DENSO
Specifications of b-CAP Communication
For RC8

Version 1.0.1

August 20, 2013

[Remarks]

【Revision history】

Date	Revision	Content
2013-2-12	1.0.0	The first release.
2013-8-20	1.0.1	Added the reference of sample libraries written by ANSI-C, Java. Added sample programs about Variable access, Controlling tasks and Controlling the robot. Added the explanation of the Slave Mode speed/accel limit. Added the b-CAP Tester Row Packet Mode. Added the correspondence table about b-CAP function ID and CAO interface.

【Devices】

Device	Version	Notes
RC8	V1.4.0	

Contens

1. Introduction 5
1.1. System Configuration 5
1.2. Reference information 6
1.3. b-CAP Slave Mode 6
2. Setup of RC8 8
2.1. Setup of system parameters 8
2.2. Changing to the AutoMode 10
3. Communication procedure 11
3.1. Variable access 11
3.1.1. Connecting to the server 11
3.1.2. Connecting to the objects 12
3.1.3. Reading and writing the variable 13
3.1.4. Disconnecting the objects 15
3.1.5. Disconnecting the server 16
3.1.6. Access to other variables 16
3.1.7. Sample program 17
3.2. Controlling task 18
3.2.1. Connecting to the object 19
3.2.2. Startting and Stopping the task 20
3.2.3. Disconnecting the object 21
3.2.4. Sample program 22
3.3. Controlling the robot 23
3.3.1. Connecting the object 24
3.3.2. Taking and releasing the arm control authority 25
3.3.3. Turning On or Off the motor 27
3.3.4. Moving and halting the robot 28
3.3.5. Disconnecting the objects 29
3.3.6. Other execute methods 30
3.3.7. Sample program 31
4. How to use the b-CAP Slave Mode 33
4.1. Whats the Slave Mode 33
4.2. The functions of the Slave Mode 33
4.3. The summarys of the Slave Mode 34
4.3.1. Mode0 35
4.3.2. Mode1 37
4.3.3. Mode2 38

4.4. Treatment of buffer underflow........ 39
4.5. The communication procedure of the Slave Mode........ 40
4.5.1. Moving to the initial position........ 41
4.5.2. Starting and stopping the Slave Mode........ 42
4.5.3. Slave Move 44
4.6. Handling the error........ 45
4.6.1. Clearing the error of the RC8........ 45
4.6.2. Restarting the Slave mode........ 46
4.7. Setting about the speed/accel limit........ 46
4.8. Sample programs 47
5. b-CAP Tester........ 50
5.1. The Slave mode by the b-CAP Tester 53
5.1.1. How to test the Slave Mode by the b-CAP Tester 53
5.1.2. Confirming the packet of the Slave move in the b-CAP Tester 55
5.2. About row packet mode 56
5.2.1. Connecting to the controller........ 56
5.2.2. Sending and Receiving the b-CAP packets........ 57
5.3. Description of the VT_ARRAY | VT_VARIANT in the b-CAP Tester 59
Appendix A. Correspondence table about b-CAP function ID and CAO interface........ 61

1. Introduction

This specification provides communication protocol of b-CAP for RC8.

b-CAP is a protocol which is created following the concept of CAP to improve communication speed. Therefore, b-CAP has the same feature as CAP series, as follows.

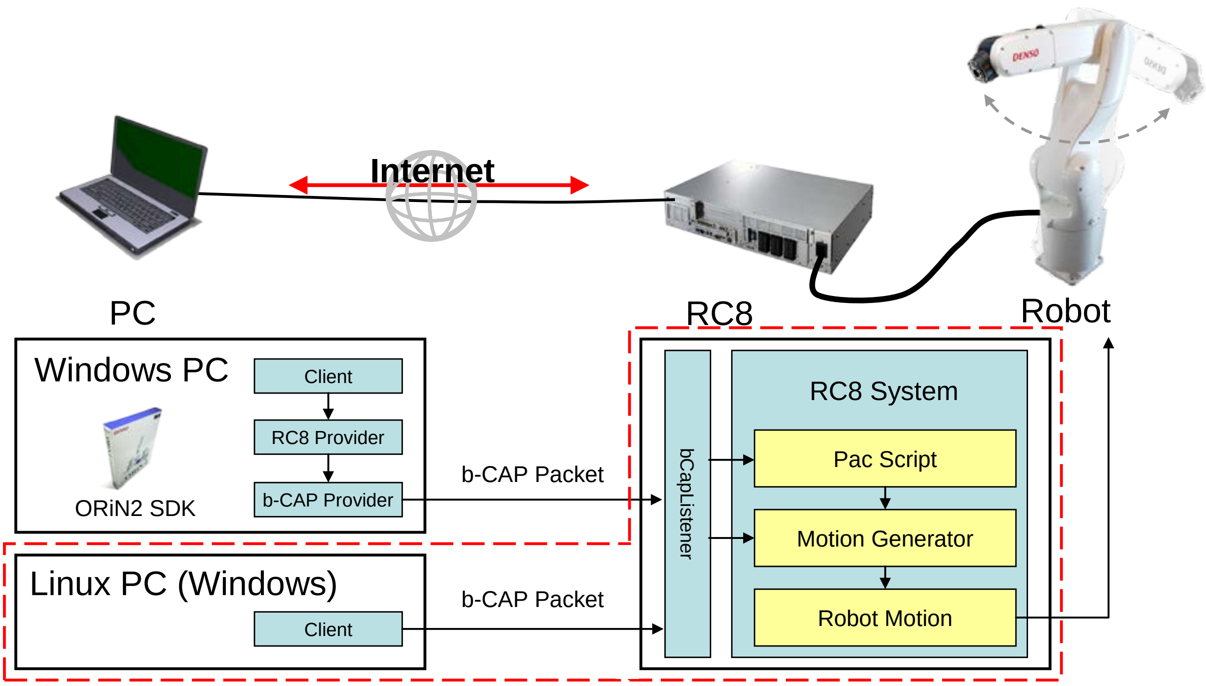
(For more detail information about CAP series, Please refer to “CAP provider User’s guide” (CAP_ProvGuide_en.pdf) included in ORiN2 SDK.)

- It has the same service structure as the object model of CAO provider.
- It calls function by specifying objects by the object ID.
- It provides events of the server by polling.

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 lator version.



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

Client that runs on Windows with ORiN2 SDK-installed can operate RC8 by means of RC8 Provider. RC8

Provider convert commands to b-CAP packets through b-CAP Provider, then transmit 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 packet with concrete examples.

1.2. Reference information

This document includes examples of b-CAP packet transmission which offers the 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.

- Specifications of b-CAP Communication User's guide
ORiN2\CAP\b-CAP\Doc\b-CAP_Spec_en.pdf

Command supported by RC8 specification b-CAP complies with RC8 Provider requirement. For argumentof 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 written by ANSI-C
ORiN2\CAP\b-CAP\CapLib\DENSO\RC8\Include\C
- Sample library written by Java
ORiN2\CAP\b-CAP\CapLib\DENSO\RC8\Include\Java

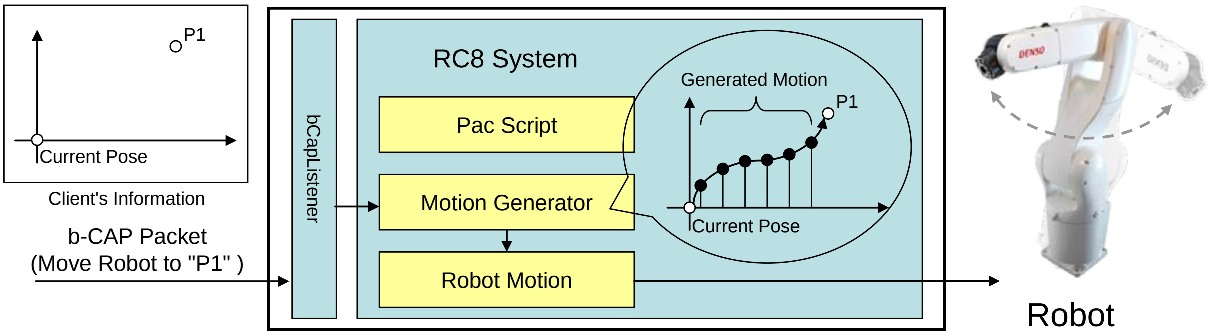
1.3. b-CAP Slave Mode

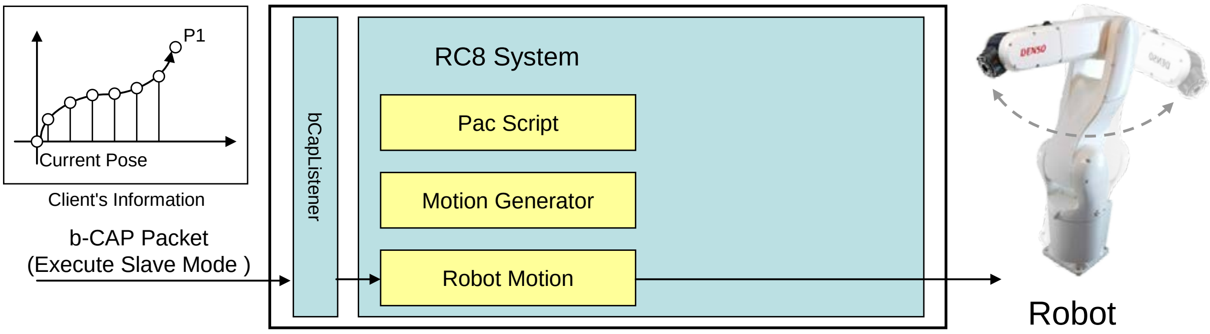
In normal robot motion commands(e.g, “Move 1”, “P1”, etc.), RC8 controls the robot by means of generating trajectory in real time in order to achieve the target posture designated by client. On the other hand, in Slave Mode, client specifies the robot posture in turn in order to control robot motion in real time. By means of this function, client can control trajectory of the robot freely.

Slave Mode is an optional function of RC8 robot controller. Please add the license key depending on 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.

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

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





2. Setup of RC8

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

2.1. Setup of system parameters

Before operating the robot controller by b-CAP packet, you need to setup the robot controller which is to be controlled.

In order to setup the robot controller, you need to prepare either a teachng pendant (TP) or a mini-pendant (MiniTP). Items that have to be set are 1) Communication permission and 2) Executable token.

Communication permission is an item that gives authority 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 permissions to the communication device.

Executable token is an item that provides the communication devices with the right to activate (or perform) a program task in the robot controller, turns the motor power ON, and controls the robot (motion command). Assignable value is any one of 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.

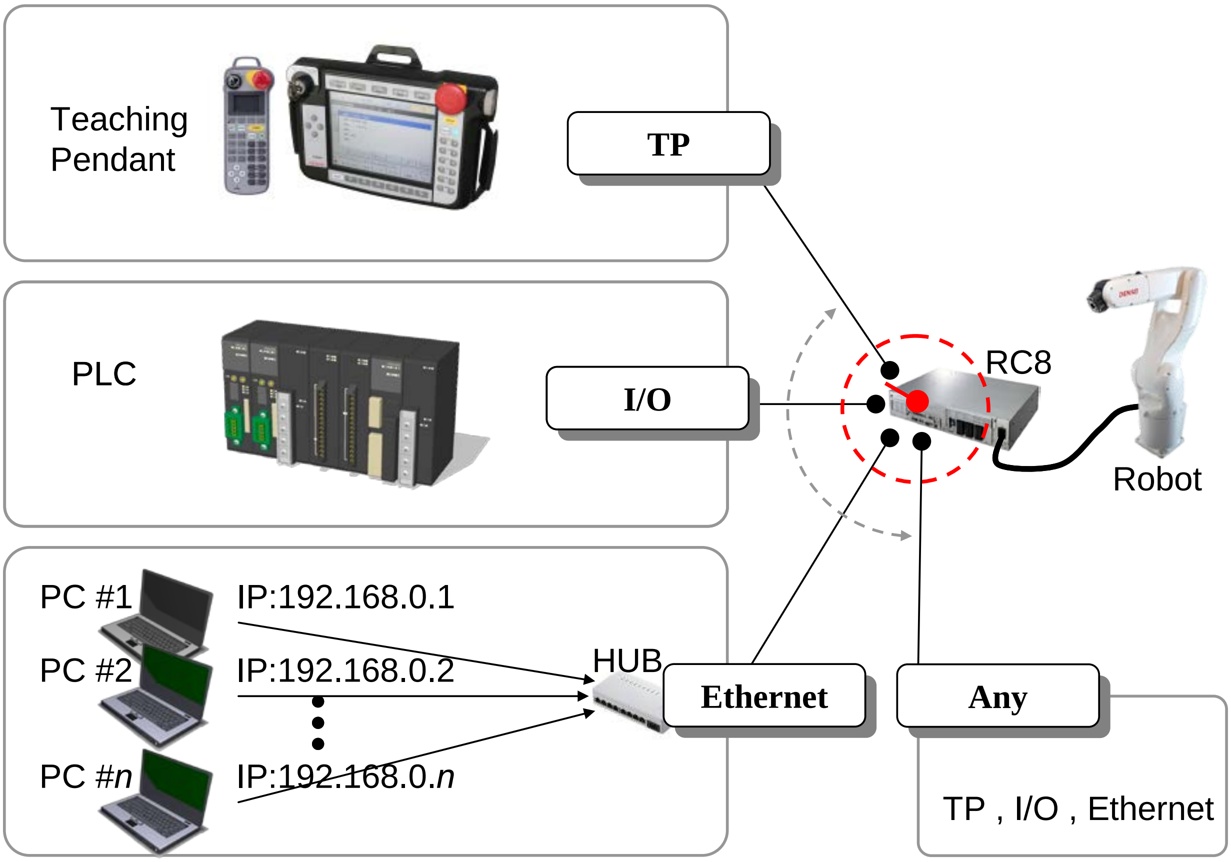


Figure 2-1 Setting of the device with executable token

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.

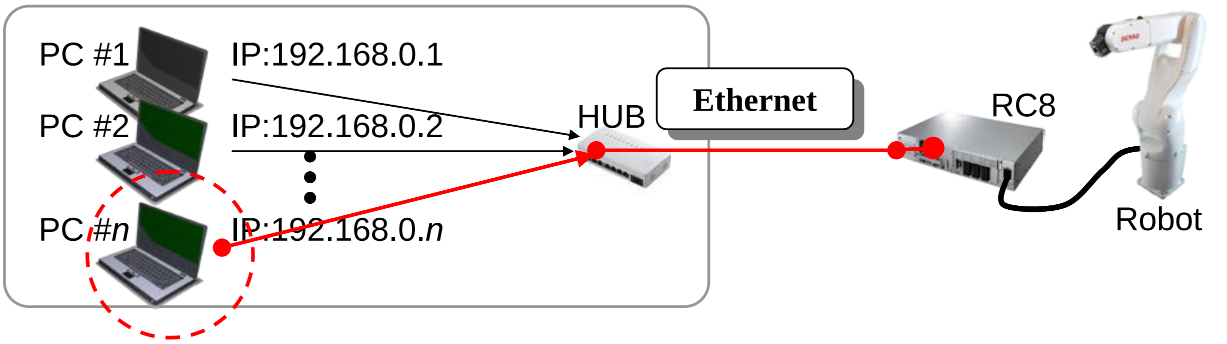


Figure 2-2 Set executable token to Ethernet

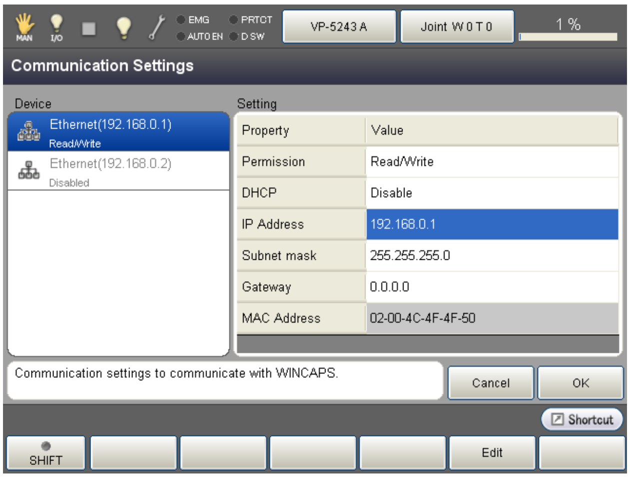


Figure 2-3 Communication permission setting

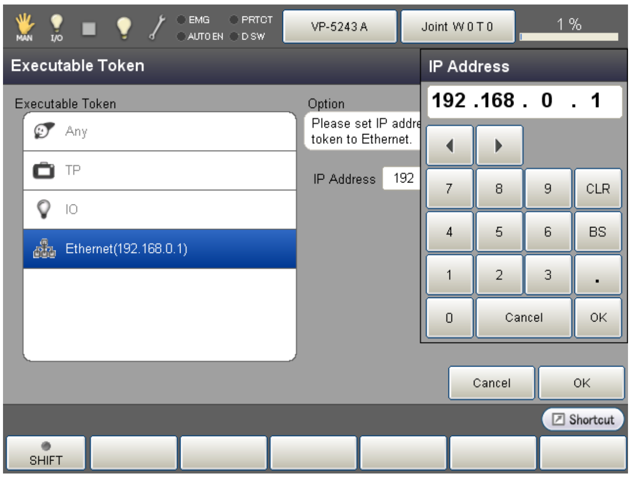


Figure 2-4 Executable token setting

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.

In order to select the auto mode of the robot controller, 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 section describes the procedure for communicating with the robot controller by means of b-CAP for RC8 with showing concrete examples as follows.

3.1. Variable access

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

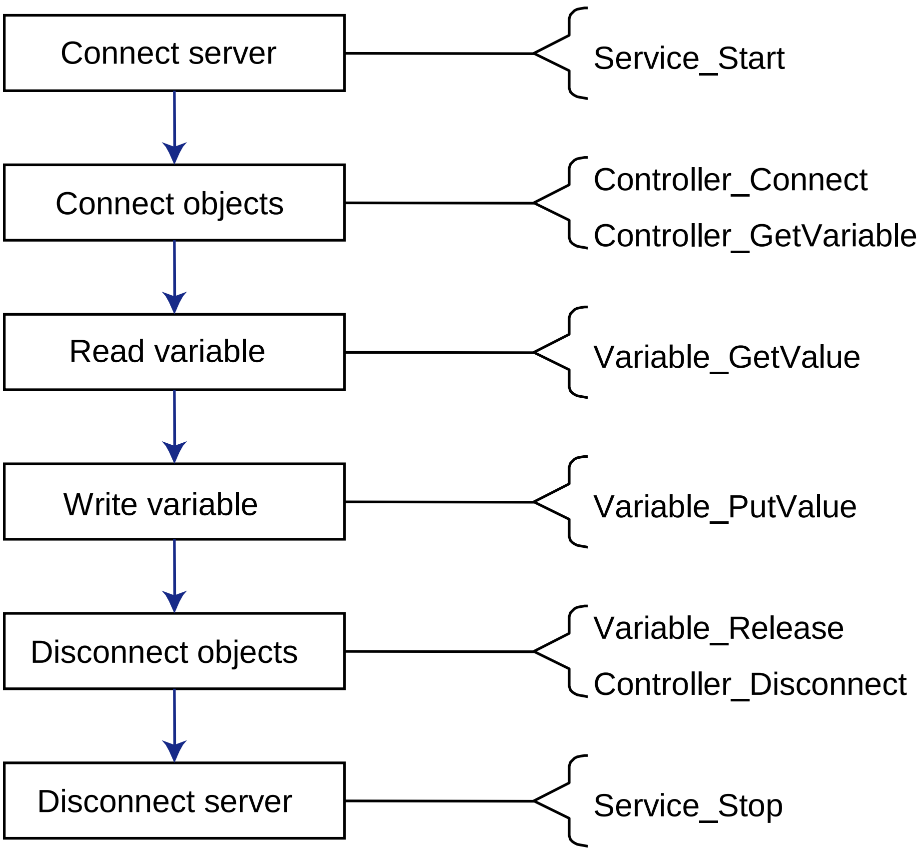


Figure 3-1 Flow of the variable access

3.1.1. Connecting to the server

Sending the “Service_Start” packet starts a server service of RC8.

Service_Start				
Packet TX	Client -> Server 01 10 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	-	-	-
		-		

3.1.2. Connecting to the objects

Connect to each object to acquire the handle of the controller object. Controller handle is required when connecting the variable object 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.

Controller_Connect				
This function returns the handle of the controller object.				
Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		
	bstrCtrlName	The name of the controller	VT_BSTR	“b-CAP”
		14 00 00 00 08 00 01 00 00 00 0A 00 00 00 62 00 2D 00 43 00 41 00 50 00		
	bstrProvName	The name of the provider	VT_BSTR	“CaoProv.DENSO.VRC”
		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		
	bstrPcName	The name of the client PC	VT_BSTR	“192.168.0.1”
		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		

	bstrOption	The connecting option	VT_BSTR	Null String
		0A 00 00 00 08 00 01 00 00 00 00 00 00 00		
Packet RX	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			
	Argument	Description	Data Type	Value
		Binary		
	hController	The handle of the controller	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		

Controller_GetVariable				
This function returns the handle of the variable object.				
Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		
	hController	The handle of the controller	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrName	The name of the variable	VT_BSTR	“IO150”
		14 00 00 00 08 00 01 00 00 00 0A 00 00 00 49 00 4F 00 31 00 35 00 30 00		
	bstrOption	The 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 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	The handler of the variable	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		

3.1.3. Reading and writing the variable

Read and write the value 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="4">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 rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>hVariable</td><td>The handler of the variable</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 03 00 00 00</td></tr>
<tr><td rowspan="4">Packet
RX</td><td colspan="4">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</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>pVal</td><td>The value of the variable “IO150“</td><td>VT_BOOL</td><td>0x0000 (FALSE)</td></tr>
<tr><td></td><td></td><td colspan="3">00 08
00 00 00 0B 00 01 00 00 00 00 00</td></tr>
</table>

<table>
<tr><th colspan="5">Variable_PutValue</th></tr>
<tr><td colspan="5">This function put the value to the variable specified by “hVariable”.</td></tr>
<tr><td rowspan="6">Packet
TX</td><td colspan="4">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</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 rowspan="2">hVariable</td><td>The handler of the variable</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">newVal</td><td>The value to put</td><td>VT_BOOL</td><td>0xFFFF (TRUE)</td></tr>
<tr><td></td><td colspan="3">08 00 00
00 0B 00 01 00 00 00 FF FF</td></tr>
<tr><td rowspan="4">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 05 00 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 rowspan="2">No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

3.1.4. Disconnecting the objects

Disconnect the 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.

<table>
<tr><th colspan="5">Variable_Release</th></tr>
<tr><td colspan="5">This function disconnects the client from the variable object specified by the handle of the variable “hVariable”.</td></tr>
<tr><td rowspan="5">Packet
TX</td><td colspan="4">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</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 rowspan="2">hVariable</td><td>The handler of the variable</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="5">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 06 00 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 rowspan="2">No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

<table>
<tr><th colspan="5">Controller_Disconnect</th></tr>
<tr><td colspan="5">This function disconnects the client from the controller object specified by the handle of the controller “hController”.</td></tr>
<tr><td rowspan="5">Packet
TX</td><td colspan="4">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</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 rowspan="2">hController</td><td>The handle of the controller</td><td>VT_I4</td><td>0x00000002</td></tr>
<tr><td colspan="3">0A
00 00 00 03 00 01 00 00 00 02 00 00 00</td></tr>
<tr><td rowspan="5">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 07 00 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 rowspan="2">No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

3.1.5. Disconnecting 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 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	-	-	-
		-		

3.1.6. Access to other variables

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

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

```
Dim caoVar as CaoVariable
Set caoVar = caoCtrl.AddVariable("@MODE","")   'Specifying the system variable @MODE
```

The above is translated into b-CAP 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	The handle of the controller	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		

	bstrName	The name of the variable	VT_BSTR	“@MODE”
		14 00 00 00 08 00 01 00 00 00 0A 00 00 00 40 00 4D 00 4F 00 44 00 45 00		
	bstrOption	The 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 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	The handler of the variable	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. CAO class objects, such as caoCtrl or caoVar, correspond to the b-CAP object handles. In this case, caoCtrl and caoVar correspond to the handle of the controller and the handle of the variable, respectively.

3.1.7. Sample program

Following is an example program, which accesses variables, with using ANSI-C sample library.

The 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.cpp

```
#include "b-Cap.c"

#define SERVER_IP_ADDRESS        "192.168.0.1"
#define SERVER_PORT_NUM          5007

int main()
{
        int iSockFD;
        u_long lhController;
        BCAP_HRESULT hr = BCAP_S_OK;

        /* Init and Start b-CAP */
        hr = bCap_Open(SERVER_IP_ADDRESS, SERVER_PORT_NUM, &iSockFD);

        /* Init socket */
        if FAILED(hr) return (hr);

        /* Start b-CAP service */
        hr = bCap_ServiceStart(iSockFD);

        /* Get controller handle */
```

```
	hr = bCap_ControllerConnect(iSockFD, "b-CAP", "caoProv.DENSO.VRC", SERVER_IP_ADDRESS, "",
&lhController);

	u_long lhVar;
	long lResult;

	/* Get variable handle */
	hr = bCap_ControllerGetVariable(iSockFD, lhController, "IO150", "", &lhVar);

	/* Read variable */
	bCap_VariableGetValue(iSockFD, lhVar, &lResult);

	/* Write variable */
	lResult = -1L;
	bCap_VariablePutValue(iSockFD, lhVar, VT_BOOL, 1, &lResult);

	/* Release variable handle */
	bCap_VariableRelease(iSockFD, lhVar);

	/* Release controller handle */
	bCap_ControllerDisconnect(iSockFD, lhController);

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

	bCap_Close(iSockFD);

	return 0;
}
```

3.2. Controlling task

When executing task control, you need to follow the procedure shown in Figure 3-2. In order to run the task, 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.

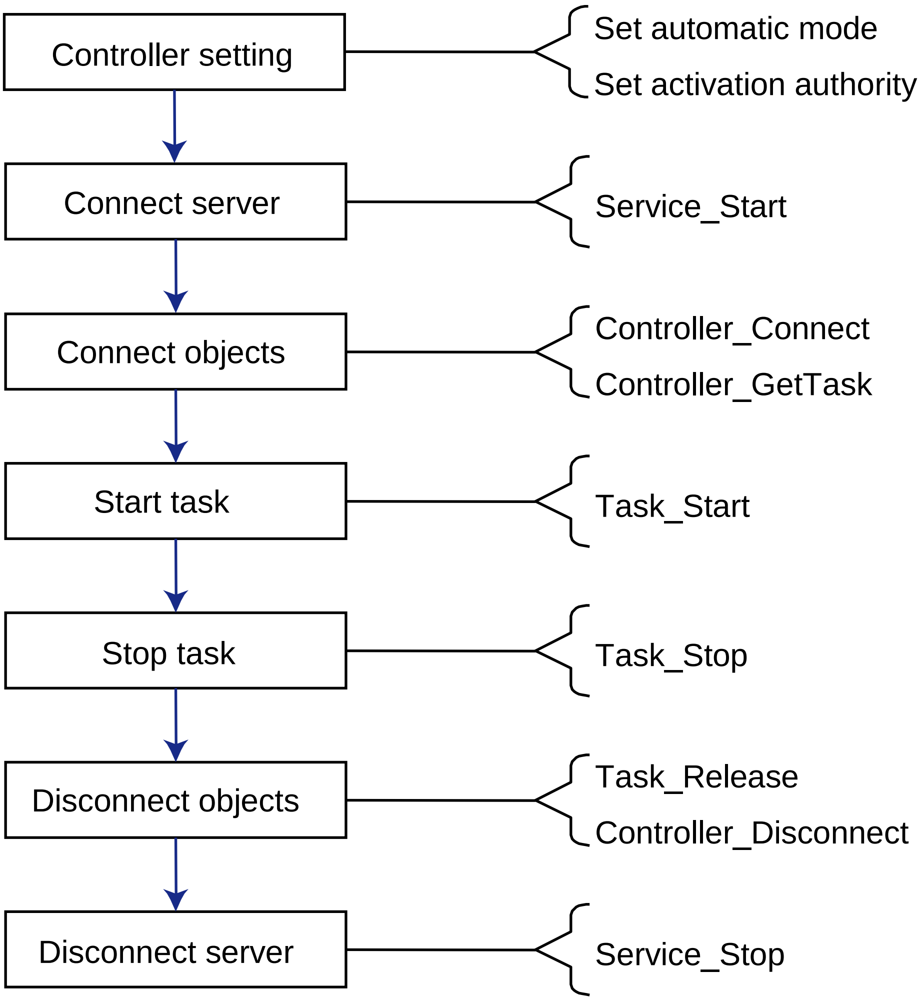


Figure 3-2 Flow of the task control

3.2.1. Connecting to the object

With regards to the procedures until connecting the controller object, refer to "3.1Variable access". To connect to the 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	The handle of the controller	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrName	The name of the task	VT_BSTR	“Pro1”
		12 00 00 00 08 00 01 00 00 00 08 00 00 00 50 00 72 00 6F 00 31 00		
	bstrOption	The 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 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	The handler of the task	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		

3.2.2. Startting and Stopping the task

Start and stop the connected task. To start the task, use Task_Start (88) as function ID. To stop the 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 58 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	The handler of the task	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	lMode	The start mode(=One cycle execution)	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrOption	The 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 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	The handler of the task	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	lMode	The stop mode(3:Cycle stop)	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrOption	The option string	VT_BSTR	Null String
		0A 00 00 00 08 00 01 00 00 00 00 00 00 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 the object

Disconnect from the object. With regards to the procedures after the task object disconnection, refer to "3.1Variable access". To disconnect the task object, use Task_Release (99) as function ID. The following table shows a packet to disconnect the 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="4">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 rowspan="2">hTask</td><td>The handler of the task</td><td>VT_I4</td><td>0x00000003</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 03 00 00 00</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 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 rowspan="2">No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

3.2.4. Sample program

Following is an example program, which controls tasks, with using ANSI-C sample library.

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

List 3-2 **bCapTask.cpp**

```
#include "b-Cap.c"

#define SERVER_IP_ADDRESS        "192.168.0.1"
#define SERVER_PORT_NUM          5007

int main()
{
	int iSockFD;
	u_long lhController;
	BCAP_HRESULT hr = BCAP_S_OK;

	/* Init and Start b-CAP */
	hr = bCap_Open(SERVER_IP_ADDRESS, SERVER_PORT_NUM, &iSockFD);

	/* Init socket */
	if FAILED(hr) return (hr);

	/* Start b-CAP service */
	hr = bCap_ServiceStart(iSockFD);

	/* Get controller handle */
	hr = bCap_ControllerConnect(iSockFD, "b-CAP", "caoProv.DENSO.VRC", SERVER_IP_ADDRESS, "",
&lhController);

	u_long lhTask;
	long lMode;

	/* Get task handle */
	hr = bCap_ControllerGetTask(iSockFD, lhController, "Pro1", "", &lhTask);

	/* Start task */
	lMode = 2L;
	bCap_TaskStart(iSockFD, lhTask, lMode, "");
```

```
        /* Stop task */
        lMode = 3L;
        bCap_TaskStop(iSockFD, lhTask, lMode, "");

        /* Release task handle */
        bCap_TaskRelease(iSockFD, lhTask);

        /* Release controller handle */
        bCap_ControllerDisconnect(iSockFD, lhController);

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

        bCap_Close(iSockFD);

        return 0;
}
```

3.3. Controlling the robot

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.

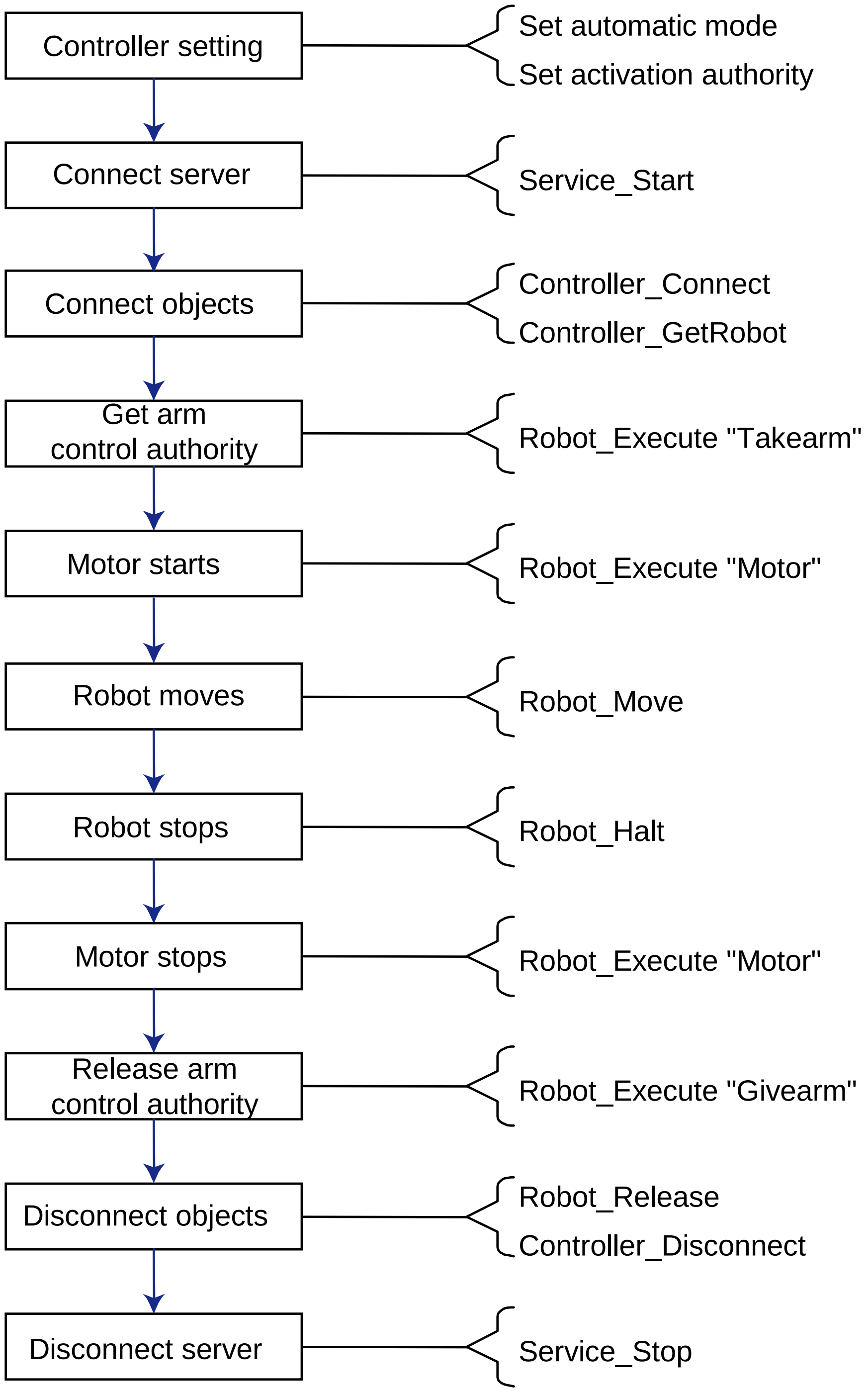


Figure 3-3 Flow of the robot control

3.3.1. Connecting the object

With regards to the procedures until connecting the controller object, refer to “3.1Variable access“. To connect to the 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.

Controller_GetRobot				
This function gets the handle of the robot object – hRobot.				
Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		
	hController	The handle of the controller	VT_I4	0x00000002
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrName	The name of the robot	VT_BSTR	“Arm”
		10 00 00 00 08 00 01 00 00 00 06 00 00 00 41 00 72 00 6D 00		
	bstrOption	The 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 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		
	hRobot	The handle of the robot	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		

3.3.2. Taking and releasing the arm control authority

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

Robot_Execute “Takearm”, (0, 1)				
This function takes the arm control authority.				
Packet TX	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			
	Argument	Description	Data Type	Value

<table>
<tr><td rowspan="7"></td><td></td><td colspan="3">Binary</td></tr>
<tr><td rowspan="2">hRobot</td><td>The handle of the robot</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">bstrCommand</td><td>The command string</td><td>VT_BSTR</td><td>“Takearm”</td></tr>
<tr><td colspan="3">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</td></tr>
<tr><td rowspan="2">vntParam</td><td>The command parameter</td><td>VT_I4 |
VT_ARRAY</td><td>0, 1</td></tr>
<tr><td colspan="3">0E 00 00 00 03 20 02
00 00 00 00 00 00 00 01 00 00 00</td></tr>
<tr><td rowspan="5">Packet
RX</td><td colspan="4">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</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 rowspan="2">vntReturn</td><td>Return Value</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td colspan="3">06
00 00 00 00 00 01 00 00 00</td></tr>
</table>

<table>
<tr><td colspan="5">Robot_Execute “Givearm”</td></tr>
<tr><td colspan="5">This function releases the arm control authority</td></tr>
<tr><td rowspan="10">Packet
TX</td><td colspan="4">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</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 rowspan="2">hRobot</td><td>The handle of the robot</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">bstrCommand</td><td>The command string</td><td>VT_BSTR</td><td>“Givearm”</td></tr>
<tr><td colspan="3">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</td></tr>
<tr><td rowspan="2">vntParam</td><td>The parameter</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td colspan="3">06 00 00 00 00 00 01
00 00 00</td></tr>
<tr style="display:none"></tr>
<tr><td rowspan="4">Packet
RX</td><td colspan="4">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</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>vntReturn</td><td>Return Value</td><td>VT_EMPTY</td><td>-</td></tr>
</table>

		00 00 00 00 00 01 00 00 00 06

3.3.3. Turning On or Off the motor

In order to control the robot, the motor of the robot needs to be turning ON. The motor control is mounted as a command of Robot_Execute (64). The following table shows the example of packets that turns ON and OFF the motor.

Robot_Execute “Motor”, (1, 0)				
This function turns On the 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	The handle of the robot	VT_I4	0x00000003
		00 00 00 03 00 01 00 00 00 03 00 00 00 0A		
	bstrCommand	The command string	VT_BSTR	“Motor”
		00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 14 00 00 00 6F 00 72 00		
	vntParam	The parameter	VT_I4 | VT_ARRAY	1, 0
		0E 00 00 00 03 20 02 00 00 00 01 00 00 00 00 00 00 00		
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 the 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	The handle of the robot	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrCommand	The command string	VT_BSTR	“Motor”
		14 00 00 00 08 00 01 00 00 00 0A 00 00 00 4D 00 6F 00 74 00 6F 00 72 00		
	vntParam	The parameter	VT_I4 | VT_ARRAY	0, 0
		0E 00 00 00 03 20 02 00 00 00 00 00 00 00 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	-
		06 00 00 00 00 00 01 00 00 00		

3.3.4. Moving and halting the robot

In order to move the 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, the 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	The handle of the robot	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	lComp	The interpolation mode	VT_I4	0x00000001
		0A 00 00 00 03 00 01 00 00 00 01 00 00 00		
	vntPose	The destination positons	VT_BSTR	“P1”
		0E 00 00 00 08 00 01 00 00 00 04 00 00 00 50 00 31 00		
	bstrOption	The 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 the 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	The handle of the robot	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
	bstrOption	The 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 the objects

Disconnect from the object connected to. The procedure after the robot disconnection, refer to “3.1Variable access”. To disconnect the 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 the client from the robot object specified by the handle of the robot "hRobot".				
Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		
	hRobot	The handle of the robot	VT_I4	0x00000003
		0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
Packet RX	Server -> Client: 01 10 00 00 00 0B 00 00 00 00 00 00 00 00 00 04			
	Argument	Description	Data Type	Value
		Binary		
	No-Args	-	-	-
		-		

3.3.6. Other execute methods

RC8 has provider-specific extended commands for each CAO class objects. For details about the extended commands supported by RC8, refer to the "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 the "RC8 Provider Guide", the procedure to execute "ClearError", which is the extended command in controller class, is expressed as follows.

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

The above is translated into b-CAP as follows.

Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		

<table>
<tr><td rowspan="6"></td><td rowspan="2">hController</td><td>The handle of the controller</td><td>VT_I4</td><td>0x0000002</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00 00 02 00 00 00 0A</td></tr>
<tr><td rowspan="2">bstrCommand</td><td>The command string</td><td>VT_BSTR</td><td>“ClearError”</td></tr>
<tr><td colspan="3">00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 1E 00 00
00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00</td></tr>
<tr><td rowspan="2">vntParam</td><td>The parameter</td><td>VT_EMPTY</td><td></td></tr>
<tr><td colspan="3">00 00 00 00 00 01 00 00 00 06</td></tr>
<tr><td rowspan="4">Packet
RX</td><td colspan="4">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</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>vntReturn</td><td>Return Value

00 00 00 00 00 01 00 00 00 06</td><td>VT_EMPTY</td><td>-</td></tr>
</table>

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 handle of the controller. 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 an example program, which controls the robot, with using ANSI-C sample library.

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

List 3-3 **bCapRobot.cpp**

```
#include "b-Cap.c"

#define SERVER_IP_ADDRESS         "192.168.0.1"
#define SERVER_PORT_NUM           5007

int main()
{
        int iSockFD;
        u_long lhController;
        BCAP_HRESULT hr = BCAP_S_OK;

        /* Init and Start b-CAP */
        hr = bCap_Open(SERVER_IP_ADDRESS, SERVER_PORT_NUM, &iSockFD);

        /* Init socket */
        if FAILED(hr) return (hr);
```

```
        /* Start b-CAP service */
        hr = bCap_ServiceStart(iSockFD);

        /* Get controller handle */
        hr = bCap_ControllerConnect(iSockFD, "b-CAP", "caoProv.DENSO.VRC", SERVER_IP_ADDRESS, "",
&lhController);

        u_long lhRobot;
        long lResult;

        /* Get robot handle */
        hr = bCap_ControllerGetRobot(iSockFD, lhController, "Arm", "", &lhRobot);

        /* Get arm control authority */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Takearm", "", &lResult);

        /* Motor on */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Motor", "1", &lResult);

        /* Move to P1 */
        hr = bCap_RobotMove(iSockFD, lhRobot, 1L, "P1", "");

        /* Motor off */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Motor", "0", &lResult);

        /* Release arm control authority */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Givearm", "", &lResult);

        /* Release robot handle */
        bCap_RobotRelease(iSockFD, lhRobot);

        /* Release controller handle */
        bCap_ControllerDisconnect(iSockFD, lhController);

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

        bCap_Close(iSockFD);

        return 0;
}
```

4. How to use the b-CAP Slave Mode

4.1. Whats the Slave Mode

Slave mode is a function to control the robot by transmitting the position and posture data in short time period

Following three functions are mounted in b-CAP as Slave Mode.

- slvChangeMode: Change the setting of the Slave Mode.
- slvGetMode: Acquire the current setting of the Slave Mode.
- slvMove: Move the robot to the designated position and posture

4.2. The 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 the robo
	VT_BSTR	bstrCommand	Command string
	VARIANT	vntParam	Parameters
Return Value	VARIANT	pVal	ResultValue
Description	‘Execute commands of the robot(hRobot).		

Command is executed by entering the command name shown below into “bstrCommand”

Table 4-1 The Slave Mode functions

<table>
<tr><th>CommandString</th><th>Parameter</th><th>Return Value</th><th>Behaviour</th></tr>
<tr><td>slvChangeMode</td><td><SlaveMode:VT_I4>
<table>
<tr><th>Value</th><th>Type</th><th>Motion</th></tr>
<tr><td>0x000</td><td>-</td><td>Mode release</td></tr>
<tr><td>0x001</td><td>P type</td><td>Mode 0</td></tr>
<tr><td>0x002</td><td>J type</td><td>Mode 0</td></tr>
<tr><td>0x003</td><td>T type</td><td>Mode 0</td></tr>
<tr><td>0x101</td><td>P type</td><td>Mode 1</td></tr>
<tr><td>0x102</td><td>J type</td><td>Mode 1</td></tr>
<tr><td>0x103</td><td>T type</td><td>Mode 1</td></tr>
</table></td><td>None</td><td>Change the Slave Mode settings. To switch to the Slave Mode, the robot has to be in stopped state. The client to be changed to the Slave Mode has to have an arm control</td></tr>
</table>

	0x201 P type Mode 2 0x202 J type Mode 2 0x203 T type Mode 2		authority as well.
slvGetMode	None	<Slave Mode:VT_I4> See parameter of "slvChangeMode".	Acquire the currentsetting of the Slave Mode.
slvMove	<Designate position・posture :VT_R8 | VT_ARRAY>	<Current position of the robot (J type):VT_R8 | VT_ARRAY>	Move the robot to the designated position and posture.

4.3. The summarys 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 outlines of each mode.

Table 4-2 Slave Mode summarys

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 creates message sending thread in certain intervals. Message sending thread keeps creating “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 creating the message sending thread because the buffer becomes full, and waits until the server processes the messages.

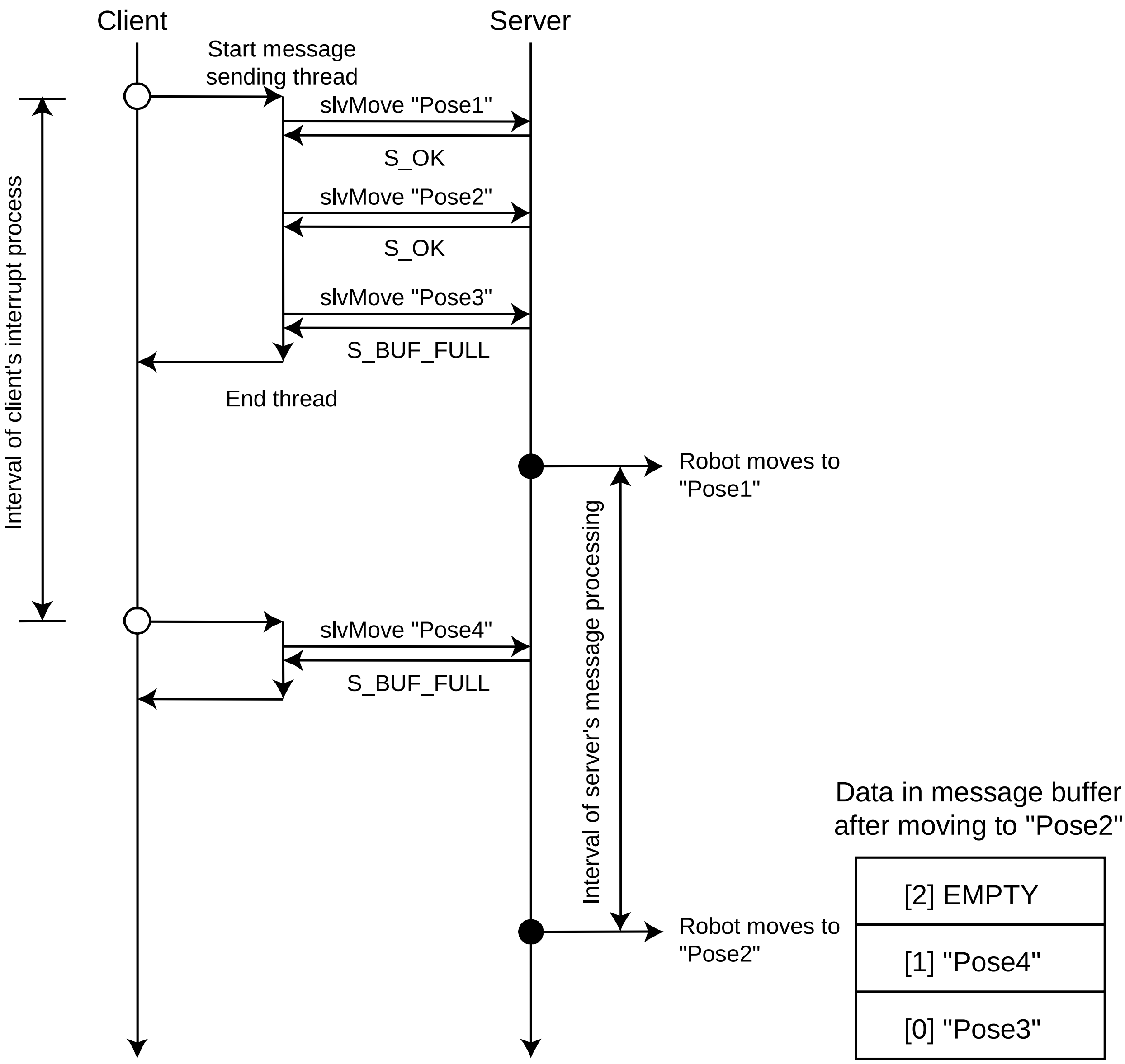


Figure 4-1 The communication procedure in the Mode0

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 processed, then re-send the "slvMode" message that triggers "Buffer Overflow" message.

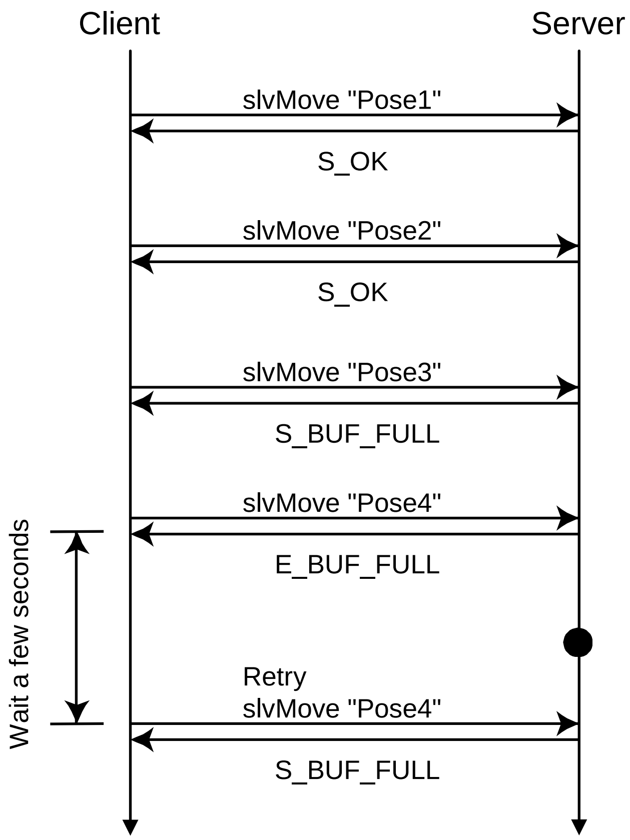


Figure 4-2 Process when buffer overflow occures

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 is the message that is sent by the client immediately before, because the server keeps overwriting the buffer. Therefore the message processed by the server is the message sent by the client immediately before.

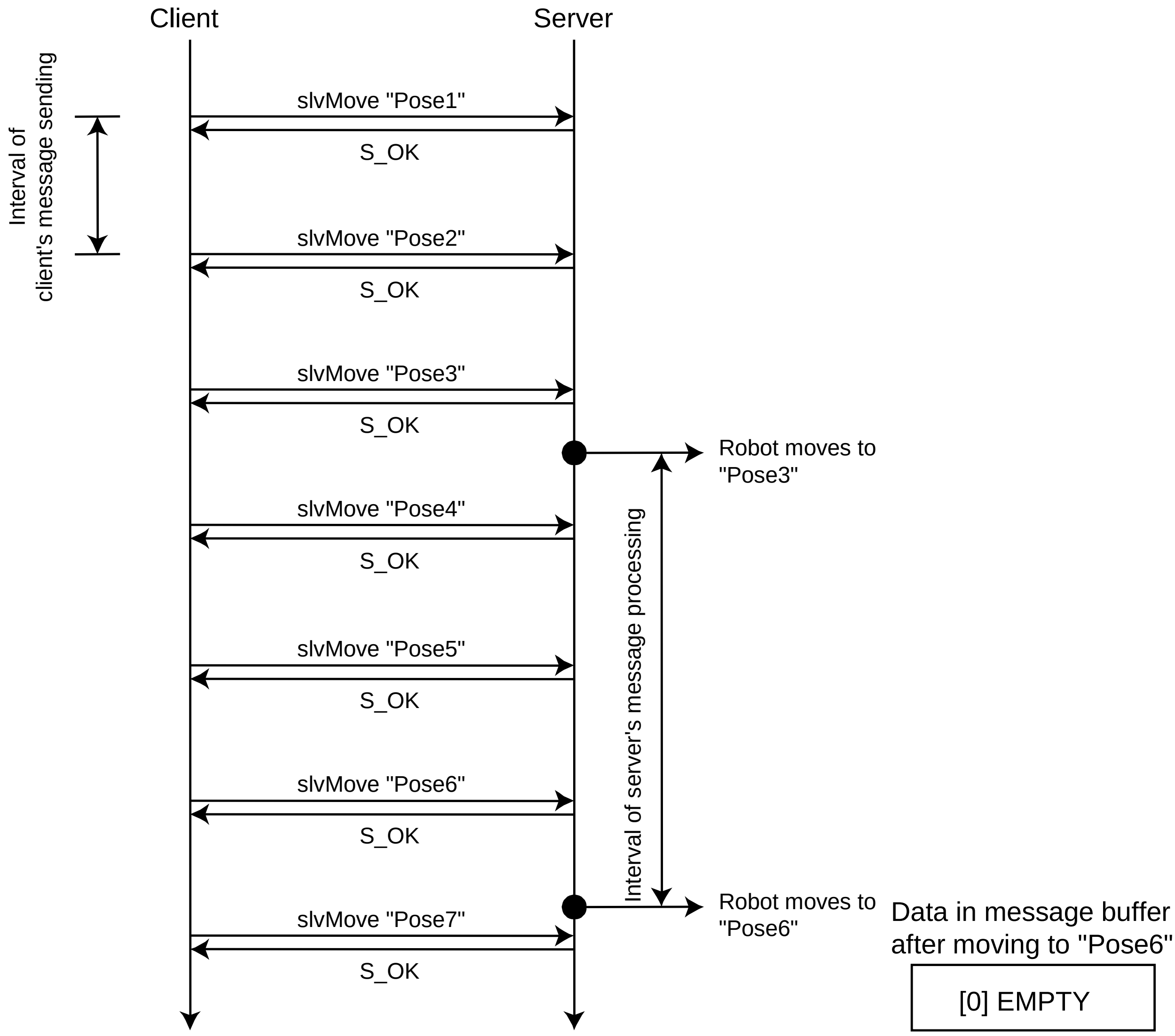


Figure 4-3 The communication procedure in the Mode1

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, when the “slvMove” message returns due to full buffer, the server does not send return code until the buffer space is secured.

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

The “slvMove “Pose5”” is transmitted when the buffer is full. Therefore, the return code does not return until the server moves the robot to “Pose2”. As a result, the client becomes standby state automatically until the buffer space is secured. By this, the client can achieve Slave Mode without mounting the processing such as “Quitting the thread by monitoring the return code” like Mode 0.

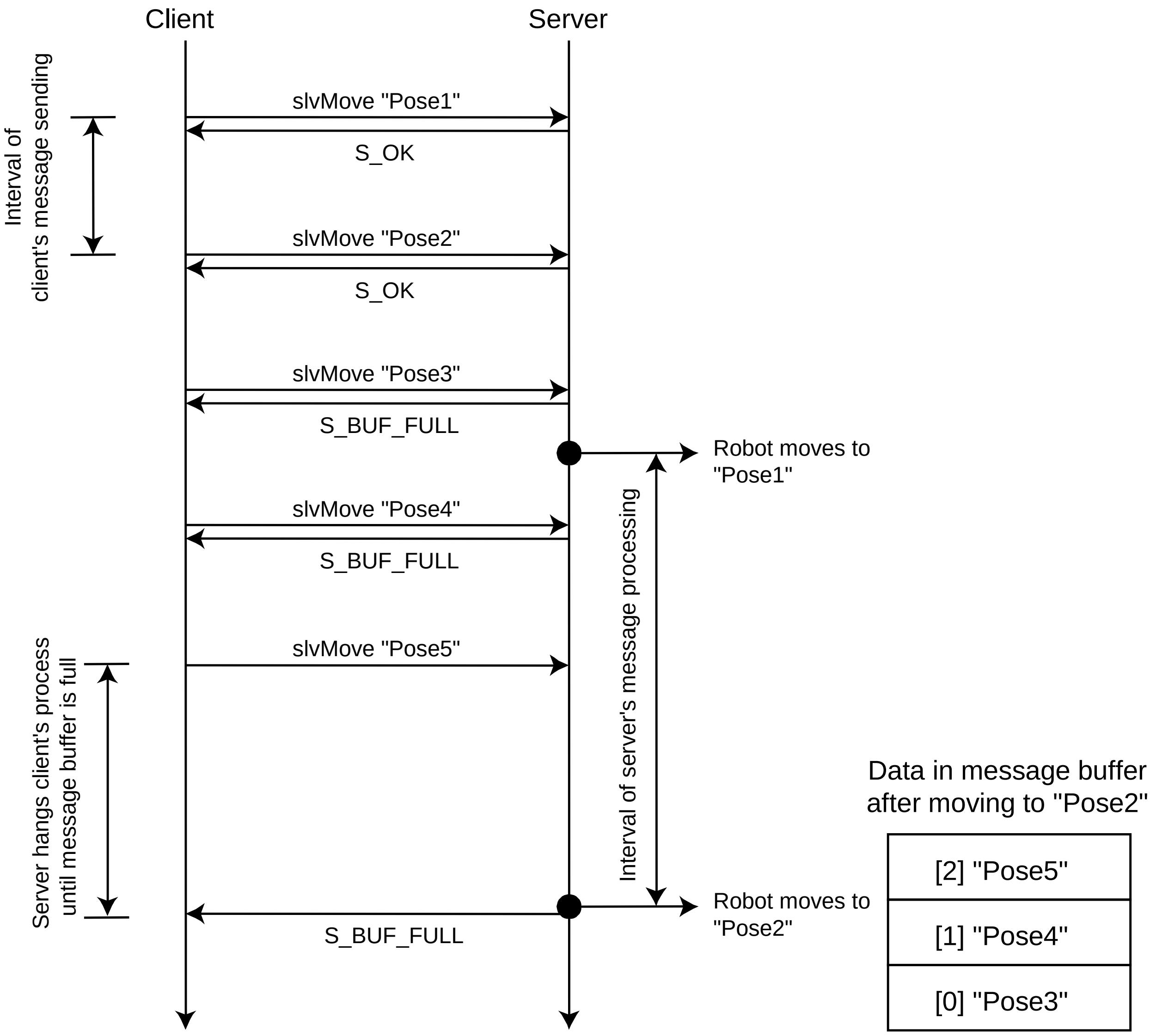


Figure 4-4 The communication procedure in the Mode2

4.4. Treatment of buffer underflow

As explained in "4.3 The summarys 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 4-3 shows the behavior of the server side in buffer underflow state.

Table 4-3 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.	SlaveMode is released

		(Error : 0x84201482 = Command value creation delay)	
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= Command value creation delay)	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. And other statuses are deemed as "Running state".

Mode 0 or Mode 2 issues "Command value creation delay (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 "Excessive command acceleration in *-axis (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 "Excessive command acceleration in *- axis (0x8420404*)" might be issued.

4.5. The 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/ after disconnecting each objects, refer to “3.3Controlling the robot”. Each step is described in more detail below.

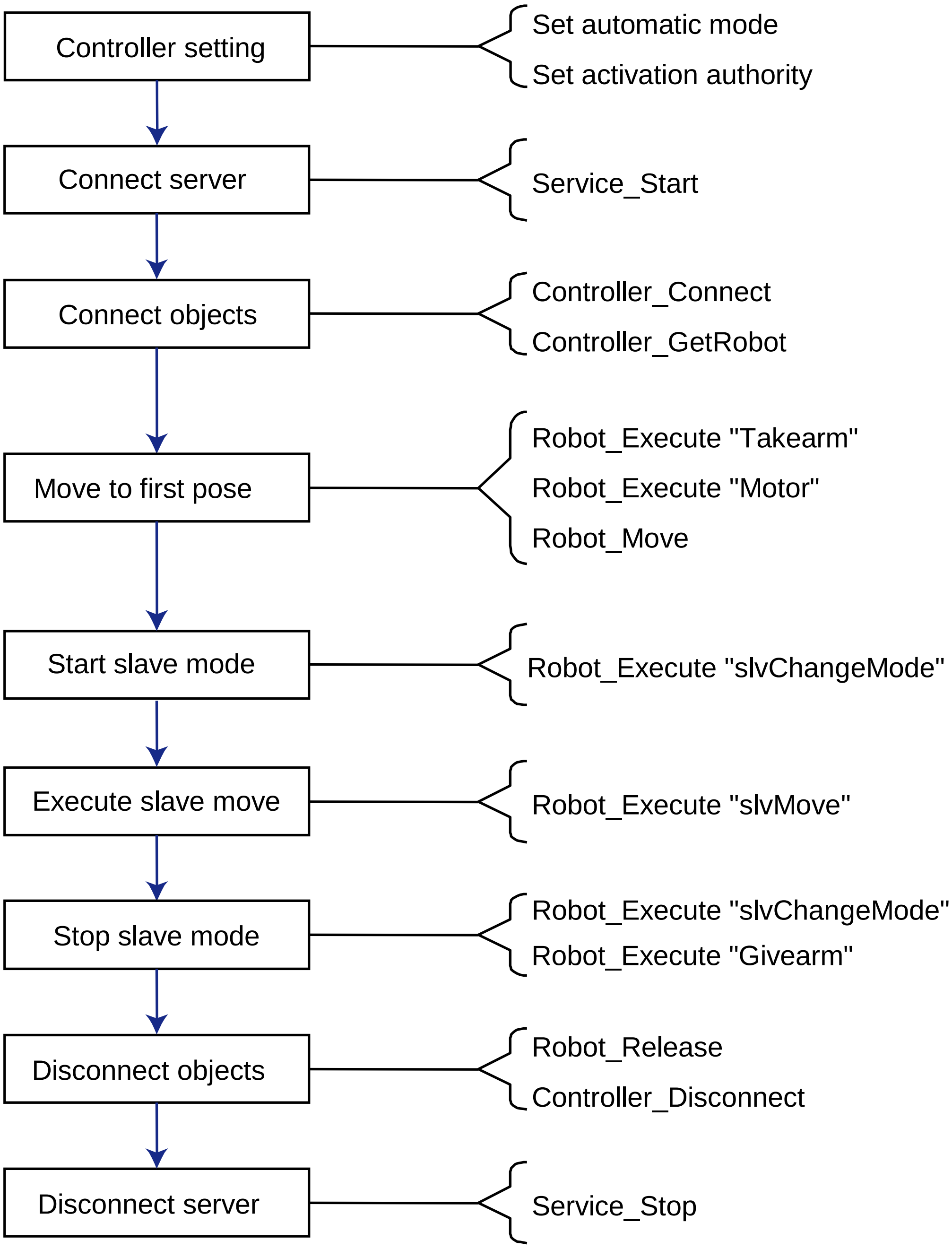


Figure 4-5 Flow of the Slave Mode

4.5.1. 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 “3.3Controlling the robot”.

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="5">Robot_Move 1, “@E P1”, “”</td></tr>
<tr><td colspan="5">This executes the motion command “MOVE 1, “@E P1” “””.</td></tr>
<tr><td rowspan="10">Packet
TX</td><td colspan="4">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 rowspan="2">Argument</td><td>Description</td><td>Data Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td rowspan="2">hRobot</td><td>The handle of the robot</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>The 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>The destination positions</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>The motion option</td><td>VT_BSTR</td><td>“”</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="4">Packet
RX</td><td colspan="4">Server -> Client:
01 10 00 00 00 06 00 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>
</table>

4.5.2. 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 arm control authority. For details instruction of how to acquire the arm control authority, refer to “3.3 Controlling the robot”. If you send a packet to stop the 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.

Packet TX	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

Argument	Description	Data Type	Value
	Binary		
hRobot	The handle of the robot	VT_I4	0x00000003
	0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
bstrCommand	The command string	VT_BSTR	“slvChangeMode”
	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		
vntParam	The parameters	VT_I4	0x001
	0A 00 00 00 03 00 01 00 00 00 01 00 00 00		

Packet RX	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

Argument	Description	Data Type	Value
	Binary		
vntReturn	Return Value	VT_EMPTY	-
	06 00 00 00 00 00 01 00 00 00		

Robot_Execute “slvChangeMode”, 0x000
Exit the Slave Mode.

Packet TX	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

Argument	Description	Data Type	Value
	Binary		
hRobot	The handle of the robot	VT_I4	0x00000003
	0A 00 00 00 03 00 01 00 00 00 03 00 00 00		
bstrCommand	The command string	VT_BSTR	“slvChangeMode”
	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		
vntParam	The parameters	VT_I4	0x000

<table>
<tr><td></td><td></td><td colspan="3">0A 00 00 00 03 00 01 00 00 00 00
00 00 00</td></tr>
<tr><td rowspan="4">Packet
RX</td><td colspan="4">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</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 rowspan="2">vntReturn</td><td>Return Value</td><td>VT_EMPTY</td><td>-</td></tr>
<tr><td></td><td></td><td colspan="3">06
00 00 00 00 00 01 00 00 00</td></tr>
</table>

4.5.3. 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 “4.3The summarys of the Slave Mode”.

<table>
<tr><td colspan="5">Robot_Execute “slvMove”</td></tr>
<tr><td colspan="5">This function sends the destination position by the slvMove command.</td></tr>
<tr><td rowspan="9">Packet
TX</td><td colspan="4">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</td></tr>
<tr><td rowspan="2">Argument</td><td>Description</td><td>Date Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td rowspan="2">hRobot</td><td>The handle of the robot</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">bstrCommand</td><td>The command string</td><td>VT_BSTR</td><td>“slvMove”</td></tr>
<tr><td colspan="3">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</td></tr>
<tr><td rowspan="2">vntParam</td><td>The parameter</td><td>VT_R8 |
VT_ARRAY</td><td>364.16, 0, 278.5355, 180,
1.272222E-14, 180, 5</td></tr>
<tr><td colspan="3">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</td></tr>
</table>

Packet RX	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			
	Argument	Description	Data Type	Value
		Binary		
	vntReturn	Return Value	VT_R8 | VT_ARRAY	-1.34873E-15, 45.0, 90, 9.922799E-15, 45, -1.348731E-15, 0, 0
		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			

4.6. 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.

4.6.1. 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 by the TeachPendant, and by sending a packet to clear the error. Error clear is implemented as a command of “Controller_Execute(17)”. The following example shows a packet to clear an error.

Controller_Execute “ClearError”				
This function clears the error of the RC8.				
Packet TX	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			
	Argument	Description	Data Type	Value
		Binary		
	hController	The handle of the controller	VT_I4	0x0000002

		00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	bstrCommand	The 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	The 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		

4.6.2. Restarting the Slave mode

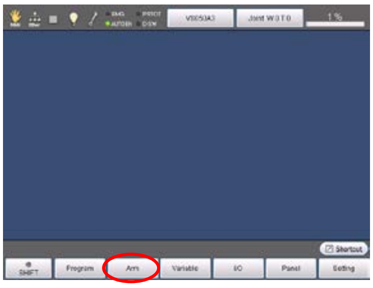
The Slave Mode is released once an error occurs. Therefore, you need to restart the Slave Mode. For details instruction of how to restart the Slave Mode, refer to “4.5The communication procedure of the Slave Mode”.

4.7. Setting of the command speed limit and acceleration limit

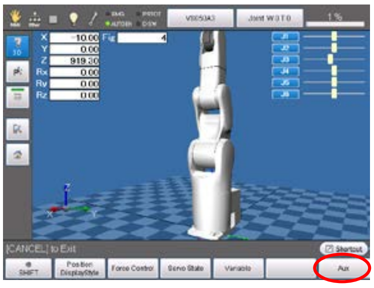
In the Slave Mode, you can set the command speed limit and acceleration limit. The limit is the threashold of the command speed and acceleration to reach the posture specified by Slave Move. If the command speed or acceleration exceeds the limit, "Excessive command speed in *-axis (0x8420405*)" or "Excessive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

To set the command speed limit and acceleration limit, use the 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 bigger than the command limit, and if you specify the (ExtSpeed), the limit is the value that multiplied the each axis limit by the rate of external speed (acceleration).



[F2] ⇒



[F6] ⇒

[F1] ⇒

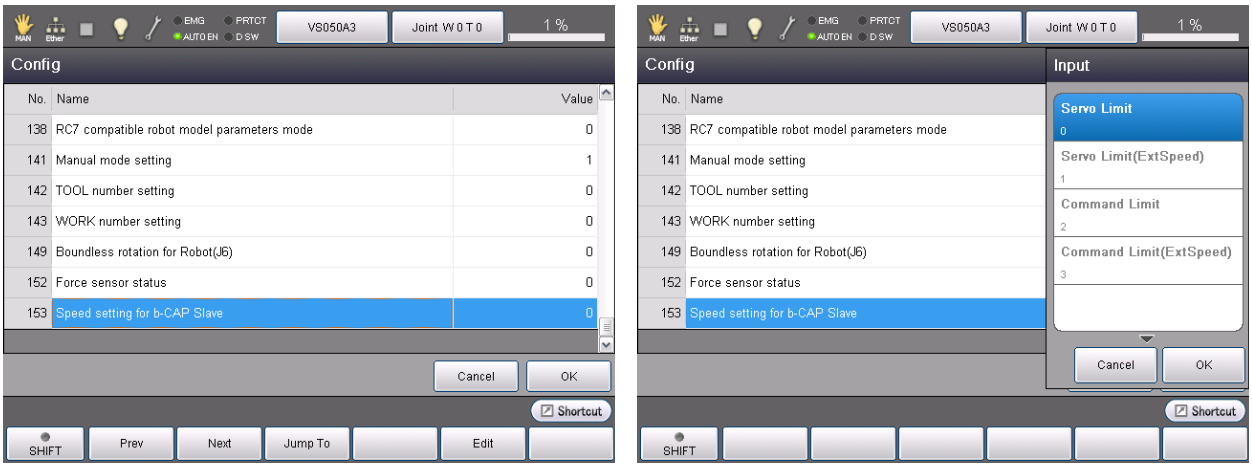


Figure 4-6 Speed setting for b-CAP Slave

4.8. Sample programs

Following is an example program, which executes Slave Mode, with using ANSI-C sample library.

This sample program executes the Slave Mode with Mode 0, then move the robot one cycle of the sine curve from the initial position. Because P1 coordinates is used as an initial position data, you need to set the robot coordinates in P1 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.cpp

```
#define _USE_MATH_DEFINES
#include <math.h>

#include "b-Cap.c"

#define SERVER_IP_ADDRESS          "192.168.0.1"
#define SERVER_PORT_NUM            5007

#define PERIOD           100
#define AMPLITUDE        15

#define E_BUF_FULL       0x83201483

int main(int argc, char* argv[])
{
        int iSockFD;
        u_long lhController;
        BCAP_HRESULT hr = BCAP_S_OK;

        /* Init and Start b-CAP */
        hr = bCap_Open(SERVER_IP_ADDRESS, SERVER_PORT_NUM, &iSockFD);

        /* Init socket */
        if FAILED(hr) return (hr);
```

```
        /* Start b-CAP service */
        hr = bCap_ServiceStart(iSockFD);

        /* Get controller handle */
        hr = bCap_ControllerConnect(iSockFD, "b-CAP", "caoProv.DENSO.VRC", SERVER_IP_ADDRESS, "",
&lhController);

        u_long lhRobot;
        long lResult;

        /* Get robot handle */
        hr = bCap_ControllerGetRobot(iSockFD, lhController, "Arm", "", &lhRobot);

        /* Get arm control authority */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Takearm", "", &lResult);

        /* Motor on */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Motor", "1", &lResult);

        /* Move to first pose */
        hr = bCap_RobotMove(iSockFD, lhRobot, 1L, "@E J1", "");

        /* Get current angle */
        double dJnt[8];
        hr = bCap_RobotExecute(iSockFD, lhRobot, "CurJnt", "", &dJnt);

        /* Start slave mode (Mode 0, J Type) */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "slvChangeMode", "2", &lResult);

        /* Execute slave move */
        BCAP_VARIANT vntPose, vntReturn;
        vntPose.Type = VT_R8 | VT_ARRAY;
        vntPose.Arrays = 8;
        for(int i = 0; i < PERIOD; i++)
        {
                vntPose.Value.DoubleArray[0] = dJnt[0] + i / 10.0;
                vntPose.Value.DoubleArray[1] = dJnt[1] + AMPLITUDE*sin(2*M_PI*i/PERIOD);
                for(int j = 2; j < 8; j++)
                {
                        vntPose.Value.DoubleArray[j] = dJnt[j];
                }

                hr = bCap_RobotExecute2(iSockFD, lhRobot, "slvMove", &vntPose, &vntReturn);

                /* if return code is not S_OK, then wait for 8 msec */
                if(hr != 0)
                {
                        Sleep(8);

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

        /* Stop robot */
        hr = bCap_RobotExecute2(iSockFD, lhRobot, "slvMove", &vntPose, &vntReturn);

        /* Stop slave mode */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "slvChangeMode", "0", &lResult);
```

```
        /* Motor off */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Motor", "0", &lResult);

        /* Release arm control authority */
        hr = bCap_RobotExecute(iSockFD, lhRobot, "Givearm", "", &lResult);

        /* Release robot handle */
        bCap_RobotRelease(iSockFD, lhRobot);

        /* Release controller handle */
        bCap_ControllerDisconnect(iSockFD, lhController);

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

        bCap_Close(iSockFD);

        return 0;
}
```

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	Meaning
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. $LAST_SEND_PACKET$ $LAST_RECEIVE_PACKET$

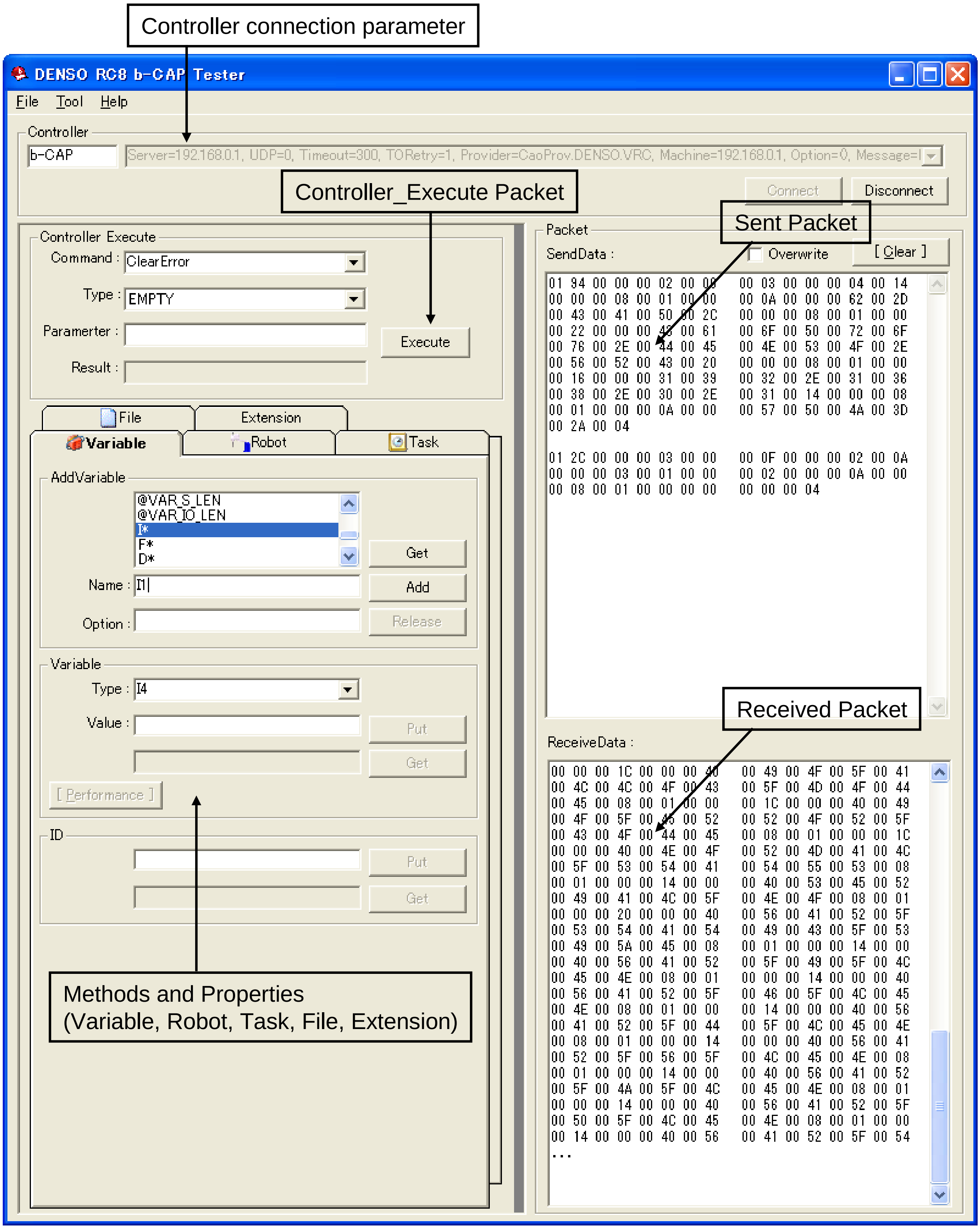


Figure 5-1 Description of functions of the b-CAP Tester

5.1. The Slave mode by the b-CAP Tester

To move the robot at Slave Mode with b-CAP Tester, following preparations are required.

- WINCAPS3 project files acquiring control logs.

  “slvMove” of b-CAP Tester initiates the robot motion by using command value of the servo log.

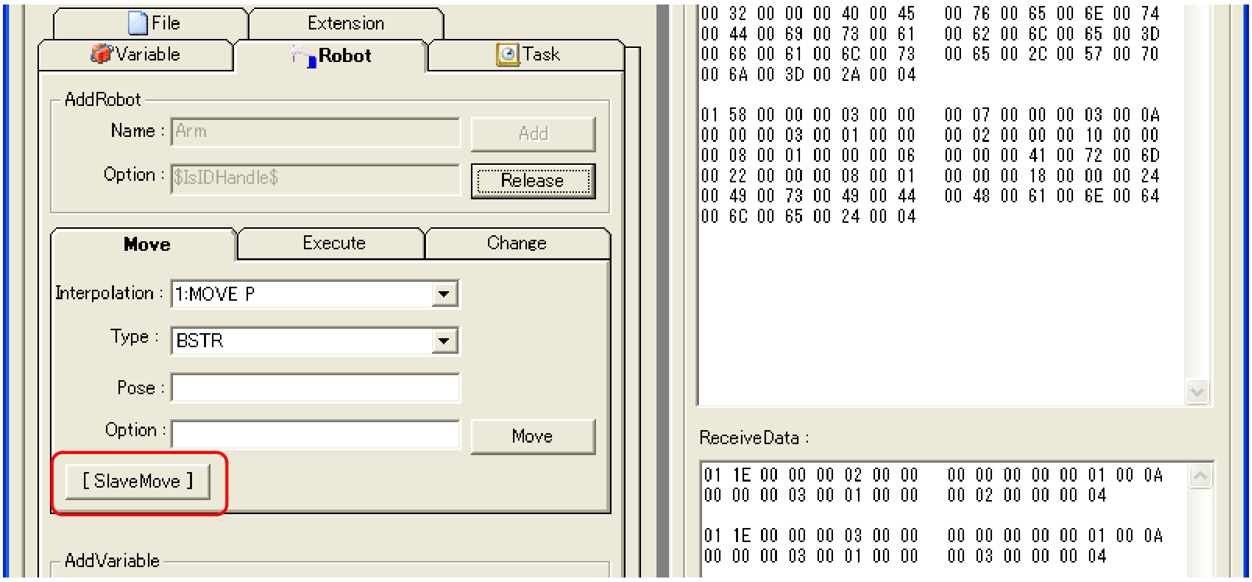
- Preparation of the controller

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

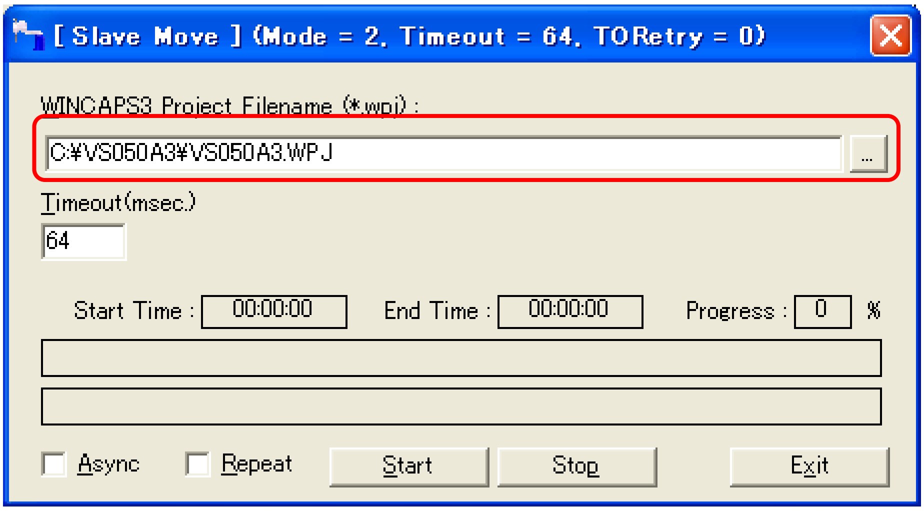
  For details, refer to “2.Setup of RC8”.

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..



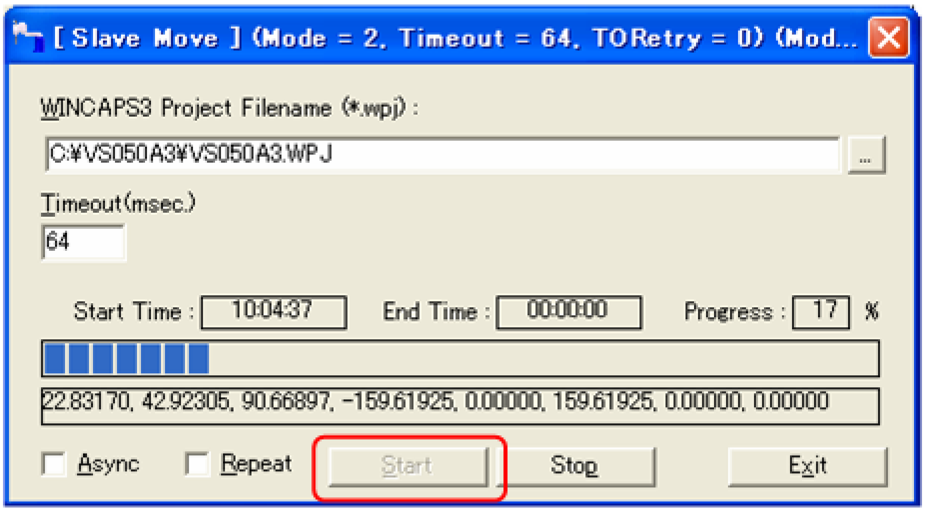
2. Specify WINCAP3 project which stores the control log.



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, controlling the motor is not executed. Please send a packet to controlling the motor before starting this process.

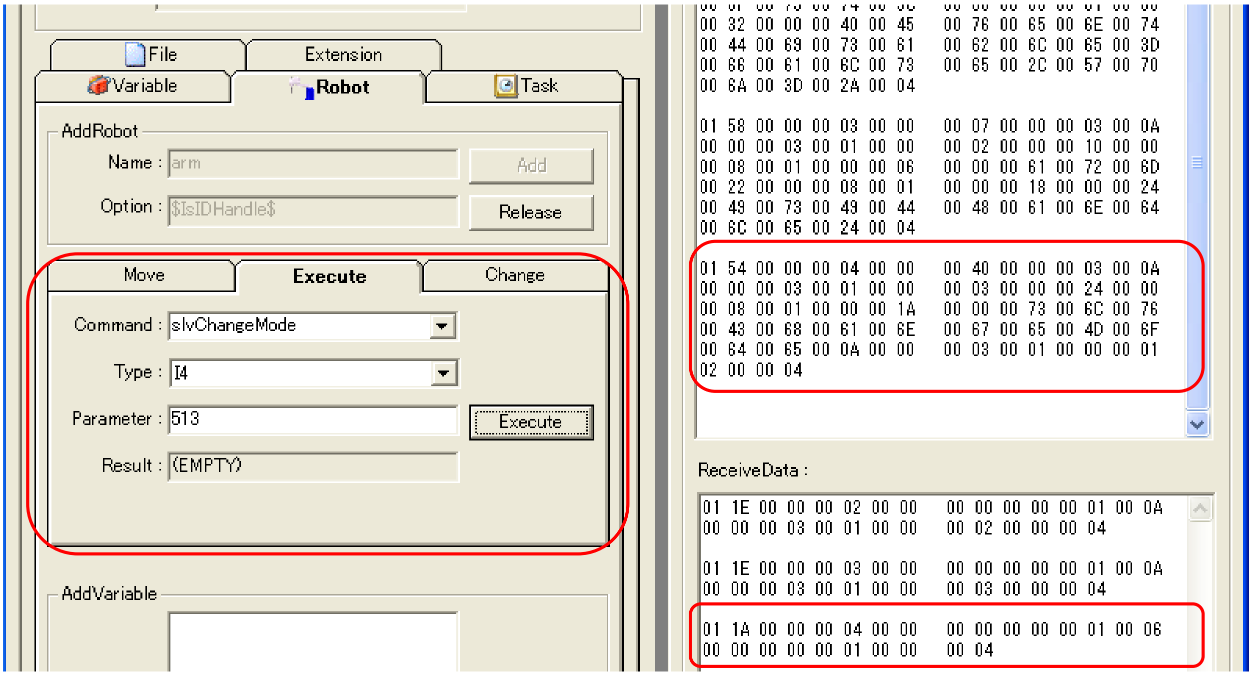
If the time-out happens while moving to initial position, please set the time-out parameter with larger value when connecting the controller.



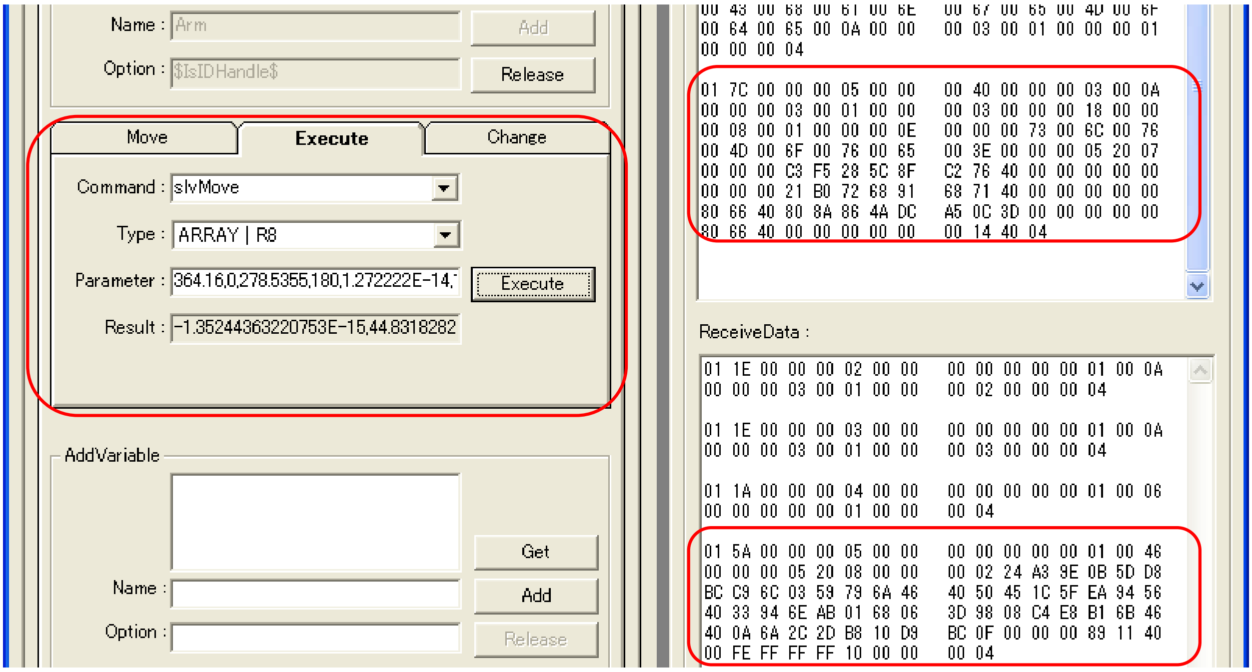
5.1.2. Confirming the packet of the Slave move in the b-CAP Tester

When the 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.

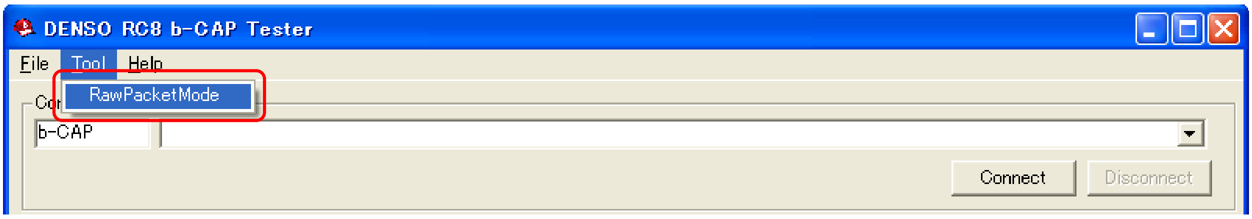


2. Execute “slvMove” command to move.



5.2. About row packet mode

In the row packet mode, you can control the controller by sending the packets which are manually created. This section describes how to use row packet mode.



5.2.1. Connecting to the controller

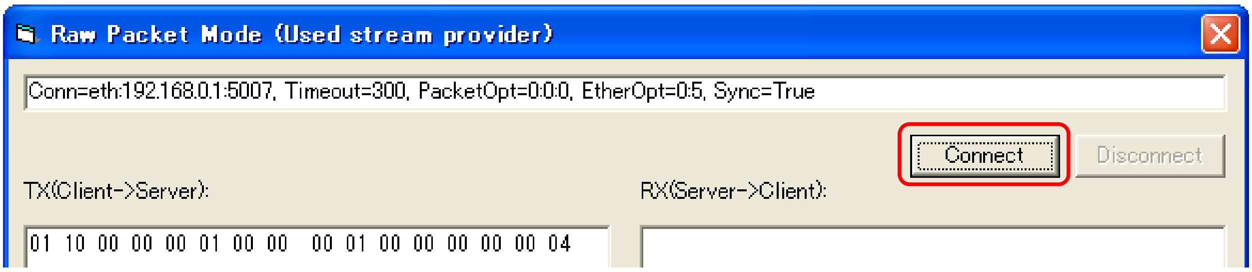
To connect to the controller in the row packet mode, set the parameters described in Table 5-2 and click [Connect] button. For details each of the parameters, refer to “Stream Provider Guide.”

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

Table 5-2 Connection parameters of row paket mode

Option	Meaning
Conn=eth:[<IP Address>[:<Port No>]]	Specify the IP address of controller to be connected
Timeout [=<Timeout>]	Timeout period when sending and receiving it. (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 and ‘1' - ENQ(0x05) <Term>: Terminator specification. ‘0'-CR(0x0D),‘1'-LF(0x0A),‘2'-CR+LF(0x0D0A) If you enter b-CAP packets directly, specify 0:0:0.
EtherOpt =[<Mode>[:<ConnMax>]]	<Mode>: Character string conversion.

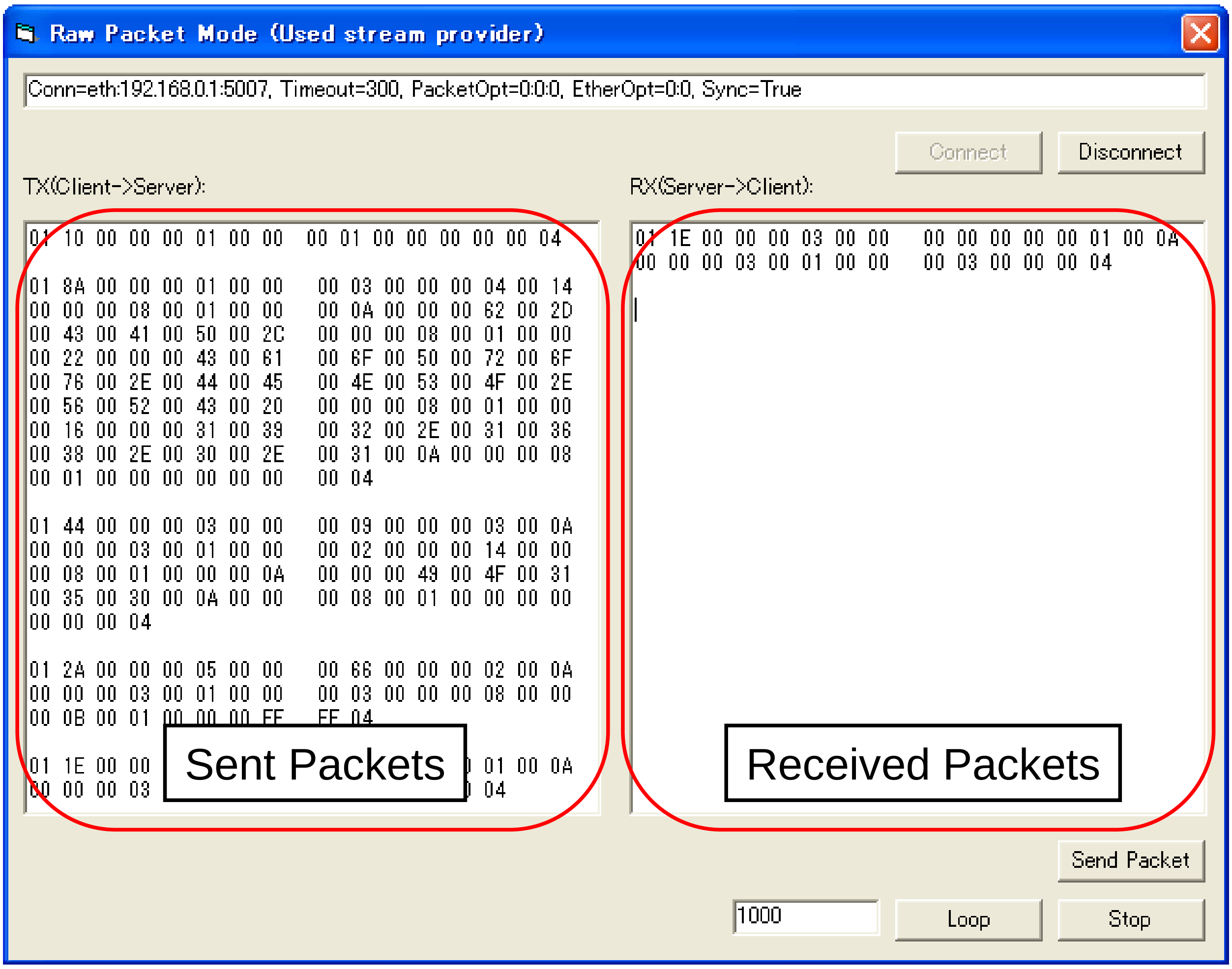
	‘0' . -TCP client mode ‘1'-TCP server - mode ‘2' . -UDP client mode ‘3'-UDP server - mode <ConnMax>: Number of maximum clients at TCP server mode. (default: 5) In the row packet mode, specify 0:0 or 2:0.
Sync=TRUE	The synchronous mode is set. In the row packet mode, specify TRUE.



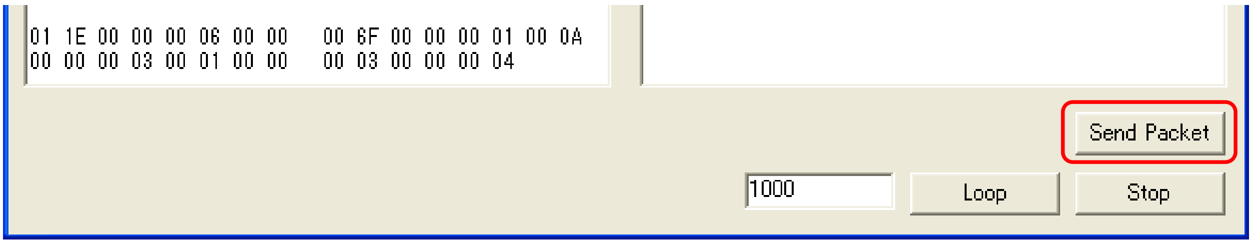
5.2.2. Sending and Receiving the b-CAP packets

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

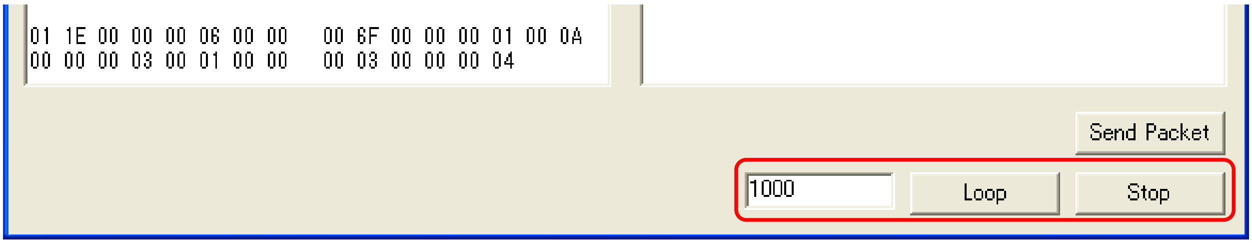
The received packet is shown in the right side of the text area. If you write and send two or more packets, the displayed packet in the right side of the text area will be the reply of the last packet.



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 more than one time, specify the time of sendings, and click [Loop] button. To cancel the sendings, click [Stop] button.



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

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

<Data type>, <Data type>

<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

When the data type is array, describe VT_ARRAY and the logical add of the data type.
For data columns, describe data by the character strings. The description of the array data is delimited by “,” (comma).

Figure 5-2 shows the example of a Pose of “Robot_Move”.
In “Robot_Move”, VT_ARRAY | VT_VARIANT can be designated as a Pose.

In the first array, “VT_R8 | VT_ARRAY (8197)” is used in order to describe 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).

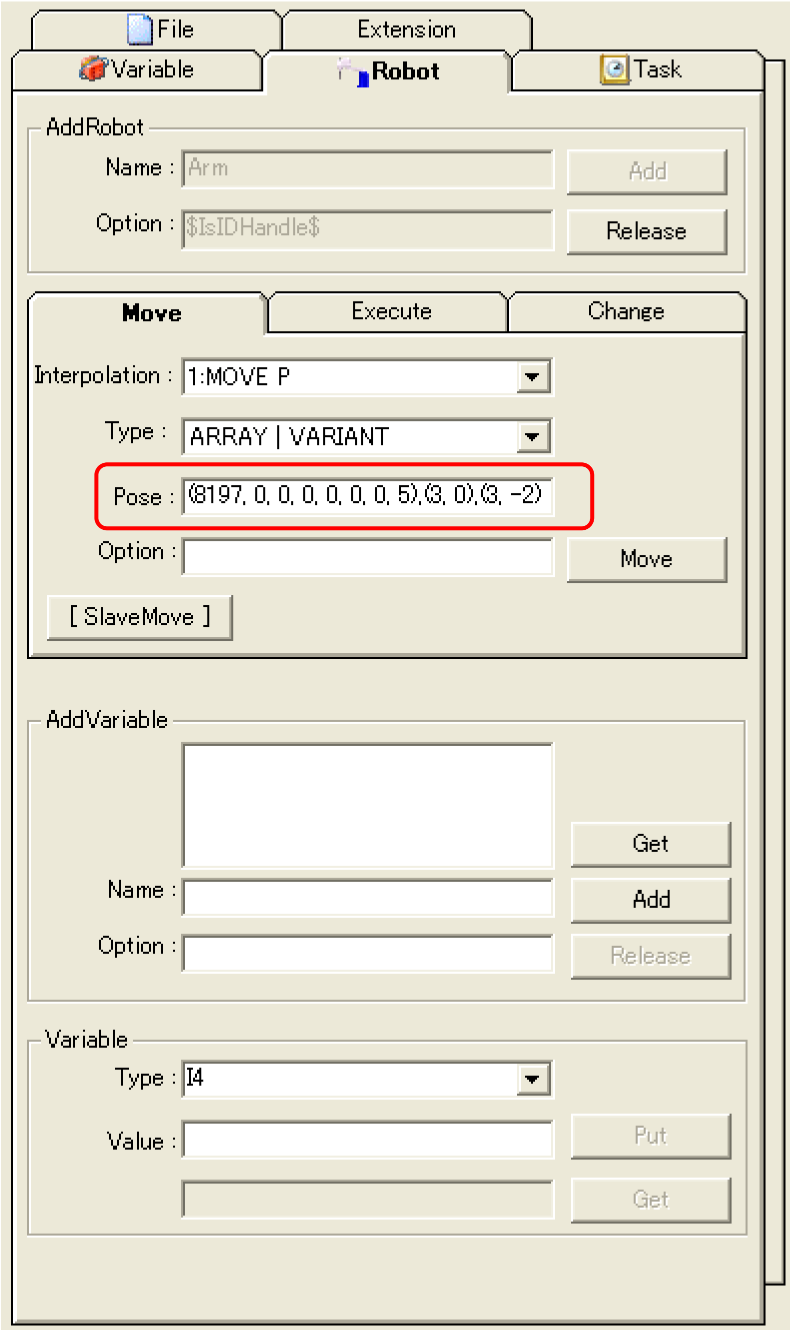


Figure 5-2 Example of the Pose in “Robot_Move”

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	Beginning of server service
2	Service_Stop	CCaoWorkspaces::Remove	Stop of server service
3	Controller_Connect		Connection with controller
4	Controller_Disconnect		Cutting with controller
5	Controller_GetExtension	CCaoController::AddExtension	The controller's extension board acquisition
6	Controller_GetFile	CCaoController::AddFile	The controller's file acquisition
7	Controller_GetRobot	CCaoController::AddRobot	The controller's robot acquisition
8	Controller_GetTask	CCaoController::AddTask	The controller's task acquisition
9	Controller_GetVariable	CCaoController::AddVariable	The controller's variable acquisition
10	Controller_GetCommand	CCaoController::AddCommand	The controller's command acquisition
11	Controller_GetExtensionNames	CCaoController::get_ExtensionNames	The controller's extension board name list acquisition
12	Controller_GetFileNames	CCaoController::get_FileNames	The controller's file name list acquisition
13	Controller_GetRobotNames	CCaoController::get_RobotNames	The controller's robot name list acquisition

14	Controller_GetTaskNames	CCaoController::get_TaskNames	The controller's task name list acquisition
15	Controller_GetVariableNames	CCaoController::get_VariableNames	The controller's variable identifier list acquisition
16	Controller_GetCommandNames	CCaoController::get_CommandNames	The controller's command name list acquisition
17	Controller_Execute	CCaoController::Execute	Execution of controller's enhancing function
18	Controller_GetMessage	CCaoController::AddMessage	The controller's event message acquisition
19	Controller_GetAttribute	CCaoController::get_Attribute	The controller's attribute value acquisition
20	Controller_GetHelp	CCaoController::get_Help	The controller's help character string acquisition
21	Controller_GetName	CCaoController::get_Name	The controller's name acquisition
22	Controller_GetTag	CCaoController::get_Tag	The controller's tag information acquisition
23	Controller_PutTag	CCaoController::put_Tag	The controller's tag information setting
24	Controller_GetID	CCaoController::get_ID	The controller's ID acquisition
25	Controller_PutID	CCaoController::put_ID	The controller's ID setting
26	Extension_GetVariable	CCaoExtension::AddVariable	Acquisition of variable of extension board
27	Extension_GetVariableNames	CCaoExtension::get_VariableNames	Acquisition of list of variable identifier of

			extension board
28	Extension_Execute	CCaoExtension::Execute	Execution of enhancing function of extension board
29	Extension_GetAttribute	CCaoExtension::get_Attribute	Attribute value acquisition of extension board
30	Extension_GetHelp	CCaoExtension::get_Help	Acquisition of help character string of extension board
31	Extension_GetName	CCaoExtension::get_Name	Acquisition of name of extension board
32	Extension_GetTag	CCaoExtension::get_Tag	Acquisition of tag information on extension board
33	Extension_PutTag	CCaoExtension::put_Tag	Setting of tag information on extension board
34	Extension_GetID	CCaoExtension::get_ID	ID acquisition of extension board
35	Extension_PutID	CCaoExtension::put_ID	ID setting of extension board
36	Extension_Release	CCaoExtension::Release	Liberating of extension board
37	File_GetFile	CCaoFile::AddFile	Another file acquisition of file
38	File_GetVariable	CCaoFile::AddVariable	Acquisition of variable of file
39	File_GetFileNames	CCaoFile::get_FileNames	Acquisition of list of another file name of file
40	File_GetVariableNames	CCaoFile::get_VariableNames	Acquisition of list of variable identifier of file
41	File_Execute	CCaoFile::Execute	Execution of enhancing function

			of file
42	File_Copy	CCaoFile::Copy	Copy of file
43	File_Delete	CCaoFile::Delete	Deletion of file
44	File_Move	CCaoFile::Move	Movement of file
45	File_Run	CCaoFile::Run	Execution of file
46	File_GetDateCreated	CCaoFile::get_DateCreated	Acquisition at the date of file
47	File_GetDateLastAccessed	CCaoFile::get_DateLastAccessed	Acquisition at the final access date of file
48	File_GetDateLastModified	CCaoFile::get_DateLastModified	Acquisition at last updated date and time of file
49	File_GetPath	CCaoFile::get_Path	Passing acquisition of file
50	File_GetSize	CCaoFile::get_Size	Size acquisition of file
51	File_GetType	CCaoFile::get_Type	File type acquisition of file
52	File_GetValue	CCaoFile::get_Value	Acquisition of content of file
53	File_PutValue	CCaoFile::put_Value	Setting of content of file
54	File_GetAttribute	CCaoFile::get_Attribute	Attribute acquisition of file
55	File_GetHelp	CCaoFile::get_Help	Acquisition of help character string of file
56	File_GetName	CCaoFile::get_Name	Acquisition of name of file
57	File_GetTag	CCaoFile::get_Tag	Acquisition of tag information on file
58	File_PutTag	CCaoFile::put_Tag	Setting of tag information on file
59	File_GetID	CCaoFile::get_ID	ID acquisition of file
60	File_PutID	CCaoFile::put_ID	ID setting of file

61	File_Release	CCaoFile::Release	Liberating of file
62	Robot_GetVariable	CCaoRobot::AddVariable	Acquisition of variable of robot
63	Robot_GetVariableNames	CCaoRobot::get_VariableNames	Acquisition of list of variable identifier of robot
64	Robot_Execute	CCaoRobot::Execute	Execution of enhancing function of robot
65	Robot_Accelerate	CCaoRobot::Accelerate	Execution of ACCEL sentence of robot
66	Robot_Change	CCaoRobot::Change	Execution of CHANGE sentence of robot
67	Robot_Chuck	CCaoRobot::Chuck	Execution of GRASP sentence of robot
68	Robot_Drive	CCaoRobot::Drive	Execution of DRIVE sentence of robot
69	Robot_GoHome	CCaoRobot::GoHome	Execution of GOHOME sentence of robot
70	Robot_Halt	CCaoRobot::Halt	Execution of HALT sentence of robot
71	Robot_Hold	CCaoRobot::Hold	Execution of HOLD sentence of robot
72	Robot_Move	CCaoRobot::Move	Execution of MOVE sentence of robot
73	Robot_Rotate	CCaoRobot::Rotate	Execution of ROTATE sentence of robot
74	Robot_Speed	CCaoRobot::Speed	Execution of SPEED/JSPEED sentence of robot

75	Robot_Unchuck	CCaoRobot::Unchuck	Execution of REELASE sentence of robot
76	Robot_Unhold	CCaoRobot::Unhold	Release of HOLD sentence of robot
77	Robot_GetAttribute	CCaoRobot::get_Attribute	Attribute value acquisition of robot
78	Robot_GetHelp	CCaoRobot::get_Help	Acquisition of help character string of robot
79	Robot_GetName	CCaoRobot::get_Name	Acquisition of name of robot
80	Robot_GetTag	CCaoRobot::get_Tag	Acquisition of tag information on robot
81	Robot_PutTag	CCaoRobot::put_Tag	Setting of tag information on robot
82	Robot_GetID	CCaoRobot::get_ID	ID acquisition of robot
83	Robot_PutID	CCaoRobot::put_ID	ID setting of robot
84	Robot_Release	CCaoRobot::Release	Liberating of robot
85	Task_GetVariable	CCaoTask::AddVariable	Acquisition of variable of task
86	Task_GetVariableNames	CCaoTask::get_VariableNames	Acquisition of list of variable identifier of task
87	Task_Execute	CCaoTask::Execute	Execution of enhancing function of task
88	Task_Start	CCaoTask::Start	Beginning of task
89	Task_Stop	CCaoTask::Stop	Stop of task
90	Task_Delete	CCaoTask::Delete	Deletion of task
91	Task_GetFileName	CCaoTask::get_FileName	Former file name of task
92	Task_GetAttribute	CCaoTask::get_Attribute	Attribute acquisition of task
93	Task_GetHelp	CCaoTask::get_Help	Acquisition of help

			character string of task
94	Task_GetName	CCaoTask::get_Name	Acquisition of name of task
95	Task_GetTag	CCaoTask::get_Tag	Acquisition of tag information on task
96	Task_PutTag	CCaoTask::put_Tag	Setting of tag information on task
97	Task_GetID	CCaoTask::get_ID	ID acquisition of task
98	Task_PutID	CCaoTask::put_ID	ID setting of task
99	Task_Release	CCaoTask::Release	Liberating of task
100	Variable_GetDateTime	CCaoVariable::get_DateTime	Stamp acquisition of time of variable
101	Variable_GetValue	CCaoVariable::get_Value	Value acquisition of variable
102	Variable_PutValue	CCaoVariable::put_Value	Value setting of variable
103	Variable_GetAttribute	CCaoVariable::get_Attribute	Attribute value acquisition of variable
104	Variable_GetHelp	CCaoVariable::get_Help	Acquisition of help character string of variable
105	Variable_GetName	CCaoVariable::get_Name	Acquisition of name of variable
106	Variable_GetTag	CCaoVariable::get_Tag	Acquisition of tag information on variable
107	Variable_PutTag	CCaoVariable::put_Tag	Setting of tag information on variable
108	Variable_GetID	CCaoVariable::get_ID	ID acquisition of variable
109	Variable_PutID	CCaoVariable::put_ID	ID setting of variable

110	Variable_GetMicrosecond	CCaoVariable::get_Microsecond	Time stamp (millisecond) acquisition of variable
111	Variable_Release	CCaoVariable::Release	Liberating of variable
112	Command_Execute	CCaoCommand::Execute	Execution of command
113	Command_Cancel	CCaoCommand::Cancel	Cancellation of command
114	Command_GetTimeout	CCaoCommand::get_Timeout	Acquisition at time-out time of command
115	Command_PutTimeout	CCaoCommand::put_Timeout	Setting at time-out time of command
116	Command_GetState	CCaoCommand::get_State	State acquisition of command
117	Command_GetParameters	CCaoCommand::get_Parameters	Acquisition of parameter of command
118	Command_PutParameters	CCaoCommand::put_Parameters	Setting of parameter of command
119	Command_GetResult	CCaoCommand::get_Result	Execution result acquisition of command
120	Command_GetAttribute	CCaoCommand::get_Attribute	Attribute value acquisition of command
121	Command_GetHelp	CCaoCommand::get_Help	Acquisition of help character string of command
122	Command_GetName	CCaoCommand::get_Name	Acquisition of name of command
123	Command_GetTag	CCaoCommand::get_Tag	Acquisition of tag information on command

124	Command_PutTag	CCaoCommand::put_Tag	Setting of tag information on command
125	Command_GetID	CCaoCommand::get_ID	ID acquisition of command
126	Command_PutID	CCaoCommand::put_ID	ID setting of command
127	Command_Release	CCaoCommand::Release	Liberating of command
128	Message_Reply	CCaoMessage::Reply	Response of event message
129	Message_Clear	CCaoMessage::Clear	Clearness of event message
130	Message_GetDateTime	CCaoMessage::get_DateTime	Stamp acquisition of time of event message
131	Message_GetDescription	CCaoMessage::get_Description	Acquisition of explanation of event message
132	Message_GetDestination	CCaoMessage::get_Destination	Destination acquisition of event message
133	Message_GetNumber	CCaoMessage::get_Number	Acquisition of message number of event message
134	Message_GetSerialNumber	CCaoMessage::get_SerialNumber	Acquisition of serial number of event message
135	Message_GetSource	CCaoMessage::get_Source	Former transmission acquisition of event message
136	Message_GetValue	CCaoMessage::get_Value	Value acquisition of event message
137	Message_Release	CCaoMessage::Release	Liberating of event message