapTask.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM            5007

int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hTask;
	int32_t lMode;
	HRESULT hr;

	/* Init and Start b-CAP */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
	bstrName = SysAllocString(L"");
	bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
	bstrMachine = SysAllocString(L"localhost");
	bstrOpt = SysAllocString(L"");
	hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
	SysFreeString(bstrName);
	SysFreeString(bstrProv);
```

```
        SysFreeString(bstrMachine);
        SysFreeString(bstrOpt);
        if FAILED(hr) return (hr);

        /* Get task handle */
        BSTR bstrTskName, bstrTskOpt;
        bstrTskName = SysAllocString(L"Pro1");
        bstrTskOpt = SysAllocString(L"");
        hr = bCap_ControllerGetTask(fd, hCtrl, bstrTskName, bstrTskOpt, &hTask);
        SysFreeString(bstrTskName);
        SysFreeString(bstrTskOpt);
        if FAILED(hr) return (hr);

        /* Start task */
        lMode = 2L;
        bCap_TaskStart(fd, hTask, lMode, bstrTskOpt);

        /* Stop task */
        lMode = 3L;
        bCap_TaskStop(fd, hTask, lMode, bstrTskOpt);

        /* Release task handle */
        bCap_TaskRelease(fd, &hTask);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

3.3. Robot control

When executing robot control, you need to follow the procedure shown in Figure 3-3. In order to operate the robot motion, the controller needs to be set to the Auto mode. The executable token of the controller needs to be set to the IP of the client PC as well. For details, refer to "2.Setup of RC8". Each step is described in more detail below.

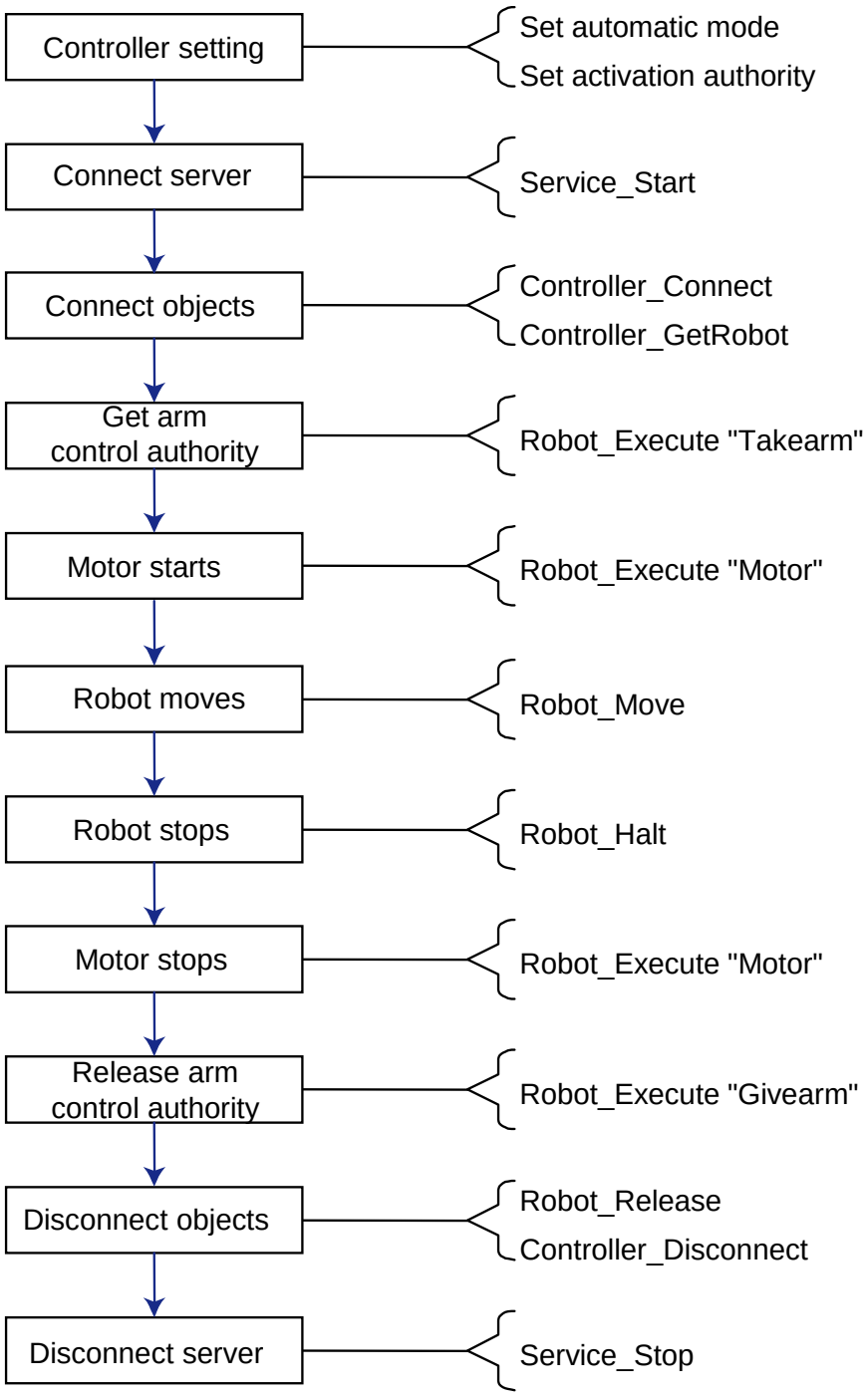


Figure 3-3 Flow of the robot control

3.3.1. Connecting to an object

With regards to the procedures until connecting to a controller object, refer to "3.1Variable access". To connect to a robot object, use Controller_GetRobot (7) as function ID. A controller handle is required as argument as well. The following table shows a packet to connect to the robot object.

<table>
<tr><th colspan="5">Controller_GetRobot</th></tr>
<tr><td colspan="5">This function gets the handle of the robot object (hRobot).</td></tr>
<tr><td rowspan="8">Packet
TX</td><td colspan="4">Client -> Server:
01 40 00 00 00 02 00 00 00 07 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 10 00 00
00 08 00 01 00 00 00 06 00 00 00 41 00 72 00 6D
00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>hController</td><td>Controller handle</td><td>VT_I4</td><td>0x00000002</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 02 00 00 00 0A</td></tr>
<tr><td>bstrName</td><td>Robot name</td><td>VT_BSTR</td><td>"Arm"</td></tr>
<tr><td></td><td colspan="3">00 08 00 01 00 00 00 06 00 00 00 41 00 72 00 6D 00 10 00 00</td></tr>
<tr><td>bstrOption</td><td>Option string</td><td>VT_BSTR</td><td>Null String</td></tr>
<tr><td></td><td colspan="3">0A 00 00 00 08 00 01 00 00 00 00 00 00 00</td></tr>
<tr><td rowspan="7">Packet
RX</td><td colspan="4">Server -> Client:
01 1E 00 00 00 03 00 00 00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
</table>

3.3.2. Taking and releasing the arm control authority

When executing the robot control, the arm control authority of the robot needs to be obtained. Arm control authority must be released before disconnecting from the controller. Each of them are mounted as a command of Robot_Execute (64). The following table shows the example of each packet.

<table>
<tr><th colspan="5">Robot_Execute "Takearm", (0, 1)</th></tr>
<tr><td colspan="5">This function takes the arm control authority.</td></tr>
<tr><td rowspan="8">Packet
TX</td><td colspan="4">Client -> Server:
01 4C 00 00 00 05 00 00 00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 18 00 00
00 08 00 01 00 00 00 0E 00 00 00 54 00 61 00 6B
00 65 00 61 00 72 00 6D 00 0E 00 00 00 03 20 02
00 00 00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
</table>

	bstrCommand	Command string	VT_BSTR	"Takearm"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 54 00 61 00 6B 00 65 00 61 00 72 00 6D 00		
	vntParam	Command parameter	VT_I4 | VT_ARRAY	0, 1
		0E 00 00 00 03 20 02 00 00 00 00 00 00 00 01 00 00 00		
Packet RX	Server -> Client: 01 1A 00 00 00 05 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

Robot_Execute "Givearm"				
This function releases the arm control authority				
Packet TX	Client -> Server: 01 44 00 00 00 0A 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 18 00 00 00 08 00 01 00 00 00 0E 00 00 00 47 00 69 00 76 00 65 00 61 00 72 00 6D 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hRobot	Robot handle	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	Command string	VT_BSTR	"Givearm"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 47 00 69 00 76 00 65 00 61 00 72 00 6D 00		
	vntParam	Parameter	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		
Packet RX	Server -> Client: 01 1A 00 00 00 0A 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

3.3.3. Turning On/Off motors

In order to control a robot, motors of the robot need to turn ON. The motor control is mounted as a command of Robot_Execute (64). The following table shows the example of packets that turn ON/OFF the motor.

Robot_Execute "Motor", (1, 0)				
This function turns On a motor.				
Packet TX	Client -> Server: 01 48 00 00 00 06 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 14 00 00 00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00 0E 00 00 00 03 20 02 00 00 00 01 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hRobot	Robot handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 0A		
	bstrCommand	Command string	VT_BSTR	"Motor"
		00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00 14 00 00		
	vntParam	Parameter	VT_I4 | VT_ARRAY	1, 0
		00 00 00 00 00 00 00 0E 00 00 00 03 20 02 00 00 00 01		
Packet RX	Server -> Client: 01 1A 00 00 00 06 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		00 00 00 00 00 01 00 00 00 06		

Robot_Execute "Motor", (0, 0)				
This function turns Off a motor.				
Packet TX	Client -> Server: 01 48 00 00 00 09 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 14 00 00 00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00 0E 00 00 00 03 20 02 00 00 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		

	hRobot	Robot handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 0A		
	bstrCommand	Command string	VT_BSTR	"Motor"
		00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00 14 00 00		
	vntParam	Parameter	VT_I4 | VT_ARRAY	0, 0
		00 00 00 00 00 00 00 0E 00 00 00 03 20 02 00 00 00 00		
Packet RX	Server -> Client: 01 1A 00 00 00 09 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		00 00 00 00 00 01 00 00 00		06

3.3.4. Moving and halting a robot

In order to move a robot, use Robot_Move (72) as function ID. For details about Move command, refer to RC8 Provider Guide. When using Move command with "NEXT" option, a robot can be halted by using Robot_Halt (70) as function ID. The following table shows the example of each packet. In this case, the robot is moved to the position stored in P1 (the first of the P type variable) with NEXT option.

Robot_Move 1, "P1", "NEXT"				
This executes the motion command "MOVE 1, "P1" "NEXT"".				
Packet TX	Client -> Server: 01 54 00 00 00 07 00 00 00 48 00 00 00 04 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 0E 00 00 00 08 00 01 00 00 00 04 00 00 00 50 00 31 00 12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hRobot	Robot handle	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00		0A
	lComp	Interpolation mode	VT_I4	0x00000001
		00 03 00 01 00 00 00 01 00 00 00		0A 00 00
	vntPose	Posture	VT_BSTR	"P1"
		00 01 00 00 00 04 00 00 00 50 00 31 00		0E 00 00 00 08

	bstrOption	Motion option	VT_BSTR	"NEXT"
		12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00		
Packet RX	Server -> Client: 01 10 00 00 00 07 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

Robot_Halt			
This function halts a robot.			
Packet TX	Client -> Server: 01 2C 00 00 00 08 00 00 00 46 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	hRobot	Robot handle	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrOption	Option string	VT_BSTR	Null String
		0A 00 00 00 08 00 01 00 00 00 00 00 00 00		
Packet RX	Server -> Client: 01 10 00 00 00 08 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.3.5. Disconnecting from an object

Disconnect from an object connected. Regarding to the procedure after the robot disconnection, refer to "3.1Variable access". To disconnect a robot object, use Robot_Release (84) as function ID. The following table shows a packet to disconnect a robot object.

Robot_Release
This function disconnects a client from a robot object specified by the handle of the robot "hRobot".

<table>
<tr><td rowspan="5">Packet
TX</td><td colspan="4">Client -> Server:
01 1E 00 00 00 0B 00 00 00 54 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 04</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td colspan="4">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
<tr><td rowspan="5">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 0B 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="4">-</td></tr>
</table>

3.3.6. Other execution methods

RC8 has provider-specific extended commands for each CAO class object. For details about the extended commands supported by RC8, refer to "RC8 Provider Guide 5.2.11 CaoController::Execute method" etc. This section describes how to translate the procedure of the extended command execution written in "RC8 Provider Guide" into b-CAP with concrete examples.

In "RC8 Provider Guide", the procedure to execute "ClearError", which is the extended command in controller class, is expressed as follows.

```
caoCtrl.Execute "ClearError"
```

In b-CAP, the above code is expressed as follows.

<table>
<tr><td rowspan="6">Packet
TX</td><td colspan="4">Client -> Server:
01 4A 00 00 00 12 00 00 00 11 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 1E 00 00
00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65
00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00 06
00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>hController</td><td>Controller hanle</td><td>VT_I4</td><td>0x0000002</td></tr>
<tr><td colspan="4">00 00 00 03 00 01 00 00 00 02 00 00 00 0A</td></tr>
<tr><td rowspan="2">bstrCommand</td><td>Command string</td><td>VT_BSTR</td><td>"ClearError"</td></tr>
<tr><td colspan="4">00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65
00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00 1E 00 00</td></tr>
</table>

	vntParam	Parameter	VT_EMPTY	
		00 00 00 00 00 01 00 00 00 06		
Packet RX	Server -> Client: 01 1A 00 00 00 12 00 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		00 00 00 00 00 01 00 00 00 06		

The b-CAP funciont ID corresponds to the Method of RC8 provider. In this case, Controller_Execute (17) corresponds to caoCtrl.Execute. CAO class objects such as caoCtrl correspond to the b-CAP object handles. In this case, caoCtrl correspond to the controller handle. The name of extended commands, such as "ClearError", can be translated into b-CAP by adding byte arrays, which are converted from command name strings, after the object handle. If the extended command requires parameters, add byte arrays, which are converted from the parameters, after the command name strings.

3.3.7. Sample program

Following is a sample program, which controls a robot, with using ANSI-C sample library.

The sample program moves a robot to the position stored in P1 (1st element of P-type variable).

List 3-3 bCapRobot.c

```
#include <atlbase.h>

#include "stdint.h"
#include "bCAPClient/bcap_client.h"

#define SERVER_IP_ADDRESS "tcp:192.168.0.1"
#define SERVER_PORT_NUM            5007

int main()
{
	int fd;
	VARIANT vntResult;
	uint32_t hCtrl, hRobot;
	HRESULT hr;

	/* Init socket */
	hr = bCap_Open_Client(SERVER_IP_ADDRESS, SERVER_PORT_NUM, 0, &fd);
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(fd, NULL);
	if FAILED(hr) return (hr);

	/* Get controller handle */
	BSTR bstrName, bstrProv, bstrMachine, bstrOpt;
```

```
bstrName = SysAllocString(L"");
bstrProv = SysAllocString(L"CaoProv.DENSO.VRC");
bstrMachine = SysAllocString(L"localhost");
bstrOpt = SysAllocString(L"");

/* Connect controller */
hr = bCap_ControllerConnect(fd, bstrName, bstrProv, bstrMachine, bstrOpt, &hCtrl);
SysFreeString(bstrName);
SysFreeString(bstrProv);
SysFreeString(bstrMachine);
SysFreeString(bstrOpt);
if FAILED(hr) return (hr);

/* Get robot handle */
BSTR bstrRobotName, bstrRobotOpt;
bstrRobotName = SysAllocString(L"Arm");
bstrRobotOpt = SysAllocString(L"");
hr = bCap_ControllerGetRobot(fd, hCtrl, bstrRobotName, bstrRobotOpt, &hRobot);
SysFreeString(bstrRobotName);
SysFreeString(bstrRobotOpt);
if FAILED(hr) return (hr);

/* Get arm control authority */
BSTR bstrCommand;
VARIANT vntParam;
bstrCommand = SysAllocString(L"Takearm");
vntParam.bstrVal = SysAllocString(L"");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Motor on */
bstrCommand = SysAllocString(L"Motor");
vntParam.bstrVal = SysAllocString(L"1");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Move to P1 */
VARIANT vntPose;
BSTR bstrMoveOpt;
vntPose.bstrVal = SysAllocString(L"P1");
vntPose.vt = VT_BSTR;
bstrMoveOpt = SysAllocString(L"");
hr = bCap_RobotMove(fd, hRobot, 1L, vntPose, bstrMoveOpt);
VariantClear(&vntPose);
SysFreeString(bstrMoveOpt);
if FAILED(hr) return (hr);

/* Motor off */
bstrCommand = SysAllocString(L"Motor");
vntParam.bstrVal = SysAllocString(L"0");
vntParam.vt = VT_BSTR;
hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
SysFreeString(bstrCommand);
VariantClear(&vntParam);
if FAILED(hr) return (hr);

/* Release arm control authority */
bstrCommand = SysAllocString(L"Givearm");
vntParam.bstrVal = SysAllocString(L"");
vntParam.vt = VT_BSTR;
```

```
        hr = bCap_RobotExecute(fd, hRobot, bstrCommand, vntParam, &vntResult);
        SysFreeString(bstrCommand);
        VariantClear(&vntParam);
        if FAILED(hr) return (hr);

        /* Release robot handle */
        bCap_RobotRelease(fd, &hRobot);

        /* Release controller handle */
        bCap_ControllerDisconnect(fd, &hCtrl);

        /* Stop b-CAP service (Very important in UDP/IP connection) */
        bCap_ServiceStop(fd);

        /* Close socket */
        bCap_Close_Client(&fd);

        return 0;
}
```

4. How to use b-CAP Slave Mode

4.1. Whats the Slave Mode

Slave mode is a function to control a robot by periodically transmitting the position and posture data in very short interval.

The following five functions are installed in b-CAP as Slave Mode.

- slvChangeMode[2]: Change the setting of the Slave Mode.
- slvGetMode: Acquire the current setting of the Slave Mode.
- slvMove: Move a robot to the designated position and posture
- slvSendFormat: Change the parameters format of slvMove command.
- slvRecvFormat: Change the return value format of slvMove command.

4.2. Functions of the Slave Mode

Slave Mode functions are mounted as commands of Robot_Execute.

Function	Robot_Execute		
Function ID	64		
Argument	VT_I4	hRobot	Handle of a robot
	VT_BSTR	bstrCommand	Command string
	VARIANT	vntParam	Parameter
Return Value	VARIANT	pVal	Result
Description	Execute a command of a robot(hRobot).		

A command is executed by entering a command name (see Table 4-1) into "bstrCommand".

Table 4-1 The Slave Mode functions

CommandString	Parameter			Return Value	Behaviour
slvChangeMode	<SlaveMode:VT_I4>			None	Change the Slave Mode settings. To switch to the Slave Mode, a robot must be stopped. The client to be changed to the Slave
	Value	Type	Motion		
	0x000	-	Mode release		
	0x001	P type	Mode 0		
	0x002	J type	Mode 0		
	0x003	T type	Mode 0		
	0x101	P type	Mode 1		

[2] In Slave Mode, only slvGetMode or slvMove commands can be used.

<table>
<tr><td></td><td>0x102 J type Mode 1
0x103 T type Mode 1
0x201 P type Mode 2
0x202 J type Mode 2
0x203 T type Mode 2</td><td></td><td>Mode must have the arm control authority as well.</td></tr>
<tr><td>slvGetMode</td><td>None</td><td><Slave Mode:VT_I4>
See parameter of "slvChangeMode".</td><td>Acquire the current setting of the Slave Mode.</td></tr>
<tr><td>slvMove</td><td>Parameter differs depending on the setting of slvSendFormat and slvRecvFormat. Refer to Table 4-2.</td><td>Return value differs depending on the setting of slvRecvFormat.
Refer to Table 4-3.</td><td>Move a robot to the designated position and posture.</td></tr>
<tr><td>slvSendFormat</td><td><Expansion format:VT_I4><table><tr><td>Value</td><td>Expansion</td></tr><tr><td>0x0000</td><td>None</td></tr><tr><td>0x0020</td><td>HandIO-mode</td></tr><tr><td>0x0100</td><td>MiniIO-mode</td></tr><tr><td>0x0120</td><td>MiniIO + HandIO mode</td></tr><tr><td>0x0200</td><td>User IO mode</td></tr><tr><td>0x0220</td><td>User IO + HandIO mode</td></tr></table></td><td>None</td><td>For output, MiniIO, HandIO, or User IO are available.[3]</td></tr>
<tr><td>slvRecvFormat</td><td><Return value format:VT_I4 | VT_ARRAY>
<First argument><table><tr><td>Value</td><td>Format</td></tr><tr><td>0x0001</td><td>P type</td></tr><tr><td>0x0002</td><td>J type</td></tr><tr><td>0x0003</td><td>T type</td></tr><tr><td>0x0004</td><td>P + J type</td></tr></table></td><td>None</td><td>Obtain current robot position, time stamp, and each IO status (Mini IO, Hand IO, and User IO)3</td></tr>
</table>

[3] If the designated size is large, it may not work normally due to communication delay.

	0x0005 T + J type 0x0010 Time stamp 0x0020 HandIO-status 0x0040 Obtain electric current value 0x0100 MiniIO-status 0x0200 User IO-status <Second argument> 0:Return time stamp by ms (default) 1:Return time stamp by us ※ If it is omitted, it is assumed to be "0".		

Table 4-2 Parameter formats of slvMove command

Expansion format	Parameter type	Parameter
Position only(default)	VT_R8 | VT_ARRAY	Position(VT_R8 | VT_ARRAY)
Position and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of receiving UserIO (VT_I4) Size of receiving UserIO (VT_I4)
Position and HandIO status	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) HandIO status(VT_I4)
Position, HandIO status, and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of receiving UserIO (VT_I4) Size of receiving UserIO (VT_I4) HandIO status(VT_I4)
Position and MiniIO status	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status(VT_I4)
Position, MiniIO status, and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status(VT_I4) Offset of receiving UserIO (VT_I4) Size of receiving UserIO (VT_I4)
Position, MiniIO, and HandIO status	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status(VT_I4) HandIO status(VT_I4)

Position, MiniIO, HandIO, and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status(VT_I4) Offset of receiving UserIO (VT_I4) Size of receiving UserIO (VT_I4) HandIO status(VT_I4)
Position and parameters of sending UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of sending UserIO (VT_I4) Size of sending UserIO (VT_I4) UserIO status(VT_UI1 | VT_ARRAY)
Position, parameters of sending UserIO, and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of sending UserIO (VT_I4) Size of sending UserIO (VT_I4) Sending UserIO status(VT_UI1 | VT_ARRAY) Offset of receiving UserIO (VT_I4) Size of receiving UserIO(VT_I4)
Position, parameters of sending UserIO, and HandIO status	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of sending UserIO (VT_I4) Size of sending UserIO (VT_I4) UserIO status(VT_UI1 | VT_ARRAY) HandIO status (VT_I4)
Position, parameters of sending UserIO, HandIO status, and parameters of receiving UserIO	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Offset of sending UserIO (VT_I4) Size of sending UserIO (VT_I4) Sending UserIO status(VT_UI1 | VT_ARRAY) Offset of receiving UserIO (VT_I4) Size of receiving UserIO(VT_I4) HandIO status (VT_I4)

Table 4-3 Return value formats of slvMove command

Return value pattern	Return value type	Return value

Position only(default)	VT_R8 | VT_ARRAY	Position(VT_R8 | VT_ARRAY)
Position and MiniIO status	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status(VT_I4)
Time stamp and Position	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY)
Time stamp, Position, and MiniIO status	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) MiniIO status (VT_I4)
Position and HandIO static	VT_R8 | VT_ARRAY	Position(VT_R8 | VT_ARRAY) HandIO status(VT_I4)
Position, MiniIO and HandIO statuc	VT_VARIANT | VT_ARRAY	Position(VT_R8 | VT_ARRAY) MiniIO status(VT_I4) HandIO status(VT_I4)
Time stamp, Position, and HandIO status	VT_VARIANT | VT_ARRAY	Time stamp(VT_I4) Position(VT_R8 | VT_ARRAY) HandIO status(VT_I4)
Time stamp, Position, MiniIO, and HandIO status	VT_VARIANT | VT_ARRAY	Time stamp(VT_I4) Position(VT_R8 | VT_ARRAY) MiniIO status(VT_I4) HandIO status(VT_I4)
Position and User IO status	VT_VARIANT | VT_ARRAY	Position(VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY)
Time stamp, Position, and User IO status	VT_VARIANT | VT_ARRAY	Time stamp(VT_I4) Position(VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY)
Position, User IO, and HandIO status	VT_VARIANT | VT_ARRAY	Position(VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY) HandIO status(VT_I4)

Time stamp, Position, User IO, and HandIO status	VT_VARIANT | VT_ARRAY	Time stamp(VT_I4) Position(VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY) HandIO status(VT_I4)
Position and Electric current value	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) Electric current value(VT_R8 | VT_ARRAY)
Position, MiniIO status and Electric current value	VT_VARIANT | VT_ARRAY	Position(VT_R8 | VT_ARRAY) MiniIO status(VT_I4) Electric current value(VT_R8 | VT_ARRAY)
Time stamp, Position, MiniIO and Electric current value	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) MiniIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)
Position, HandIO, and Electric current value	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) HandIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)
Position, MiniIO + HandIO, and Electric current value	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status (VT_I4) HandIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)
Time stamp, Position , HandIO, and Electric current value	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) HandIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)

Time stamp, Position, MiniIO + HandIO, and Electric current value	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) MiniIO status (VT_I4) HandIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)
Position, User IO, and Electric current value	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) User IO status (VT_UI1 | VT_ARRAY) Electric current value (VT_R8 | VT_ARRAY)
Time stamp, Position, User IO, and Electric current value	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY) Electric current value (VT_R8 | VT_ARRAY)
Position, MiniIO + User IO, and Electric current value	VT_VARIANT | VT_ARRAY	Position (VT_R8 | VT_ARRAY) MiniIO status (VT_I4) User IO status (VT_UI1 | VT_ARRAY) Electric current value (VT_R8 | VT_ARRAY)
Time stamp, Position, User IO + HandIO, and Electric current value	VT_VARIANT | VT_ARRAY	Time stamp (VT_I4) Position (VT_R8 | VT_ARRAY) User IO status(VT_UI1 | VT_ARRAY) HandIO status (VT_I4) Electric current value (VT_R8 | VT_ARRAY)

4.3. Summary of the Slave Mode

There are various modes in the Slave Mode depending on the process specifications of the messages. Table 4-2 shows the summary of each mode.

Table 4-4 Slave Mode summary

Mode	Parameter	Number of buffer	Wait until buffer is generated?	Note
Mode 0 Synchronous - without waiting time (Not compartible with RC7)	0x0**	3 (Buffering data is always used)	No	Queue a message which is sent by the client into the buffer. Return the return code immediately according to the buffer state.
Mode 1 asynchronous (Compartible with unsynchronous of RC7)	0x1**	1 (Data is overwritten when buffering)	No	Keep overwriting the buffer with a message which is sent by the client
Mode 2 Synchronous - with waiting time (Similar with synchronous of RC7)	0x2**	3 (Buffering data is always used) (Buffer size of RC7 is 1)	Yes	Queue a message which is sent by the client into the buffer. Return code is not issued until the buffer secures enough space.

Message process specifications of each mode are described as follows.

4.3.1. Mode0

In Mode 0, the coordinate and posture date transmitted by "Robot_Execute "slvMode"" are queued in the buffer of the server. The server returns the return code to the client immediately, according to the queued buffer condition.

Buffer condition	Return code
More than one buffer space	S_OK(0x00000000)
Buffer Full	S_ BUF_FULL (0x0F200501)
Buffer overflow	E_ BUF_FULL (0x83201483)

Figure 4-1 shows the communication flow between server and client at Mode 0.

Client generates a message sending thread in certain intervals. Message sending thread keeps sending "slvMode" message to the server until "S_BUF_FULL" returns from the server as a return code. When "S_BUF_FULL" is returned, the client stops generating the message sending thread because the buffer

becomes full, and waits until the server processes the messages.

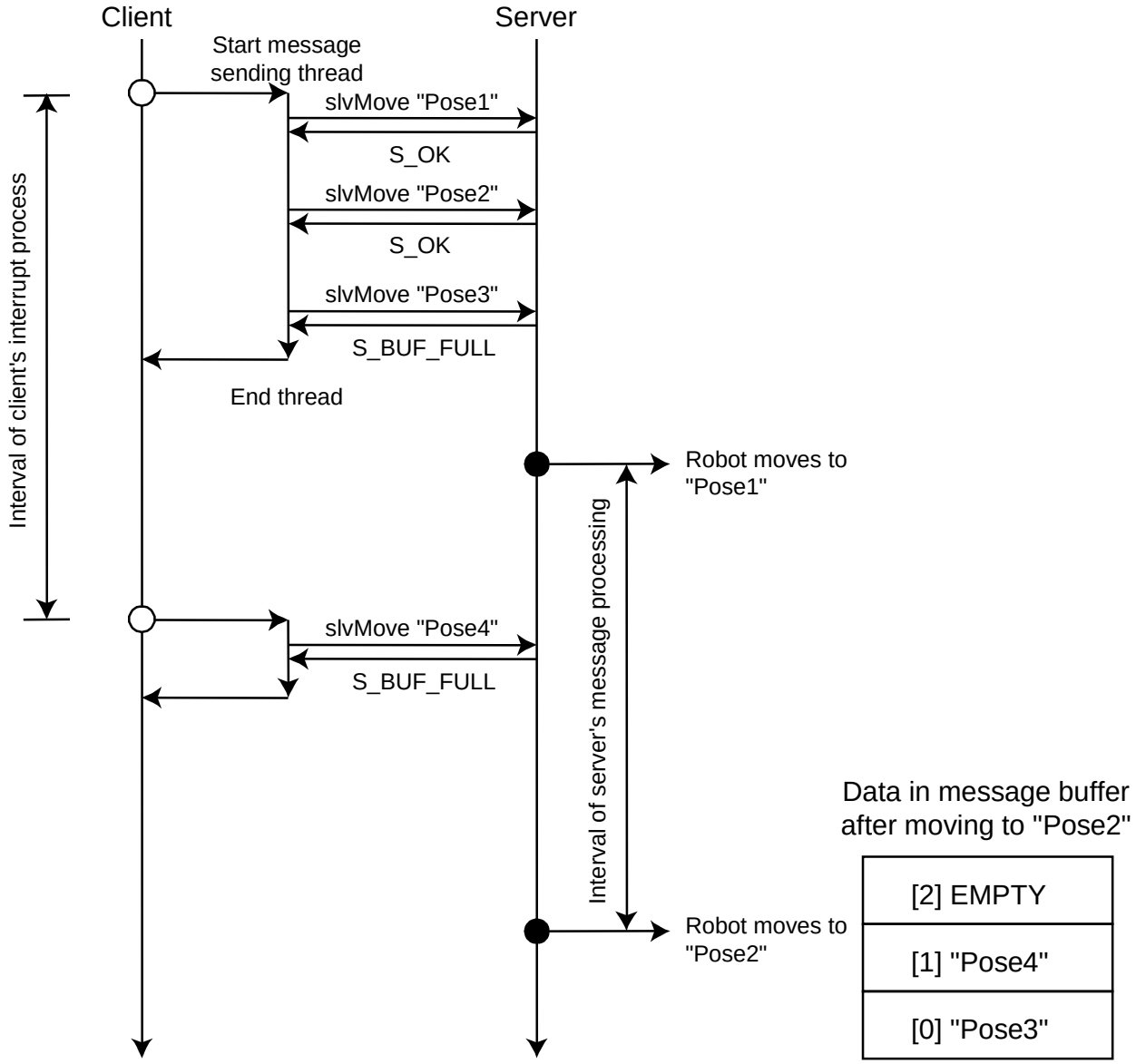


Figure 4-1 The communication procedure in the Mode0

If the "Buffer Overflow" message returns from the server, "slvMode" message which was sent immediately before was not accumulated in buffer. As Figure 4-2 shows, you need to wait until the message has been processed, then re-send the "slvMode" message that triggers "Buffer Overflow" message.

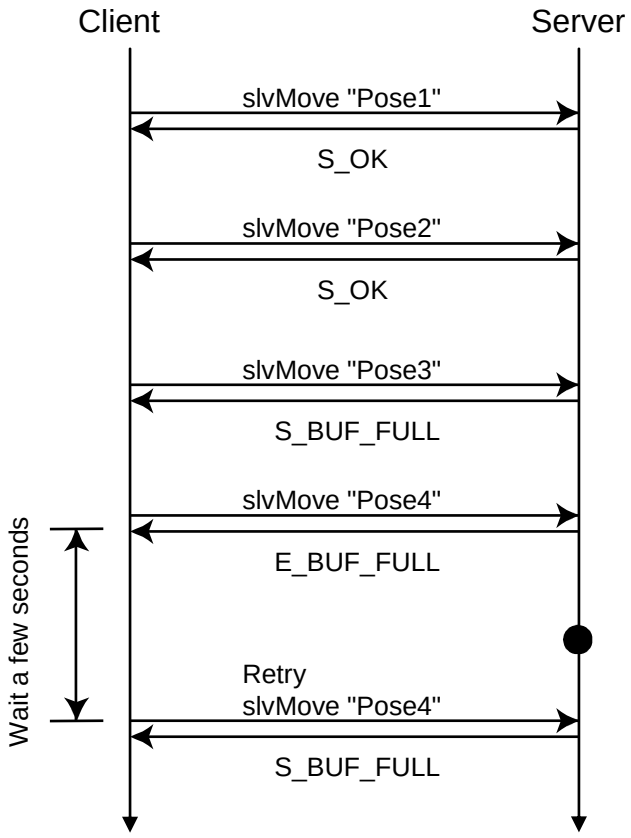


Figure 4-2 Process when buffer overflow occures

4.3.2. Mode1

In Mode 1, the server has only one buffer. Coordinate and position date transmitted by "Robot_Execute"slvMove"" is stored by overwriting the buffer of the server. Figure 4-3 shows the communication flow between server and client at Mode 1.

A "slvMove" message transmitted by the client keeps overwriting the buffer; therefore, the message processed by the server is always the message that has been transmitted from the client immediately before.

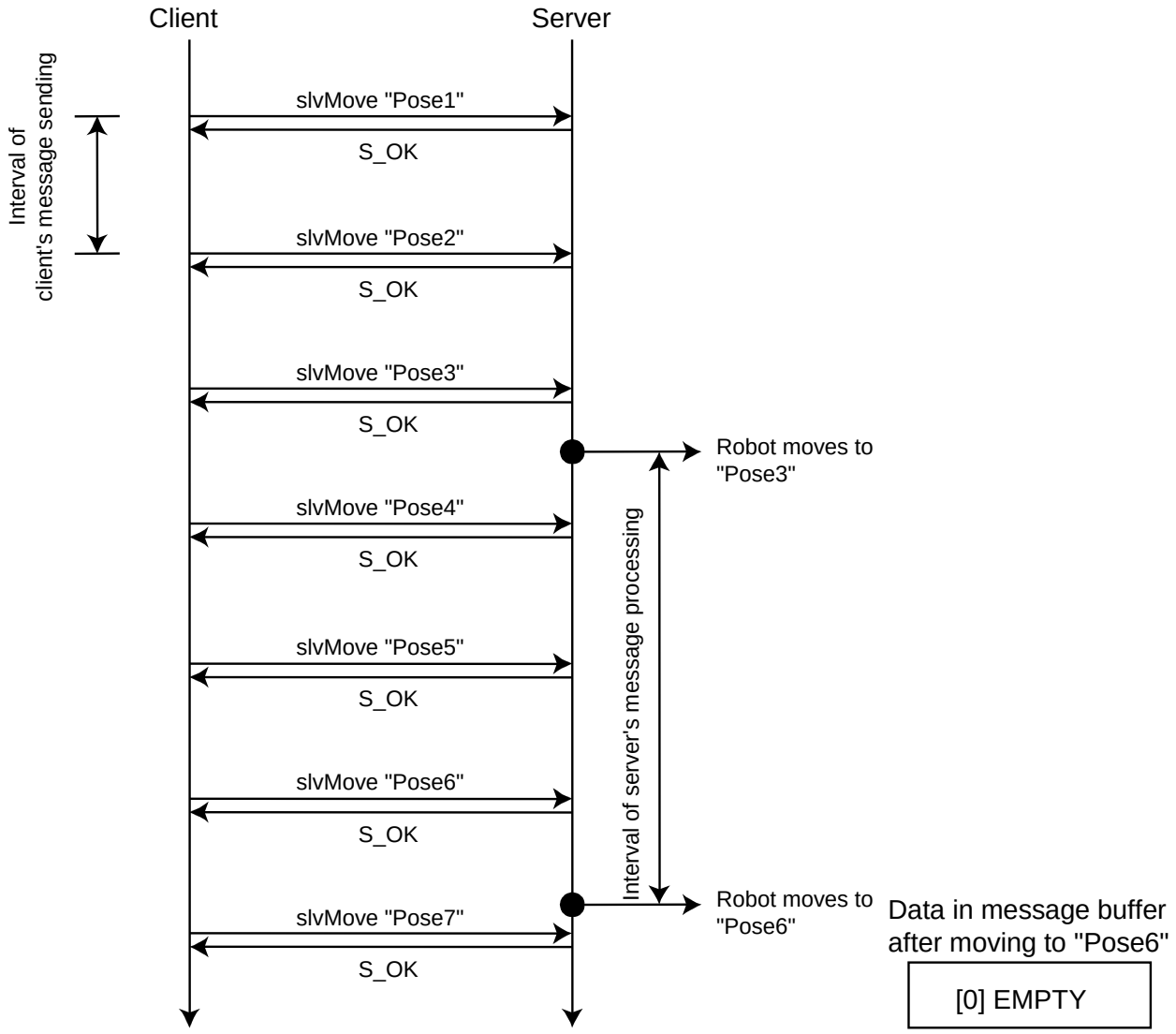


Figure 4-3 The communication procedure in the Mode1

4.3.3. Mode2

In Mode 2, same as Mode 0, the coordinate / posture date transmitted by "Robot_Execute"slvMove"" are queued in the buffer of the server. The server returns the return code to the client immediately, according to the queued buffer condition. The difference between Mode 0 is, if a "slvMove" message arrives while the buffer is full, the server does not send a return code until the buffer space is secured.

Figure 4-4 shows the communication flow between server and client at Mode 2.

In Figure4-4, "slvMove "Pose5"" is transmitted when the buffer is full; therefore, the server does not send return code until it moves the robot to "Pose2". As a result, the client becomes stand-by state automatically until the buffer space is secured. By this, the client can achieve Slave Mode without mounting the processing such as "Suspending the thread by monitoring the return code" like Mode 0.

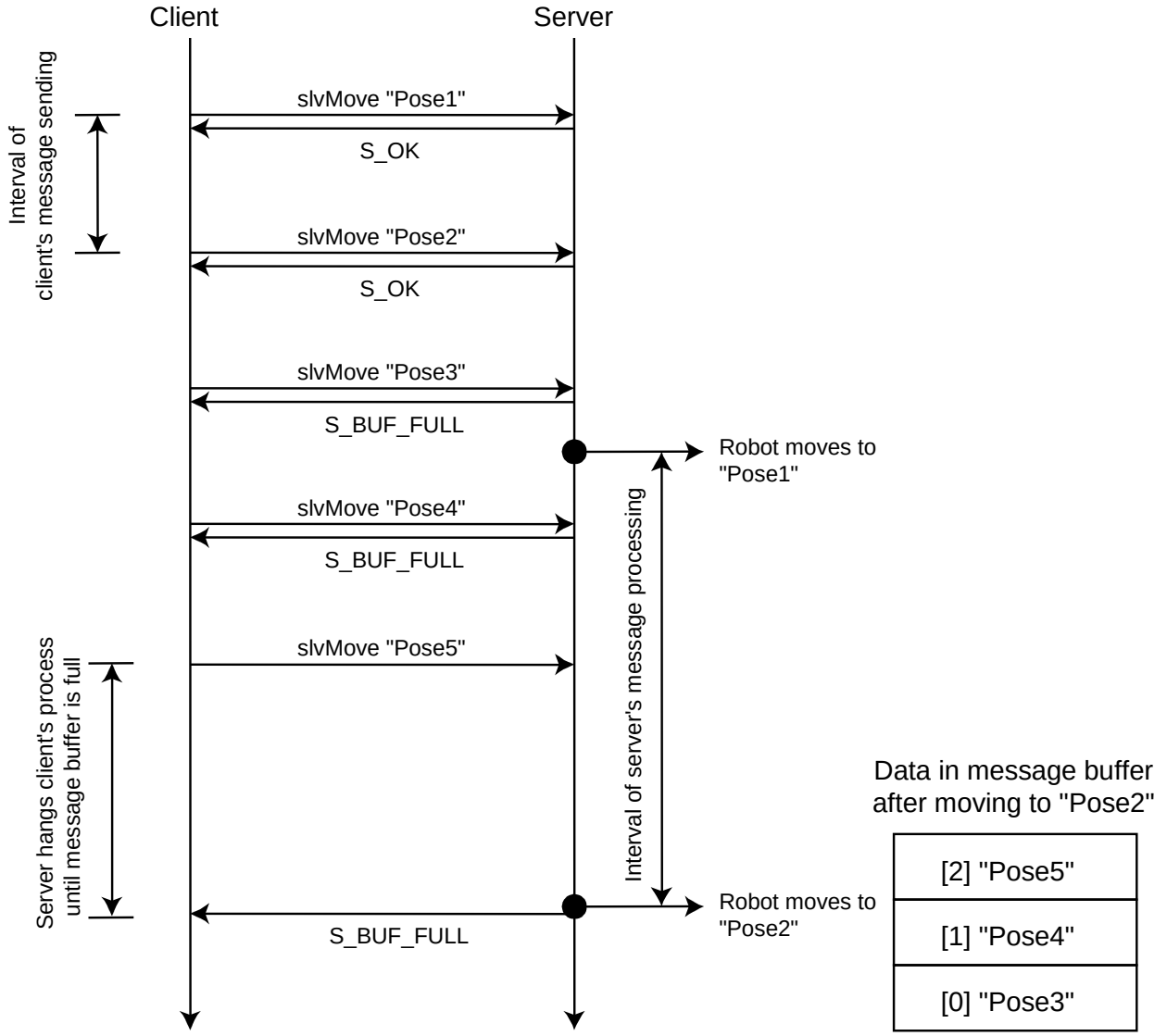


Figure 4-4 The communication procedure in the Mode2

4.4. Treatment of extended-joint

In the Slave Mode, you can control extended-joint by adding extended-joint option to the position information in the parameter of slvMove command. Table 4-5 shows the detail of the extended-joint option.

Table 4-5 Extended-joint option

Position type	Extended-joint	Position information
P Type	Axis 7 Only	VT_R8 | VT_ARRAY (9 Elements) (X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0x40, J7)
P Type	Axis 8 Only	VT_R8 | VT_ARRAY (9 Elements)

		(X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0x80, J8)
P Type	Axis 7 and 8	VT_R8 | VT_ARRAY (10 Elements) (X, Y, Z, RX, RY, RZ, Fig, 0x00400000 | 0xC0, J7, J8)
J Type	Axis 7 Only	VT_R8 | VT_ARRAY (9 Elements) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0x40)
J Type	Axis 8 Only	VT_R8 | VT_ARRAY (9 Elements) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0x80)
J Type	Axis 7 and 8	VT_R8 | VT_ARRAY (9 Elements) (J1, J2, J3, J4, J5, J6, J7, J8, 0x00400000 | 0xC0)
T Type	Axis 7 Only	VT_R8 | VT_ARRAY (12 Elements) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0x40, J7)
T Type	Axis 8 Only	VT_R8 | VT_ARRAY (12 Elements) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0x80, J8)
T Type	Axis 7 and 8	VT_R8 | VT_ARRAY (13 Elements) (X, Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, 0x00400000 | 0xC0, J7, J8)

4.5. Treatment of buffer underflow

As explained in "4.3 Summary of the Slave Mode ", Slave Mode stores position data and posture data which are sent from client in the buffer area, then creates the motion by reading out the buffer information in a certain period of time. If the buffer is empty (buffer underflow) when reading out the buffer information, the behavior of the server side differs depending on the running mode. Table 4-6 shows the behavior of the server side in buffer underflow state.

Table 4-6 Server behaviors in each mode under buffer underflow state

Slave mode	State of the robot	Behavior of the server	Note
Mode 0	Running state	Error is issued. (Error:0x84201482 = Position command buffer is empty)	SlaveMode is released
Mode 0	Stop state	Error is not issued.	Slave Mode is maintained.
Mode 1	Running state	Error is not issued.	Command to stay the current position is issued Slave Mode is maintained.
Mode 1	Stop state	Error is not issued.	Command to stay the current position is issued Slave Mode is maintained.

Mode 2	Running state	Error is issued. (Error:0x84201482= Position command buffer is empty)	Slave Mode is released
Mode 2	Stop state	Error is not issued	Slave Mode is maintained.

In this case, "Stop state" indicates that the speed of each robot axis is 0 m/s. Other status are deemed as "Running state".

Mode 0 or Mode 2 issues "Position command buffer is empty (0x84201482)" as an error message when the buffer becomes empty during running state. In order to stop the robot motion, you need to send the same command value for two or more consecutive times to set the command speed at 0 m/s. If this operation is executed when the robot speed is not sufficiently decreased, the robot suddenly stops then an error message of " J* command accel limit over (0x8420404*) " might be issued.

If the buffer is empty, Mode 1 issues a command to remain in a current position, regardless of the state of the robot. If the command to stay in the current position is issued while the robot speed is not enough decreased, the robot stops suddenly then an error message of " J* command accel limit over (0x8420404*)" might be issued.

4.6. Communication procedure of the Slave Mode

Figure 4-5 shows the communication flow of the Slave Mode. SlaveMode requires controller objects and robot objects. With regards to the procedures until acquiring handler of each objects and the procedures after the disconnection of each object, refer to "3.3 Robot control". Each step is described in more detail below.

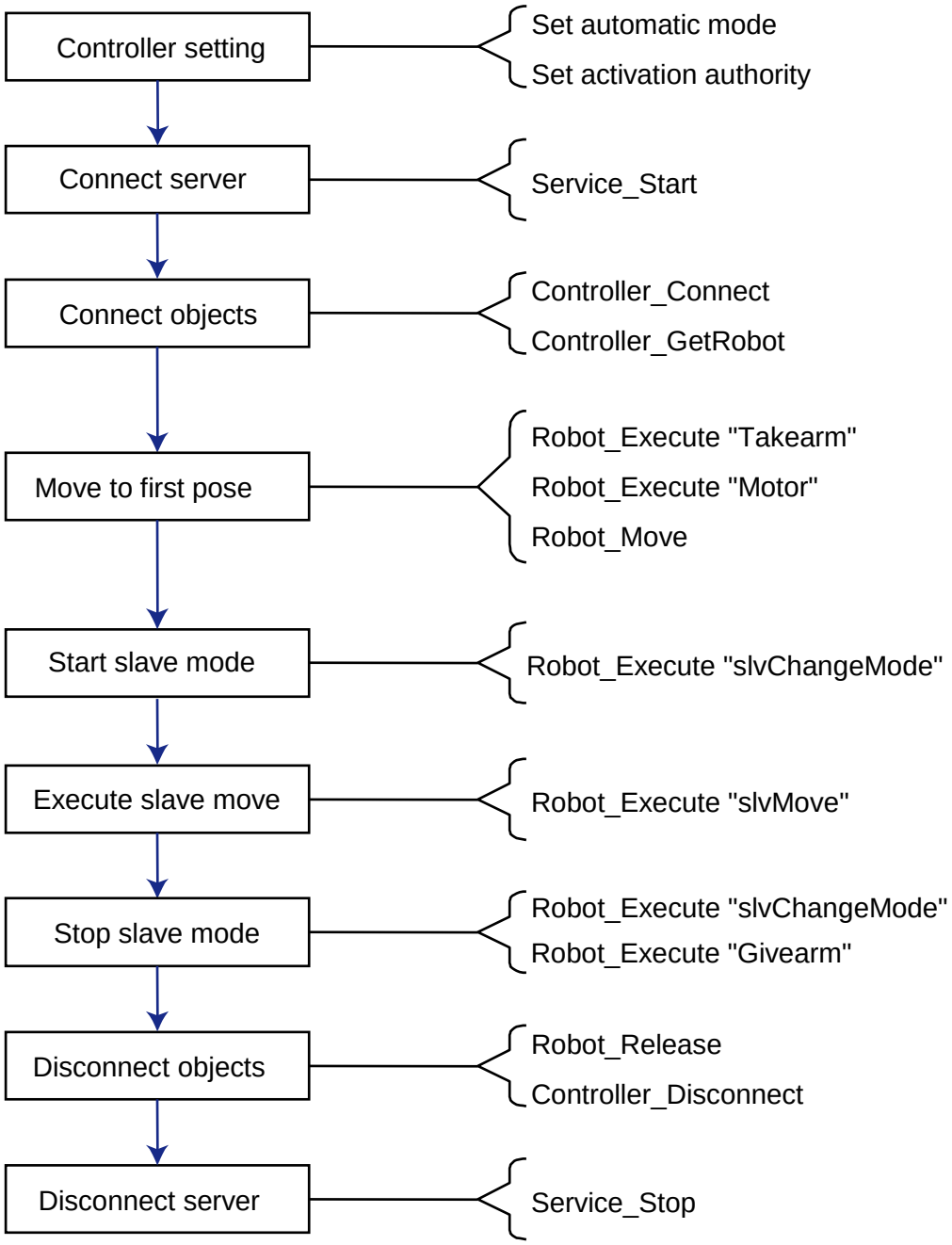


Figure 4-5 Flow of the Slave Mode

4.6.1. Moving to the initial position

Before running Slave Mode, the robot needs to be located in the initial coordinate and posture where the robot starts moving. For about detailed procedure to move the robot, refer to "3.3 Robot control".

When starting the Slave Mode, the robot has to be in stopped state. In order to achieve the designated initial position and posture completely, using "@E" as an option of "Robot_Move" command is recommended. The following example shows a packet to move a robot with "@E" option.

<table>
<tr><td colspan="4">Robot_Move 1, "@E P1", ""</td></tr>
<tr><td colspan="4">This executes the motion command "MOVE 1, "@E P1" """.</td></tr>
<tr><td rowspan="11">Packet
TX</td><td colspan="3">Client -> Server:
01 52 00 00 00 06 00 00 00 48 00 00 00 04 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 0A 00 00
00 03 00 01 00 00 00 01 00 00 00 14 00 00 00 08
00 01 00 00 00 0A 00 00 00 40 00 45 00 20 00 50
00 31 00 0A 00 00 00 08 00 01 00 00 00 00 00 00
00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td rowspan="2">hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td colspan="3">0A
00 00 00 03 00 01 00 00 00 03 00 00 00</td></tr>
<tr><td rowspan="2">lComp</td><td>Interpolation mode</td><td>VT_I4</td><td>0x00000001</td></tr>
<tr><td colspan="3">0A 00 00
00 03 00 01 00 00 00 01 00 00 00</td></tr>
<tr><td rowspan="2">vntPose</td><td>Posture</td><td>VT_BSTR</td><td>"@E P1"</td></tr>
<tr><td colspan="3">14 00 00 00 08
00 01 00 00 00 0A 00 00 00 40 00 45 00 20 00 50
00 31 00</td></tr>
<tr><td rowspan="2">bstrOption</td><td>Motion option</td><td>VT_BSTR</td><td>""</td></tr>
<tr><td colspan="3">0A 00 00 00 08 00 01 00 00 00 00 00 00
00</td></tr>
<tr><td rowspan="6">Packet
RX</td><td colspan="3">Server -> Client:
01 10 00 00 00 06 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td rowspan="2">No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

4.6.2. Starting and stopping the Slave Mode

To start or stop the Slave Mode, use "Robot_Execute"slvChangeMode"". Before starting the Slave Mode, the client has to have an executable token of arm. For details instruction of how to acquire the executable token of arm, refer to "3.3 Robot control". If you send a packet to stop Slave Mode, the server returns the return code after processing all of message buffers. The following example shows a packet to start and stop the Slave Mode. Here, the Slave Mode starts with Mode 0.

Robot_Execute "slvChangeMode", 0x001
This function starts the Slave Mode in the Mode0.

<table>
<tr><td>Packet
TX</td><td colspan="4">Client -> Server:
01 54 00 00 00 08 00 00 00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 24 00 00
00 08 00 01 00 00 00 1A 00 00 00 73 00 6C 00 76
00 43 00 68 00 61 00 6E 00 67 00 65 00 4D 00 6F
00 64 00 65 00 0A 00 00 00 03 00 01 00 00 00 01
00 00 00 04</td></tr>
<tr><td rowspan="8"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
<tr><td>bstrCommand</td><td>Command string</td><td>VT_BSTR</td><td>"slvChangeMode"</td></tr>
<tr><td></td><td colspan="3">00 08 00 01 00 00 00 1A 00 00 00 73 00 6C 00 76
00 43 00 68 00 61 00 6E 00 67 00 65 00 4D 00 6F
00 64 00 65 00 24 00 00</td></tr>
<tr><td>vntParam</td><td>Parameter</td><td>VT_I4</td><td>0x001</td></tr>
<tr><td></td><td colspan="3">00 00 00 0A 00 00 00 03 00 01 00 00 00 01</td></tr>
<tr><td>Packet
RX</td><td colspan="4">Server -> Client:
01 1A 00 00 00 08 00 00 00 00 00 00 00 01 00 06
00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="6"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>vntReturn</td><td>Return Value</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td></td><td colspan="3">00 00 00 00 00 01 00 00 00 06</td></tr>
</table>

<table>
<tr><td colspan="5">Robot_Execute "slvChangeMode", 0x000</td></tr>
<tr><td colspan="5">Exit the Slave Mode.</td></tr>
<tr><td>Packet
TX</td><td colspan="4">Client -> Server:
01 54 00 00 00 0A 00 00 00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 03 00 00 00 24 00 00
00 08 00 01 00 00 00 1A 00 00 00 73 00 6C 00 76
00 43 00 68 00 61 00 6E 00 67 00 65 00 4D 00 6F
00 64 00 65 00 0A 00 00 00 03 00 01 00 00 00 00
00 00 00 04</td></tr>
<tr><td rowspan="8"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hRobot</td><td>Robot handle</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 03 00 00 00 0A</td></tr>
<tr><td>bstrCommand</td><td>Command string</td><td>VT_BSTR</td><td>"slvChangeMode"</td></tr>
<tr><td></td><td colspan="3">00 08 00 01 00 00 00 1A 00 00 00 73 00 6C 00 76
00 43 00 68 00 61 00 6E 00 67 00 65 00 4D 00 6F
00 64 00 65 00 24 00 00</td></tr>
<tr><td>vntParam</td><td>Parameter</td><td>VT_I4</td><td>0x000</td></tr>
</table>

		0A 00 00 00 03 00 01 00 00 00 00 00 00 00		
Packet RX	Server -> Client: 01 1A 00 00 00 0A 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

4.6.3. Slave Move

To move the robot with Slave Mode, use "Robot_Execute"slvMove"". The following table shows the example of Slave Mode packet. In this example, a packet of P type variable coordinate data is transmitted. To execute the robot motion with Slave Mode, follow the procedure described in "4.3 Summary of the Slave Mode".

Robot_Execute "slvMove"				
This function sends the destination position by the slvMove command.				
Packet TX	Client -> Server: 01 7C 00 00 00 09 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 03 00 00 00 18 00 00 00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 00 4D 00 6F 00 76 00 65 00 3E 00 00 00 05 20 07 00 00 00 C3 F5 28 5C 8F C2 76 40 00 00 00 00 00 00 00 00 21 B0 72 68 91 68 71 40 00 00 00 00 00 80 66 40 80 8A 86 4A DC A5 0C 3D 00 00 00 00 00 80 66 40 00 00 00 00 00 00 14 40 04			
	Argument	Description	Date Type	Value
		Binary		
	hRobot	Robot handle	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	Command string	VT_BSTR	"slvMove"
		18 00 00 00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 00 4D 00 6F 00 76 00 65 00		
	vntParam	Parameter	VT_R8 | VT_ARRAY	364.16, 0, 278.5355, 180, 1.272222E-14, 180, 5
		3E 00 00 00 05 20 07 00 00 00 C3 F5 28 5C 8F C2 76 40 00 00 00 00 00 00 00 00 21 B0 72 68 91 68 71 40 00 00 00 00 00 80 66 40 80 8A 86 4A DC A5 0C 3D 00 00 00 00 00 80 66 40 00 00 00 00 00 00 14 40		

<table>
<tr><td>Packet
RX</td><td colspan="3">Server -> Client:
01 5A 00 00 00 09 00 00 00 00 00 00 00 01 00 46
00 00 00 05 20 08 00 00 00 C1 0B B2 0D EB 4B D8
BC F0 D1 25 37 00 80 46 40 0D 97 E5 F5 FF 7F 56
40 26 BC 9E 96 1A 58 06 3D F6 FF 0E DD FF 7F 46
40 96 B0 06 42 EC 4B D8 BC 0F 00 00 00 89 11 40
00 FE FF FF FF 10 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>vntReturn</td><td>Return Value</td><td>VT_R8 | VT_ARRAY</td><td>-1.34873E-15, 45.0, 90, 9.922799E-15, 45, -1.348731E-15, 0, 0</td></tr>
<tr><td colspan="4">00 00 00 05 20 08 00 00 00 C1 0B B2 0D EB 4B D8 46
BC F0 D1 25 37 00 80 46 40 0D 97 E5 F5 FF 7F 56
40 26 BC 9E 96 1A 58 06 3D F6 FF 0E DD FF 7F 46
40 96 B0 06 42 EC 4B D8 BC 0F 00 00 00 89 11 40
00 FE FF FF FF 10 00 00 00</td></tr>
</table>

4.7. Handling the error

When an error occurs during Slave Mode execution, Slave Mode is released and the error message appears on the teach pendant screen. In order to continue the Slave Mode after an error occurs, the client has to mount the error recovery process. Necessary items for recovery process of the client are 1) Clear the error on the teach pendant, and, 2) Starting the Slave Mode.

4.7.1. Clearing the error of the RC8

The Slave Mode cannot be resumed while the error message occurs in RC8 controller. There are two ways to clear the error of the controller, by clearing the error manually with a TeachPendant, and by sending a packet with b-CAP. In b-CAP, error clear is implemented as a command of "Controller_Execute(17)". The following example shows a packet to clear an error.

<table>
<tr><th colspan="5">Controller_Execute "ClearError"</th></tr>
<tr><td colspan="5">This function clears the error of the RC8.</td></tr>
<tr><td>Packet
TX</td><td colspan="4">Client -> Server:
01 4A 00 00 00 12 00 00 00 11 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 02 00 00 00 1E 00 00
00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65
00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00 06
00 00 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td></td><td colspan="3">Binary</td></tr>
<tr><td>hController</td><td>Controller handle</td><td>VT_I4</td><td>0x0000002</td></tr>
</table>

		00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrCommand	Command string	VT_BSTR	"ClearError"
		1E 00 00 00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00		
	vntParam	Parameter	VT_EMPTY	
		06 00 00 00 00 00 01 00 00 00		
Packet RX	Server -> Client: 01 1A 00 00 00 12 00 00 00 00 00 00 00 01 00 06 00 00 00 00 00 01 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_EMPTY	-
		06 00 00 00 00 00 01 00 00 00		

4.7.2. Restarting the Slave mode

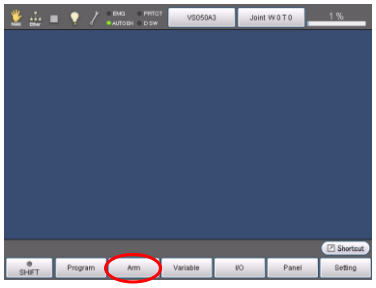
The Slave Mode is released once an error occurs; therefore, you must restart the Slave Mode. For details instruction of how to restart the Slave Mode, refer to "4.6 Communication procedure of the Slave Mode".

4.8. Setting of the command speed limit and acceleration limit

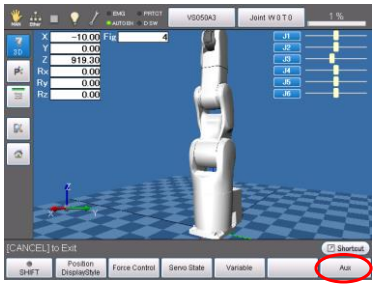
In the Slave Mode, you can set the command speed limit and acceleration limit. The limit is the threashold of the command speed and acceleration to reach the posture specified by Slave Move. If the command speed or acceleration exceeds the limit, "J* command speed limit over (0x8420405*)" or " J* command accel limit over (0x8420404*) " are issued.

To set the command speed limit and acceleration limit, use a teach pendant. From the top screen, press [F2 Arm] -> [F6 Aux] -> [F1 Config] -> [153: Speed setting for b-CAP Slave].

You can check the values that can be set to the command speed limit and acceleration limit in [0: Servo Limit], [1: Servo Limit (ExtSpeed)], [2: Command Limit] or [3: Command Limit (ExtSpeed)]. The servo limit is larger than the command limit, and if you specify the "(ExtSpeed)", the limit value will be the multiplication result of the each axis limit by the rate of external speed (acceleration).



[F2] ⇒



[F6] ⇒

[F1] ⇒

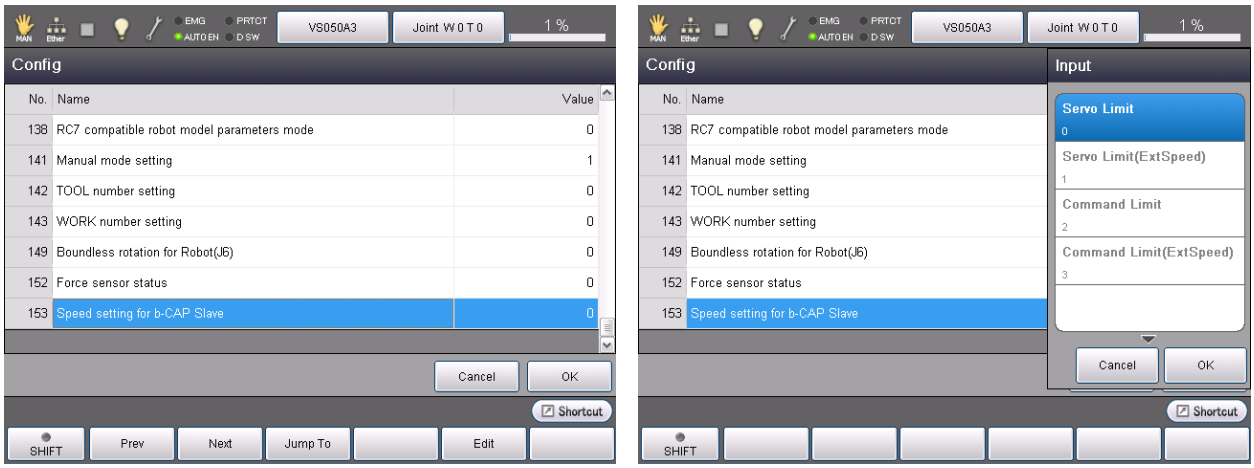


Figure 4-6 Speed setting for b-CAP Slave

4.9. Sample programs

The following is a sample program, which executes Slave Mode, with using ANSI-C sample library.

This sample program executes the Slave Mode with Mode 0, then moves the robot one cycle of the sine curve from the initial position. Because J1 coordinates is used as an initial position data, you need to set the robot coordinates in J1 beforehand. For IP, use values which was set to each controller. In this sample program, the following setting values are used.

IP:192.168.0.1

List 4-1 bCapSlvMode.c

```
#if _MSC_VER > 1500
#include <stdint.h>
#else
#include "stdint.h"
#endif
#include <stdio.h>

#ifdef _USE_WIN_API
/*ifndef _USE_LINUX_API*/
#include <Windows.h>
#endif

#define _USE_MATH_DEFINES
#include <math.h>

#include "bcap_client.h"

#define PERIOD              (100)     /* Period Cycle */
#define AMPLITUDE           (10)      /* Amplitude */
#define TIMES               (3)

#define E_BUF_FULL          (0x83201483)

int main(void)
{
```

```
int i, j, fd = 0;
long *plData;
uint32_t hCtrl = 0, hRob = 0;
double *pdData, dAng[8];
BSTR bstr1, bstr2, bstr3, bstr4;
VARIANT vntParam, vntRet;
HRESULT hr=E_FAIL, hrOpen=E_FAIL, hrCtrl=E_FAIL, hrRob=E_FAIL;

/* Open connection */
hrOpen = bCap_Open_Client("tcp:192.168.0.1:5007", 1000, 3, &fd);
if (FAILED(hrOpen)) goto END_PROC;

bstr1 = SysAllocString(L",WDT=400");

/* Send SERVICE_START packet */
bCap_ServiceStart(fd, bstr1);

SysFreeString(bstr1);

bstr1 = SysAllocString(L"b-CAP");                                    // Name
bstr2 = SysAllocString(L"CaoProv.DENSO.VRC");  // Provider
bstr3 = SysAllocString(L"localhost");                    // Machine
bstr4 = SysAllocString(L"");                                                  // Option

/* Connect robot controller */
hrCtrl = bCap_ControllerConnect(fd, bstr1, bstr2, bstr3, bstr4, &hCtrl);

SysFreeString(bstr1);
SysFreeString(bstr2);
SysFreeString(bstr3);
SysFreeString(bstr4);

if (FAILED(hrCtrl)) goto END_PROC;

VariantInit(&vntParam);
bstr1 = SysAllocString(L"ClearError");

/* Clear Error */
hr = bCap_ControllerExecute(fd, hCtrl, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"Robot");     // Name
bstr2 = SysAllocString(L"");                    // Option

/* Get robot handle */
hrRob = bCap_ControllerGetRobot(fd, hCtrl, bstr1, bstr2, &hRob);

SysFreeString(bstr1);
SysFreeString(bstr2);

if (FAILED(hrRob)) goto END_PROC;

bstr1 = SysAllocString(L"Takearm");

VariantInit(&vntParam);
vntParam.vt = (VT_I4 | VT_ARRAY);
vntParam.parray = SafeArrayCreateVector(VT_I4, 0, 2);
SafeArrayAccessData(vntParam.parray, (void **)&plData);
plData[0] = 0; plData[1] = 1;
SafeArrayUnaccessData(vntParam.parray);
```

```
/* Get arm control authority */
hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

/* EXTSPEED 20% */
bstr1 = SysAllocString(L"ExtSpeed");

VariantInit(&vntParam);
vntParam.vt = (VT_R8 | VT_ARRAY);
vntParam.parray = SafeArrayCreateVector(VT_R8, 0, 1);
SafeArrayAccessData(vntParam.parray, (void **)&pdData);
pdData[0] = 20.0; // 20.0%
SafeArrayUnaccessData(vntParam.parray);

printf("External Speed = 20%%¥n");
hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"Motor");

VariantInit(&vntParam);
vntParam.vt = VT_I4;
vntParam.lVal = 1;

/* Motor on */
bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SysFreeString(bstr1);
VariantClear(&vntParam);
VariantClear(&vntRet);

VariantInit(&vntParam);
vntParam.vt = VT_BSTR;
//vntParam.bstrVal = SysAllocString(L"@E J1");
vntParam.bstrVal = SysAllocString(L"@E J(45, 30, 120, 0, -60, 0)");

/* Move to first pose */
hr = bCap_RobotMove(fd, hRob, 1, vntParam, NULL);

VariantClear(&vntParam);

if (FAILED(hr)) goto END_PROC;

bstr1 = SysAllocString(L"CurJnt");

VariantInit(&vntParam);
vntParam.vt = VT_EMPTY;

/* Get current angle */
bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

SafeArrayAccessData(vntRet.parray, (void **)&pdData);
memcpy(dAng, pdData, 8 * sizeof(double));
SafeArrayUnaccessData(vntRet.parray);

SysFreeString(bstr1);
```

```
	VariantClear(&vntParam);
	VariantClear(&vntRet);

	bstr1 = SysAllocString(L"slvChangeMode");

	VariantInit(&vntParam);
	vntParam.vt = VT_I4;
	vntParam.lVal = 2;

	/* Start slave mode (Mode 0, J Type) */
	hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

	SysFreeString(bstr1);
	VariantClear(&vntParam);
	VariantClear(&vntRet);

	if (FAILED(hr)) goto END_PROC;

	bstr1 = SysAllocString(L"slvMove");

	VariantInit(&vntParam);
	vntParam.vt = (VT_R8 | VT_ARRAY);
	vntParam.parray = SafeArrayCreateVector(VT_R8, 0, 8);

	for(j = 0; j < TIMES; j++) {
		for(i = 0; i < PERIOD + 1; i++){
			SafeArrayAccessData(vntParam.parray, (void **)&pdData);
			memcpy(pdData, dAng, 8 * sizeof(double));
			pdData[0] = dAng[0] - AMPLITUDE * cos(2 * M_PI*i / PERIOD) + AMPLITUDE;
			pdData[1] = dAng[1] - AMPLITUDE * cos(2 * M_PI*i / PERIOD) + AMPLITUDE;
			SafeArrayUnaccessData(vntParam.parray);

			hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);
			VariantClear(&vntRet);

			/* if return code is not S_OK, then keep the message sending process
waiting for 8 msec */
			if(hr != S_OK){
				Sleep(8);

				/* if return code is E_BUF_FULL, then retry previous packet
*/
				if(FAILED(hr)){
					if(hr == E_BUF_FULL){
						i--;
					}else{
						j = TIMES;
						break;
					}
				}
			}
		}
	}

	/* Stop robot */
	hr = bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

	SysFreeString(bstr1);
	VariantClear(&vntParam);
	VariantClear(&vntRet);

	bstr1 = SysAllocString(L"slvChangeMode");

	VariantInit(&vntParam);
	vntParam.vt = VT_I4;
	vntParam.lVal = 0;
```

```
        /* Stop slave mode */
        bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);
        VariantClear(&vntParam);
        VariantClear(&vntRet);

        bstr1 = SysAllocString(L"Motor");

        VariantInit(&vntParam);
        vntParam.vt = VT_I4;
        vntParam.lVal = 0;

        /* Motor off */
        bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

        SysFreeString(bstr1);
        VariantClear(&vntParam);
        VariantClear(&vntRet);

END_PROC:
        if (SUCCEEDED(hrRob)) {
                bstr1 = SysAllocString(L"Givearm");

                VariantInit(&vntParam);
                vntParam.vt = VT_EMPTY;

                /* Release arm control authority */
                bCap_RobotExecute(fd, hRob, bstr1, vntParam, &vntRet);

                SysFreeString(bstr1);
                VariantClear(&vntParam);
                VariantClear(&vntRet);

                /* Release robot handle */
                bCap_RobotRelease(fd, &hRob);
        }
        if (SUCCEEDED(hrCtrl)) {
                /* Disconnect robot controller */
                bCap_ControllerDisconnect(fd, &hCtrl);
        }
        if (SUCCEEDED(hrOpen)) {
                /* Send SERVICE_STOP packet */
                bCap_ServiceStop(fd);

                /* Close connection */
                bCap_Close_Client(&fd);
        }

        return 0;
};
```

4.10. Unsupported functions

The below functions are unsupported with b-CAP Slave

・Move with two or more robot.

・Collision prevention with the Virtual Fence.

5. b-CAP Tester

b-CAP Tester attached in ORiN2 SDK enables you to confirm packets sent and received from the controller.

b-CAP Tester (b-CAPTester_RC8.exe) is stored in the following folder.

ORiN2\CAP\b-CAP\CapLib\DENSO\RC8\Bin

Figure 5-1 describes the functions of b-CAP Tester.

Set the parameters described in Table 5-1 to connect to the controller.

Table 5-1 RC8 connection parameters

Option	Description
Server=<IP address>	Specify IP address of the target controller.
Provider =<Provider name>	For connecting RC8, specify "CaoProv.DENSO.VRC."
Machine=<machine name>	For connecting RC8, specify the same value as Server.
Option[=<option character string>]	Specify the option character string required for a remote provider. (default value: Null character string)
Message[=<True/False>]	Status of message acquisition. True: Valid the message acquisition (default). False: Invalid the message acquisition.
UDP[=<True/False>]	Network transmission setting by UDP True:UDP False:TCP (default) The maximum size of the packet becomes 488 bytes at the UDP communication.
Timeout=< time-out time >	Time-out time when sending and receiving. (default: 500 ms)
TORetry=<.Retry frequency>	Retry frequency when UDP is sent and received. 1-7 (Default: 5) Less than one is regarded as one. More than seven is regarded as seven. The time-out response time of UDP is calculated by the following formula Time-out response time = <Timeout>×<TORetry>
Debug[=<True/False>]	Specification of debug mode True: Debug mode

	False: Normal mode The following variables can be used at debug mode. $LAST_SEND_PACKET$ $LAST_RECEIVE_PACKET$

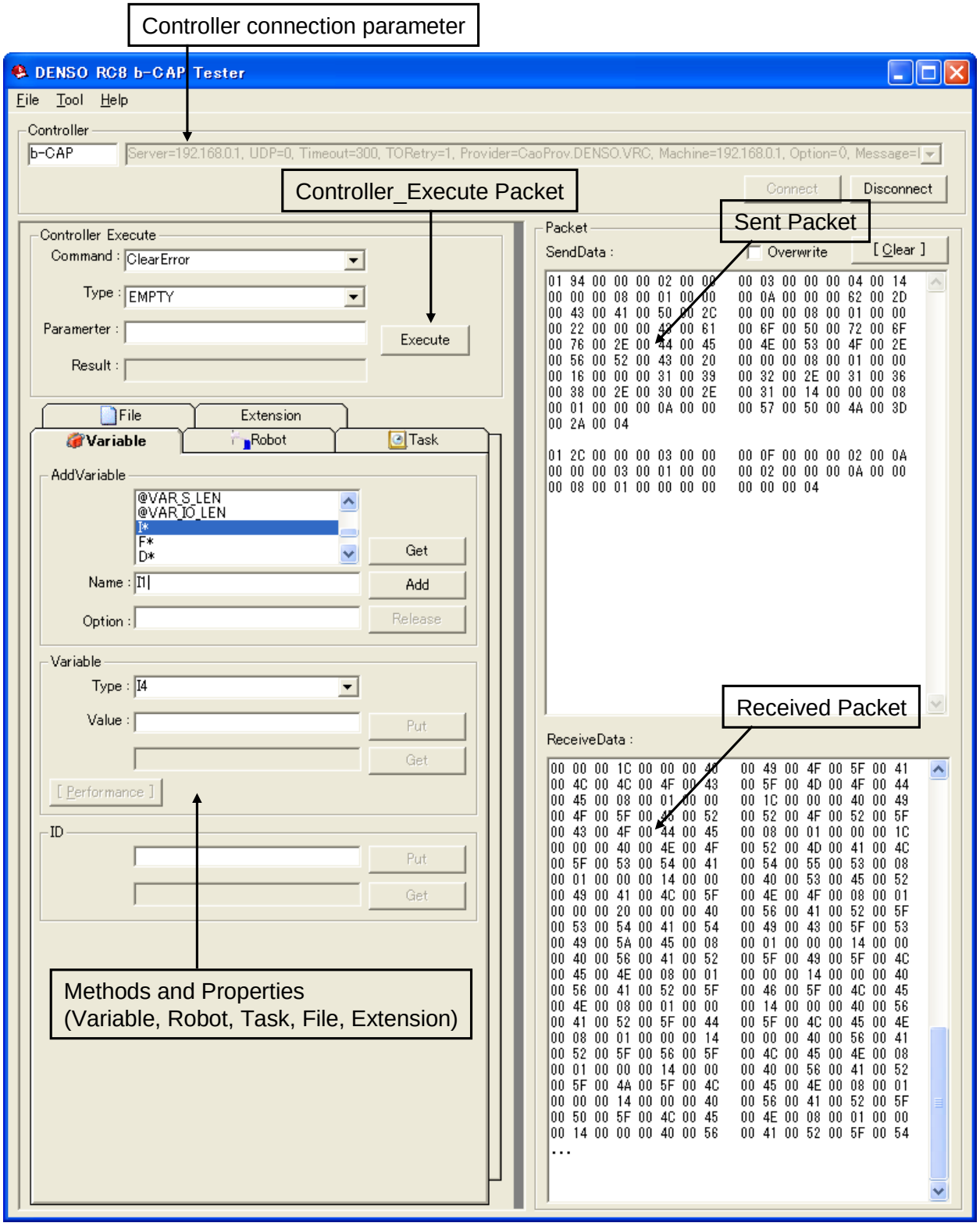


Figure 5-1 Description of functions of the b-CAP Tester

5.1. Slave Mode in b-CAP Tester

Before moving a robot at Slave Mode with b-CAP Tester, do the following preparations.

- WINCAPS3 project files that obtains the control logs.

  "slvMove" of b-CAP Tester initiates the robot motion by using command value of the control log.

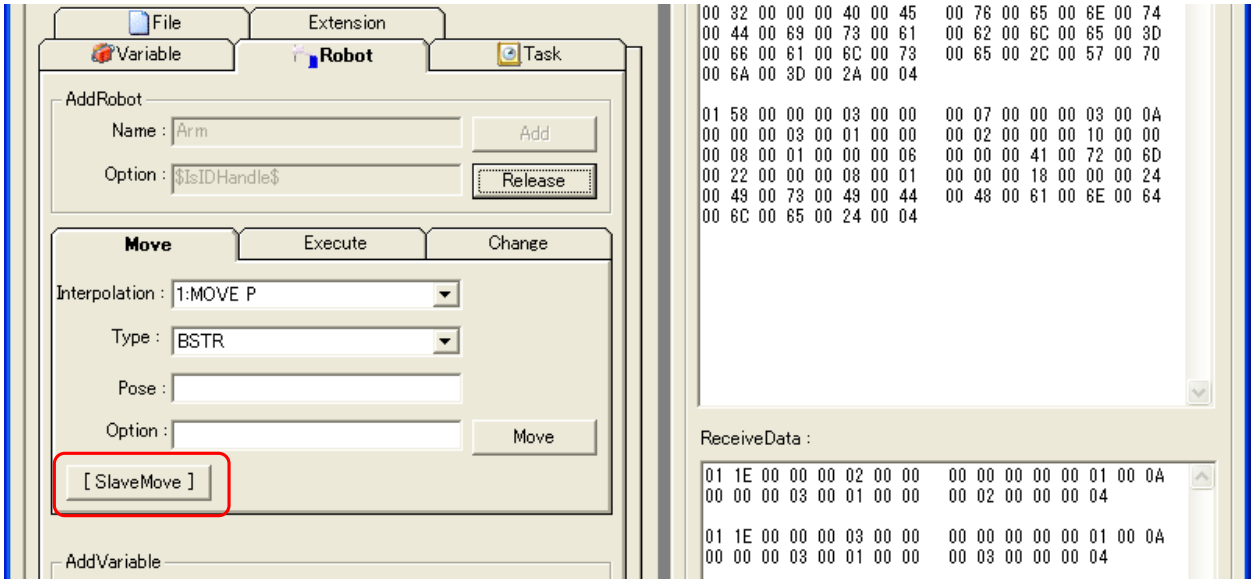
- Preparation of the controller

  Set the controller in Auto mode. Set the Executable token of the controller in the IP of the client.

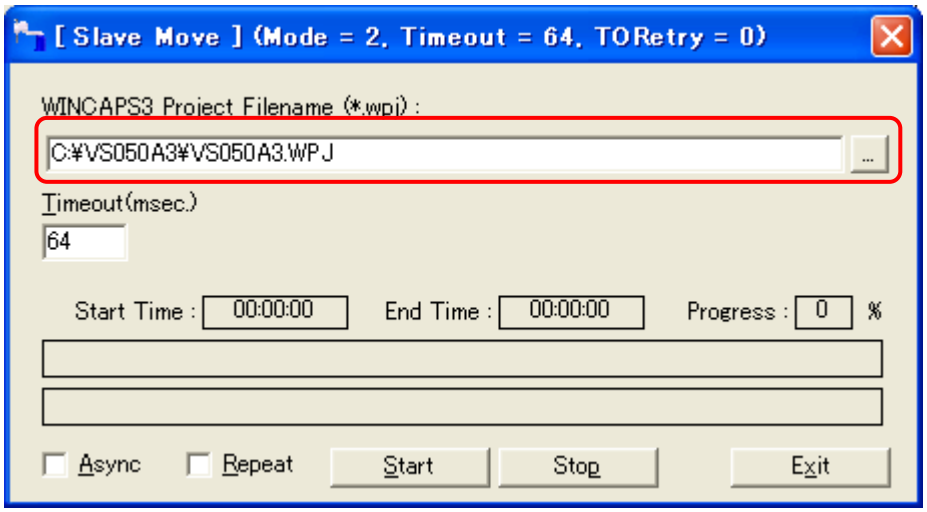
  For details, refer to "2.Setup of RC8".

5.1.1. How to test the Slave Mode by the b-CAP Tester

1. Once connected to the robot object, press [Slave Move] button to display the Slave Move window..



2. Specify WINCAP3 project which stores the control log.



3. Once Start button is pressed, the robot starts moving.

   In this process, following statements are executed.

   3.1 Taking the arm control authority

   3.2 Moving to the initial position

   3.3 Starting Slave Mode

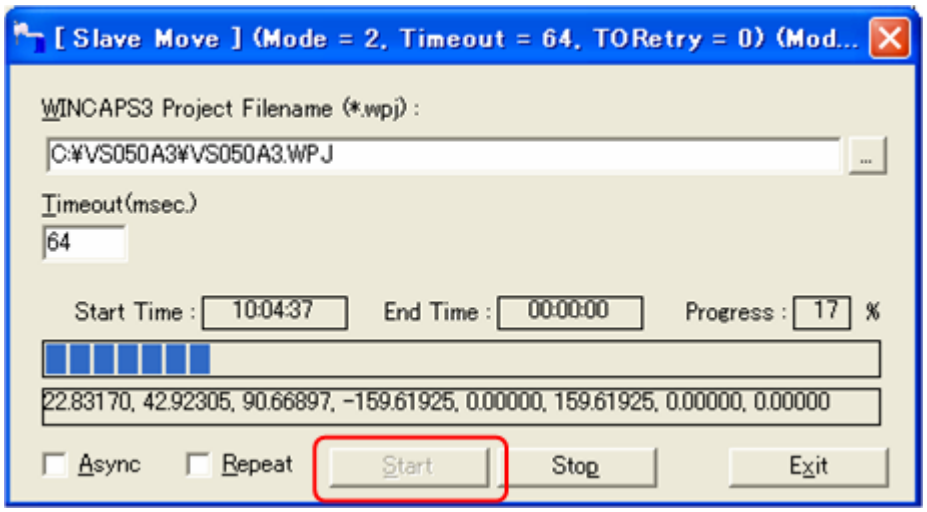
   3.4 Executing Slave Move

   3.5 Stopping Slave Mode

   3.6 Releasing the arm control authority

   In this process, a motor is not controlled automatically. Please send a control packet beforehand so that Slave Mode can be executed.

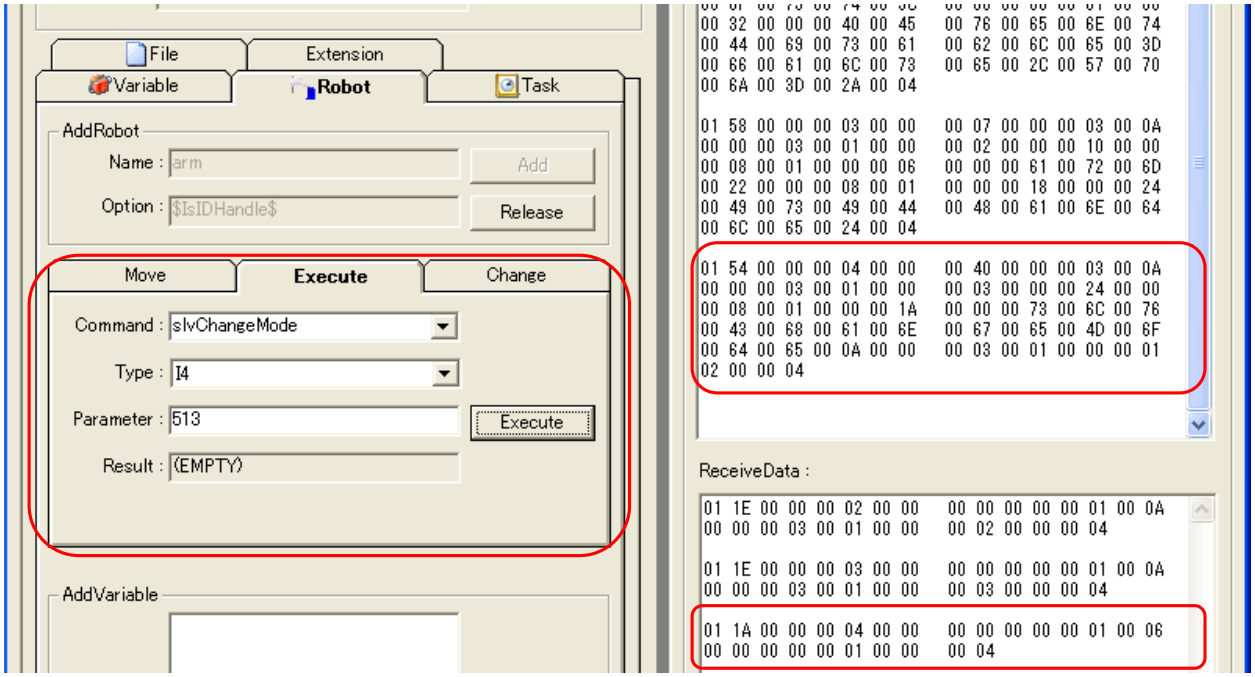
   If the time-out occurs while moving to initial position, please set the time-out parameter with larger value when connecting the controller.



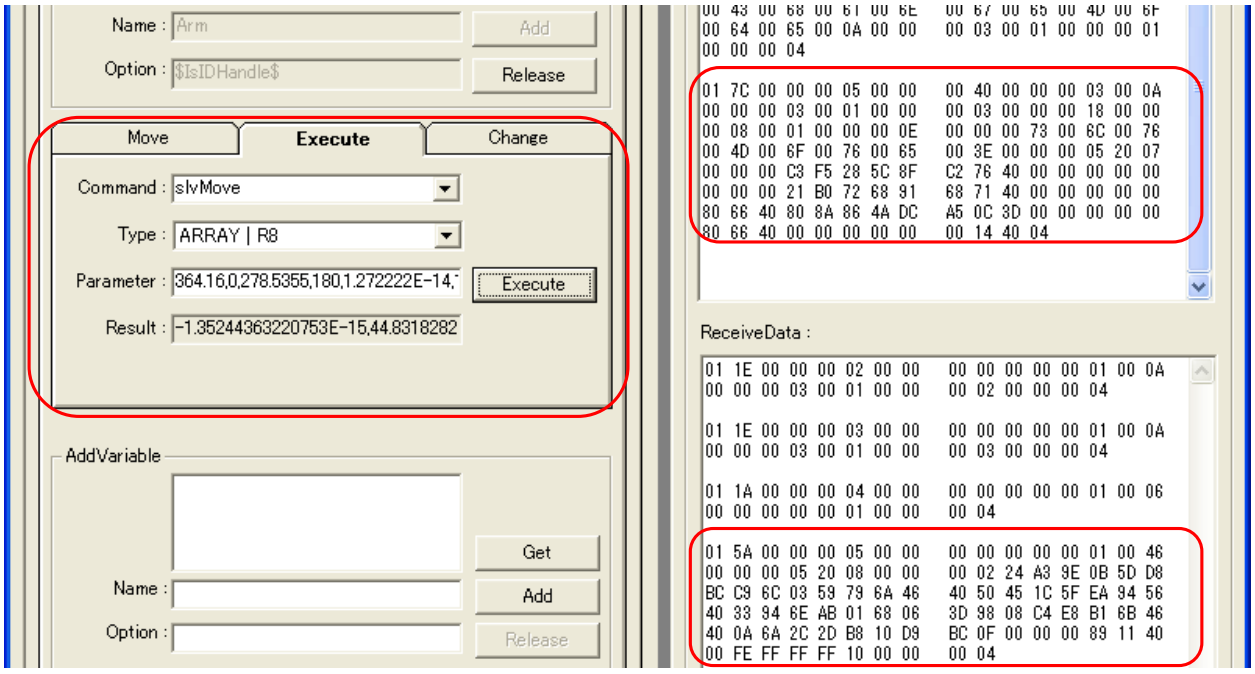
5.1.2. Packet confirmation of the Slave move in the b-CAP Tester

When a robot is moved in the Slave Move window, a packet to be sent or received is not displayed. To confirm the packet, issue a command from Execute command of the robot.

1. Change the mode by "slvChangeMode" in Execute tab.

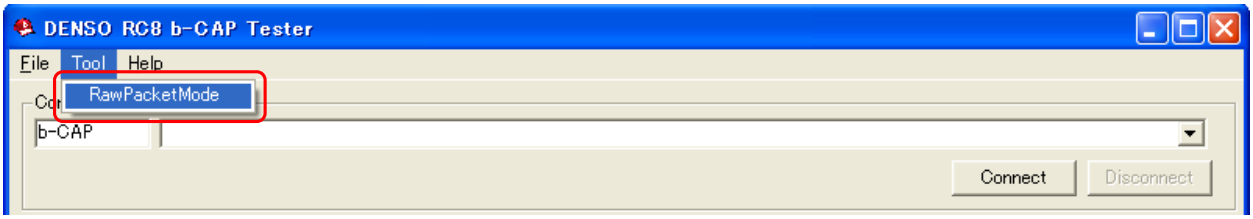


2. Execute "slvMove" command to move.



5.2. About Raw Packet Mode

In Raw Packet Mode, you can control a controller by sending packets which are manually created. This section describes how to use Raw Packet Mode.



5.2.1. Connecting to the controller

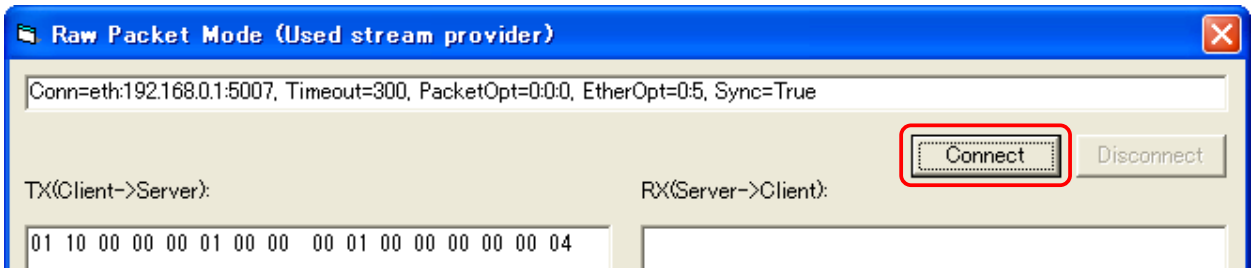
To connect to the controller in Raw Packet Mode, set the parameters described in Table 5-2 and click [Connect] button. For details each of the parameters, refer to "Stream Provider Guide."

ORiN2\CAO\ProviderLib\DENSO\Stream\Doc\ Stream_ProvGuide_en.pdf

Table 5-2 Connection parameters of Raw Paket Mode

Option	Description
Conn=eth:[<IP Address>[:<Port No>]]	Specify the IP address of controller to be connected
Timeout [=<Timeout>]	Timeout period at data sending and receiving. (default: 500)
PacketOpt =[<Mode>[:<Header>[:<Term>]]]	<Mode>: Communication data conversion. The first bit: ISO conversion The second bit: EIA conversion The third bit: Unicode conversion The fourth bit: Text mode The fifth bit: RoboTalk mode The sixth bit: b-CAP mode <Header>: Header specification. ‘0' – none, ‘1' - ENQ(0x05) <Term>: Terminator specification. ‘0'-CR(0x0D), ‘1'-LF(0x0A), ‘2'-CR+LF(0x0D0A) To enter b-CAP packets directly, specify 0:0:0.
EtherOpt =[<Mode>[:<ConnMax>]]	<Mode>: Connection mode ‘0' . -TCP client mode ‘1'-TCP server mode

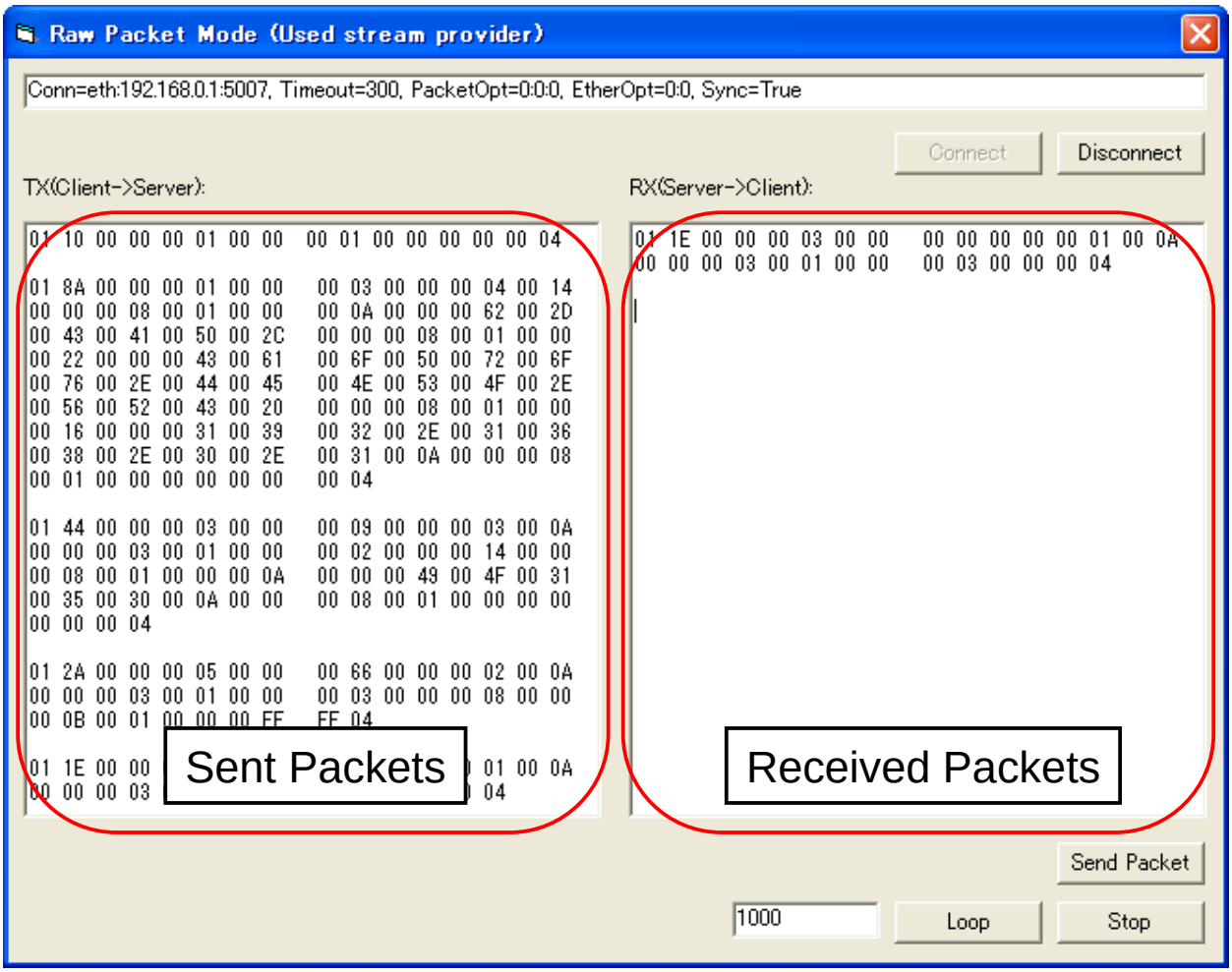
	'2' . -UDP client mode '3'-UDP server mode <ConnMax>: Number of maximum clients at TCP server mode. (default: 5) In Raw Packet Mode, specify 0:0 or 2:0.
Sync=TRUE	The synchronous mode is set. In Raw Packet Mode, specify TRUE.



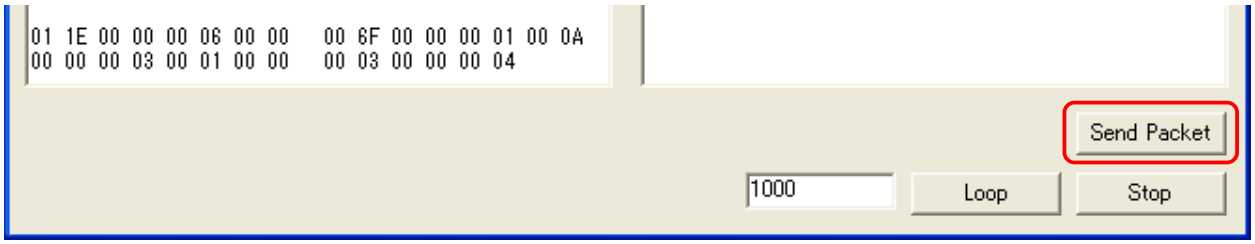
5.2.2. Sending and Receiving b-CAP packets

For sending b-CAP packet in Raw Packet Mode, you need to write packets in the left side of the text area. You can write two or more packets at one time.

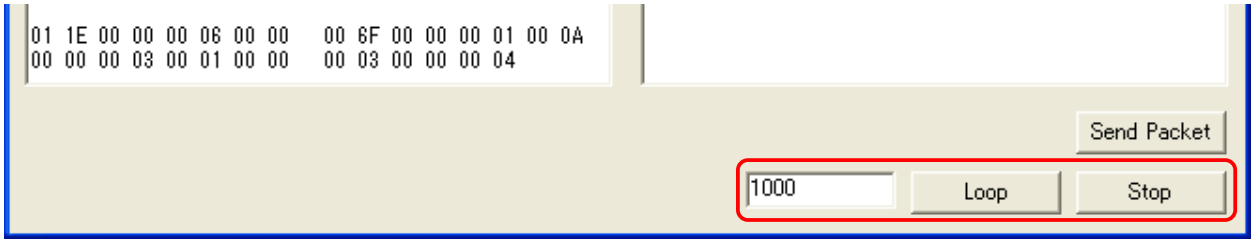
Received packets are shown in the right side of the text area. If you write and send two or more packets, the displayed packet in the right side of the text area will be the reply of the last packet.



Once the packets to send has been written, click [Send Packet] button to send the packets to the controller.



If you want to send the written packet several times, specify the count of sendings, and click [Loop] button. To cancel the sendings, click [Stop] button.



5.3. Description of VT_ARRAY | VT_VARIANT in b-CAP Tester

In b-CAP Tester, when transmitting parameters of "VT_ARRAY|VT_VARIANT", use a format shown below.

<Data type>, <Data column>

<Data type> in this format is integer value written in VARTYPE. Table 5-3 describes available data types and values.

Table 5-3 Available data types

Data type	Value	Description
VT_I2	2	Short integer
VT_I4	3	Long integer
VT_R4	4	Single-precision floating point
VT_R8	5	Double-precision floating point
VT_CY	6	Currency type
VT_DATE	7	Date type
VT_BSTR	8	String type
VT_BOOL	11	Boolean type
VT_VARIANT	12	VARIANT type
VT_UI1	17	Binary
VT_ARRAY	8192	Array

If the data type is array, specify the logical sum of VT_ARRAY and the data type.

For Data columns, specify data by the character strings. The description of the array data is delimited by "," (comma).

Figure 5-2 shows a sample of a Pose of "Robot_Move".

In "Robot_Move", VT_ARRAY | VT_VARIANT can be designated as a Pose.

For the first array, "VT_R8 | VT_ARRAY (8197)" is used in order to specify the coordinate and posture data (8197, 0, 0, 0, 0, 0, 0, 5).

In the second array, "VT_I4 (3)" is used in order to describe variable type (0).

In the third array, "VT_I4 (3)" is used in order to describe path (-2).

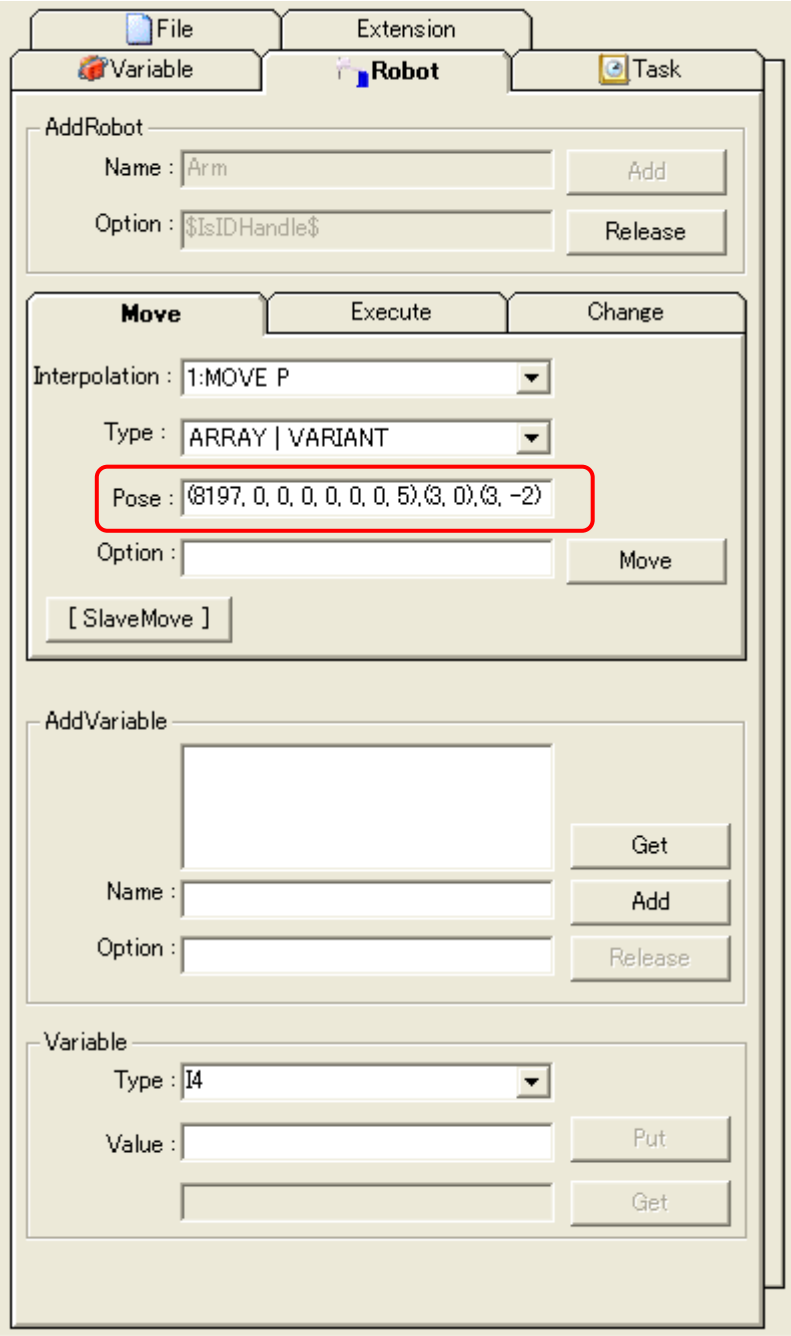


Figure 5-2 Example of the Pose in "Robot_Move"

Appendix A. Correspondence table about b-CAP function ID and CAO interface

Function ID	Function name	CAO Interface name	Explanation
1	Service_Start	CCaoWorkspace::AddController	Start the server service
2	Service_Stop	CCaoWorkspaces::Remove	Stop the server service
3	Controller_Connect		Connect with controller
4	Controller_Disconnect		Disconnect from controller
5	Controller_GetExtension	CCaoController::AddExtension	Acquire a controller's extension board
6	Controller_GetFile	CCaoController::AddFile	Acquire a controller's file
7	Controller_GetRobot	CCaoController::AddRobot	Acquire a controller's robot
8	Controller_GetTask	CCaoController::AddTask	Acquire a controller's task
9	Controller_GetVariable	CCaoController::AddVariable	Acquire a controller's variable
10	Controller_GetCommand	CCaoController::AddCommand	Acquire a controller's command
11	Controller_GetExtensionNames	CCaoController::get_ExtensionNames	Acquire a controller's extension board name list
12	Controller_GetFileNames	CCaoController::get_FileNames	Acquire a controller's file name list
13	Controller_GetRobotNames	CCaoController::get_RobotNames	Acquire a controller's robot name list
14	Controller_GetTaskNames	CCaoController::get_TaskNames	Acquire a controller's task name list
15	Controller_GetVariableNames	CCaoController::get_VariableNames	Acquire a controller's variable identifier list
16	Controller_GetCommandNames	CCaoController::get_CommandNames	Acquire a controller's command name list

17	Controller_Execute	CCaoController::Execute	Execute a controller's expansion function
18	Controller_GetMessage	CCaoController::AddMessage	Acquire a controller's event message
19	Controller_GetAttribute	CCaoController::get_Attribute	Acquire a controller's attribute value
20	Controller_GetHelp	CCaoController::get_Help	Acquire a controller's help character string
21	Controller_GetName	CCaoController::get_Name	Acquire a controller's name
22	Controller_GetTag	CCaoController::get_Tag	Acquire a controller's tag information
23	Controller_PutTag	CCaoController::put_Tag	Set the controller's tag information
24	Controller_GetID	CCaoController::get_ID	Acquire a controller's ID
25	Controller_PutID	CCaoController::put_ID	Set a controller's ID
26	Extension_GetVariable	CCaoExtension::AddVariable	Acquire a variable of extension board
27	Extension_GetVariableNames	CCaoExtension::get_VariableNames	Acquire the list of variable identifier of extension board
28	Extension_Execute	CCaoExtension::Execute	Execute an expansion function of extension board
29	Extension_GetAttribute	CCaoExtension::get_Attribute	Acquire an attribute value of extension board
30	Extension_GetHelp	CCaoExtension::get_Help	Acquire the help character string of extension board
31	Extension_GetName	CCaoExtension::get_Name	Acquire a name of extension board
32	Extension_GetTag	CCaoExtension::get_Tag	Acquire tag information on extension board
33	Extension_PutTag	CCaoExtension::put_Tag	Set tag information on

			extension board
34	Extension_GetID	CCaoExtension::get_ID	Acquire an extension board ID
35	Extension_PutID	CCaoExtension::put_ID	Set an extension board ID
36	Extension_Release	CCaoExtension::Release	Release an extension board
37	File_GetFile	CCaoFile::AddFile	Acquire other file names
38	File_GetVariable	CCaoFile::AddVariable	Acquire the variable of file
39	File_GetFileNames	CCaoFile::get_FileNames	Acquire the list of another file names
40	File_GetVariableNames	CCaoFile::get_VariableNames	Acquire the list of variable identifier of file
41	File_Execute	CCaoFile::Execute	Execute the expansion function of file
42	File_Copy	CCaoFile::Copy	Copy a file
43	File_Delete	CCaoFile::Delete	Delete a file
44	File_Move	CCaoFile::Move	Move a file
45	File_Run	CCaoFile::Run	Execute a file
46	File_GetDateCreated	CCaoFile::get_DateCreated	Acquire the date of file
47	File_GetDateLastAccessed	CCaoFile::get_DateLastAccessed	Acquire the final access date of file
48	File_GetDateLastModified	CCaoFile::get_DateLastModified	Acquire the last updated date and time of file
49	File_GetPath	CCaoFile::get_Path	Acquire the path of file
50	File_GetSize	CCaoFile::get_Size	Acquire the file size
51	File_GetType	CCaoFile::get_Type	Acquire the file type
52	File_GetValue	CCaoFile::get_Value	Acquire a content of file
53	File_PutValue	CCaoFile::put_Value	Set the file content
54	File_GetAttribute	CCaoFile::get_Attribute	Acquire the file attribute
55	File_GetHelp	CCaoFile::get_Help	Acquire the help character string of file

56	File_GetName	CCaoFile::get_Name	Acquire a name of file
57	File_GetTag	CCaoFile::get_Tag	Acquire a tag information on file
58	File_PutTag	CCaoFile::put_Tag	Set a tag information on file
59	File_GetID	CCaoFile::get_ID	Acquire a file ID
60	File_PutID	CCaoFile::put_ID	Set a file ID
61	File_Release	CCaoFile::Release	Release a file
62	Robot_GetVariable	CCaoRobot::AddVariable	Acquire the variable of robot
63	Robot_GetVariableNames	CCaoRobot::get_VariableNames	Acquire the list of variable identifier of robot
64	Robot_Execute	CCaoRobot::Execute	Execute the expansion function of robot
65	Robot_Accelerate	CCaoRobot::Accelerate	Execute the ACCEL sentence of robot
66	Robot_Change	CCaoRobot::Change	Execute the CHANGE sentence of robot
67	Robot_Chuck	CCaoRobot::Chuck	Execute the GRASP sentence of robot
68	Robot_Drive	CCaoRobot::Drive	Execute the DRIVE sentence of robot
69	Robot_GoHome	CCaoRobot::GoHome	Execute the GOHOME sentence of robot
70	Robot_Halt	CCaoRobot::Halt	Execute the HALT sentence of robot
71	Robot_Hold	CCaoRobot::Hold	Execute the HOLD sentence of robot
72	Robot_Move	CCaoRobot::Move	Execute the MOVE sentence of robot
73	Robot_Rotate	CCaoRobot::Rotate	Execute the ROTATE sentence of robot
74	Robot_Speed	CCaoRobot::Speed	Execute the SPEED/JSPEED sentence of robot

75	Robot_Unchuck	CCaoRobot::Unchuck	Execute the REELASE sentence of robot
76	Robot_Unhold	CCaoRobot::Unhold	Release of HOLD sentence of robot
77	Robot_GetAttribute	CCaoRobot::get_Attribute	Acquire the robot attribuve value
78	Robot_GetHelp	CCaoRobot::get_Help	Acquire the help character string of robot
79	Robot_GetName	CCaoRobot::get_Name	Acquire the name of robot
80	Robot_GetTag	CCaoRobot::get_Tag	Acquire the tag information on robot
81	Robot_PutTag	CCaoRobot::put_Tag	Set the tag information on robot
82	Robot_GetID	CCaoRobot::get_ID	Acquire a robot ID
83	Robot_PutID	CCaoRobot::put_ID	Set a robot ID
84	Robot_Release	CCaoRobot::Release	Release a robot ID
85	Task_GetVariable	CCaoTask::AddVariable	Acquire the variable of task
86	Task_GetVariableNames	CCaoTask::get_VariableNames	Acquire the list of variable identifier of task
87	Task_Execute	CCaoTask::Execute	Execute the expansion function of task
88	Task_Start	CCaoTask::Start	Start a task
89	Task_Stop	CCaoTask::Stop	Stop a task
90	Task_Delete	CCaoTask::Delete	Delete a task
91	Task_GetFileName	CCaoTask::get_FileName	Former file name of task
92	Task_GetAttribute	CCaoTask::get_Attribute	Aquire the task attribute
93	Task_GetHelp	CCaoTask::get_Help	Acquire the help character string of task
94	Task_GetName	CCaoTask::get_Name	Acquire the name of task
95	Task_GetTag	CCaoTask::get_Tag	Acquire the tag information on task

96	Task_PutTag	CCaoTask::put_Tag	Sett the tag information on task
97	Task_GetID	CCaoTask::get_ID	Acquire a task ID
98	Task_PutID	CCaoTask::put_ID	Set a task ID
99	Task_Release	CCaoTask::Release	Release a task ID
100	Variable_GetDateTime	CCaoVariable::get_DateTime	Acquire the timestamp of variable
101	Variable_GetValue	CCaoVariable::get_Value	Acquire a value of variable
102	Variable_PutValue	CCaoVariable::put_Value	Set a value of variable
103	Variable_GetAttribute	CCaoVariable::get_Attribute	Acquire the attribute value of variable
104	Variable_GetHelp	CCaoVariable::get_Help	Acquire the help character string of variable
105	Variable_GetName	CCaoVariable::get_Name	Acquire the name of variable
106	Variable_GetTag	CCaoVariable::get_Tag	Acquire the tag information on variable
107	Variable_PutTag	CCaoVariable::put_Tag	Set the tag information on variable
108	Variable_GetID	CCaoVariable::get_ID	Acquire a variable ID
109	Variable_PutID	CCaoVariable::put_ID	Set a variable ID
110	Variable_GetMicrosecond	CCaoVariable::get_Microsecond	Acquire the time-stamp of variable (millisecond)
111	Variable_Release	CCaoVariable::Release	Release a variable
112	Command_Execute	CCaoCommand::Execute	Execute a command
113	Command_Cancel	CCaoCommand::Cancel	Cancel a command
114	Command_GetTimeout	CCaoCommand::get_Timeout	Acquire the time-out time of command
115	Command_PutTimeout	CCaoCommand::put_Timeout	Set the time-out time of command
116	Command_GetState	CCaoCommand::get_State	Acquire the command state

117	Command_GetParameters	CCaoCommand::get_Parameters	Acquire the command parameter
118	Command_PutParameters	CCaoCommand::put_Parameters	Set the command parameter
119	Command_GetResult	CCaoCommand::get_Result	Acquire the execution result of command
120	Command_GetAttribute	CCaoCommand::get_Attribute	Acquire attribute value of command
121	Command_GetHelp	CCaoCommand::get_Help	Acquire the help character string of command
122	Command_GetName	CCaoCommand::get_Name	Acquire the name of command
123	Command_GetTag	CCaoCommand::get_Tag	Acquire the tag information on command
124	Command_PutTag	CCaoCommand::put_Tag	Set the tag information on command
125	Command_GetID	CCaoCommand::get_ID	Acquire a command ID
126	Command_PutID	CCaoCommand::put_ID	Set a command ID
127	Command_Release	CCaoCommand::Release	Release a command
128	Message_Reply	CCaoMessage::Reply	Response to an event message
129	Message_Clear	CCaoMessage::Clear	Clear an event message
130	Message_GetDateTime	CCaoMessage::get_DateTime	Acquire the time stamp of event message
131	Message_GetDescription	CCaoMessage::get_Description	Acquire the description of event message
132	Message_GetDestination	CCaoMessage::get_Destination	Acquire the destination of event message
133	Message_GetNumber	CCaoMessage::get_Number	Acquire a message number of event message
134	Message_GetSerialNumber	CCaoMessage::get_SerialNumber	Acquire a serial number of event message
135	Message_GetSource	CCaoMessage::get_Source	Acquire the source of

			the event message
136	Message_GetValue	CCaoMessage::get_Value	Acquire the value of event message
137	Message_Release	CCaoMessage::Release	Release an event message