DENSO
b-CAP Communication Specifications
for RC7

Version 1.1.12

December 07, 2011

【Remarks】

【Revision history】

Date	E/c level	Content
2006-08-02	1.0.0	First release
2007-10-28	1.0.1	Add section 5 "b-CAP Communication function" for DENSO RC7M/J
2007-12-04	1.0.2	(1) In section3, Make revision "SOH = 0x01" (2) In section 3,Add Return code "0x80010004 E_ROBOTISBUSY (3) In section 5, Add "Contoller_Execute" method for Controller Object (4) In section 5, Add "Robot_Change" for RobotObject Tool=n Work=n (5) Add System Variables in section 5 @EXTSPEED @EXTACCEL @EXTDECEL (6) Add Note about reserved variable I[10] in section 6.
2008-01-14	1.0.3	(1) In section 5, The explanation of iMode of Task_start is modified. (2) In section 5, The explanation of iMode of Task_Stop is modified. (3) In section 5, a sample packet of Controller_Execute is modified. (4) In section 5,Add so many Sample packets and explanations.
2008-01-31	1.0.4	(1) In section 5,Add explanations of controller variables for the function Controller_GetVariable: "TOOL" – Tool settings "WORK" – Work settings "_ITP" – ITP Configurations "_PAC" – PAC Configurations "_DIO" – DIO Configurations "_ARM" – ARM Configurations "_SRV" – SRV Configurations "_SPD" – SPD Configurations "_VIS" – VIS Configurations "_COM" – COM Configurations
2008-02-25	1.0.5	(1) In section 5, Add controller variables for the function Controller_GetVariable: "@AUTO_ENABLE" – Auto Enable signal

		"@PROTECTIVE_STOP" – Protective stop "@DEADMAN_SW" – Deadman Switch (2) In section 5, Modify the type of variable "@EMERGENCY_STOP" from VT_BOOL to VT_I2.
2008-05-05	1.0.6	ROM Version : 2.622M 05/13/2008 b-CAP [VERTICAL] 2.623M 05/13/2008 b-CAP [SCARA] (1) Add method and properties of Controller object. - Controller_Execute("SuspendAll") - Controller_Execute("ResetAll") - Controller_GetTaskNames() - Robot_Execute("MotionComp") (2) Add format of the move command of Controller object. - Move J(J1,J2,J3,J4,J5,J6) - Move T(X,Y,Z,Ox,Oy,Oz,Ax,Ay,Az,Fig) - Note: These functions are already supported in the robot controller version V2.801M and the later.
2008-06-12	1.0.7	ROM Version : 2.623M 06/11/2008 b-CAP [VERTICAL] 2.624M 06/11/2008 b-CAP [SCARA] (1) This firmware update solve a pendant problem: In the teachcheck mode, While step run is executing by teaching pendant, Releasing OK key may not work exceptionally. Note: These functions are already supported in the robot controller version V2.801M and the later.
2009-03-02	1.1.0	ROM Version: 2.8
2009-04-01	1.1.1	Parameter of P2J, J2P,J2T,P2T,T2J,T2P,TINV,NORMTRN is changed into Variant.
2009-05-29	1.1.2	Editorial change.
2009-06-03	1.1.3	(1) The data type of parameters are modified. Commands are below. P2J,J2P,J2T,P2T,T2J,T2P,TINV,NORMTRN This modify is valid for ROM version V3.000 and the later. (2) A sample packet of UserExtension is modified.

2009-07-31	1.1.4	(1) Add parameter of Controller_Execute, “startLog”, “stopLog”, “clearLog” (2) Add parameter of Robot_GetVariable, “@TYPE” This modify is valid for ROM version V3.000 and the later.
2009-12-02	1.1.5	(1) Add type that can be used by argument of “UserExtension” command. This modify is valid for ROM version V3.000 and the later. (2) Add “slvChange”,”slvMove”,”slvGetMode” method of Robot_Execute. This modify is valid for ROM version V3.000 and the later. (3) Add sample of “slvMove”.
2010-02-19	1.1.6	(1) Add new variables for Robot_GetVariable, “@HIGH_CURRENT_POSITION”, “@HIGH_CURRENT_ANGLE”, “@HIGH_CURRENT_TRANS” These are enabled in the RC7 version V3.000 and the later. (2) Add Communication sample 5 for Variable_GetValue..
2010-05-24	1.1.7	Add mark of version supported.
2010-11-17	1.1.8	Correct Execute command name.
2010-11-24	1.1.9	Add "GetSrvData", "SetSrvData" for Robot_Execute. Add variable length area for b-CAP packet structure.
2011-09-01	1.1.10	Add a note about the license of the b-CAP slave function.
2011-11-25	1.1.11	Correct misprint.
2011-12-07	1.1.12	In section 2, The explanation of Treatment of special I/O port is modified.

Contents

1. Introduction

This specification provides communication protocol of b-CAP.

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.

b-CAP is implemented as TCP stream communication. This is because packet of b-CAP does not have check codes and error-free protocols are needed in the lower layer.

In the section "b-CAP communication function", the function and message specifications for accessing to RC7M Robot controller are described. Various functions of RC7M/J controller are available by sending or receiving messages that corresponds to the function specification via TCP stream.

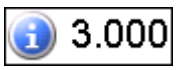
b-CAP is available with version 2.8 and the later of RC7M/J controller.

The b-CAP Slave function is a optional function of RC7M/J, therefore it is required a license key in each RC7M/J controller.

The detail of the b-CAP Slave function is described in the other document, 'How to use the Slave mode/b-CAP'.

The dependency of the NetwoRC controller's model and version is described in the next table used as Sign in this document.

Figure 1-1 NetwoRC controller's model and version

Controller		Sign	Explanation
Model	Version		
RC7J, RC7M	3.000 or more		If the controller version is 3.000 or more, it is valid.
-	-		It is reserved in preparation for mounting in the future.

2. Setup

2.1. Emergency stop device position

Emergency stop device shall be in a position where they can be reached easily to stop the robot immediately. The emergency stop device shall be in accordance with IEC 60204-1:2005, 10.7 and ISO 13850.

2.2. Controller setup

A robot controller needs to be setup before using b-CAP. A teaching pendant or a mini pendant is needed for setting up the robot controller. Following is the procedure to setup the controller.

2.2.1. Setup using a teaching pendant

Setup a robot controller with a teaching pendant according to the following procedure.

(1) Set the robot controller to the manual mode.

(2) Activate ORiN option. Select “Option” => Input password “1214” in the function expansion menu to activate “ORiN” option.[1]

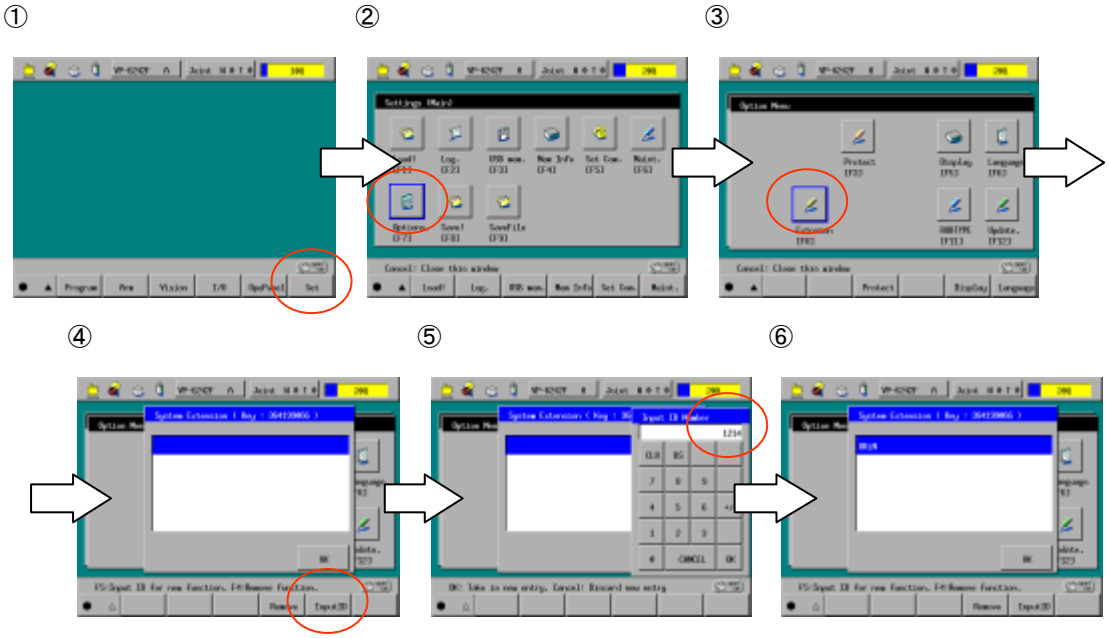


Figure 2-1 ORiN option activation

[1] By setting “ORiN” option, b-CAP client and ORiN applications can freely access the variables and I/Os. Access the variables and I/Os after fully understanding the state and the content of the robot controller program etc. Especially, the changing the variables and I/Os might give the critical effect to the movement of the robot and the program.
In the internal automatic mode, when “ORiN” option is activated, the robot program stops if an error more than level 3 occurs. However, in external automatic mode, the robot program stops if an error more than error level 2 occurs. Therefore, when a robot is in external automatic mode, please be careful not to perform wrong operation or not to transmit a wrong command.

(3) Set controller's communication permission. When you use Ethernet, Set "Read/Write" in the menu "Communications setting menu" => "Permit."

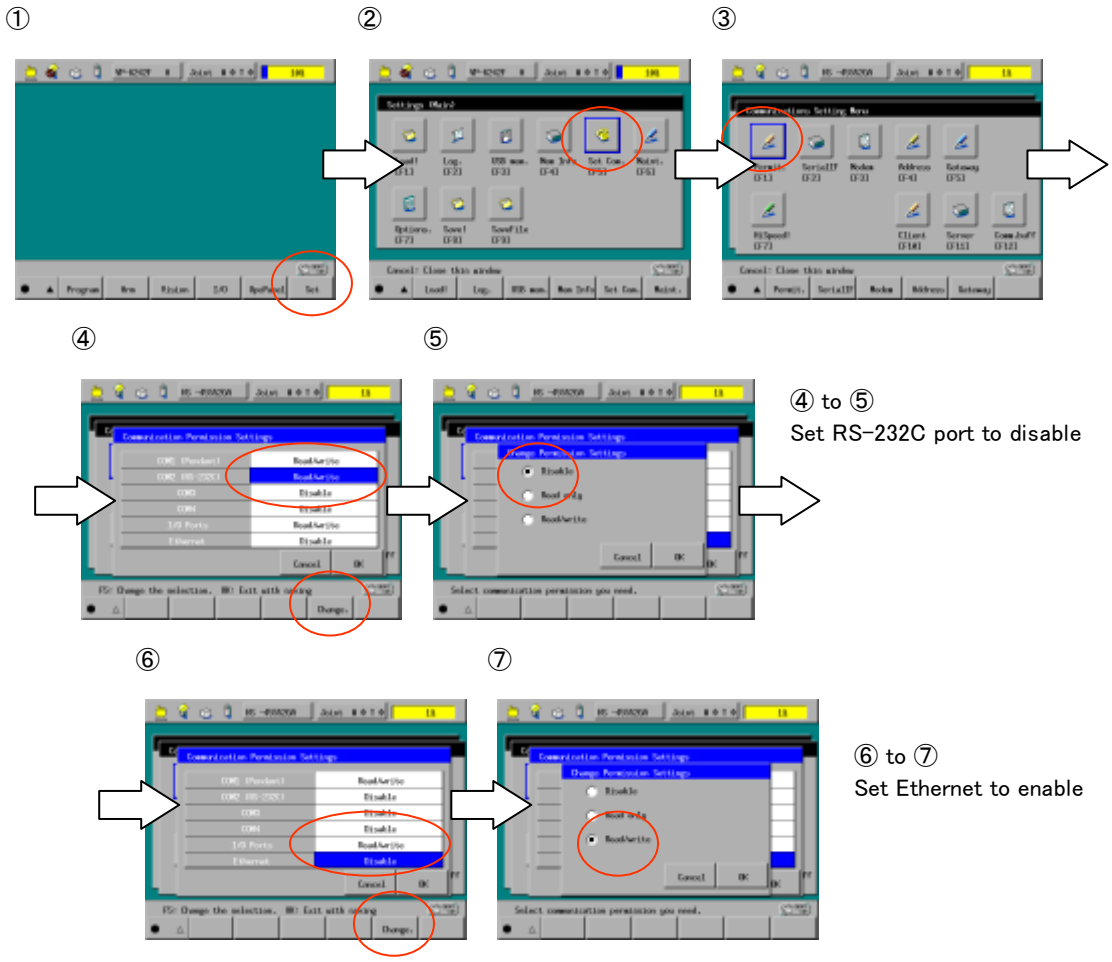


Figure 2-2 Setting of communication permission

(4) To turn ON/OFF the motor of the robot, or to start programs from an b-CAP application, it is necessary to set controller's executable token. Set the executable token to "Ethernet" by setting the "Communications setting menu" => "Ext.Run", and set IP address of client PC by "F4:IP set" menu afterwards when you use Ethernet as the connection method.

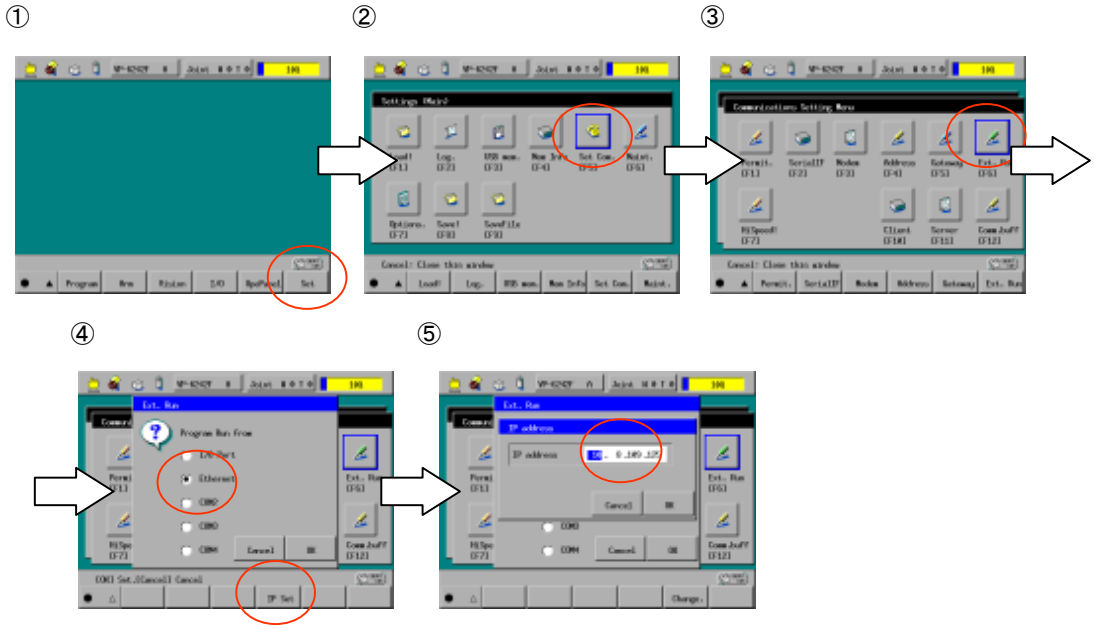


Figure 2-3 Settings of Executable token

2.2.2. Setup using the mini pendant

Setup a controller with a mini pendant according to the following procedure.

(1) Set a robot controller to the manual mode.

(2) Activate ORiN option. [Aux Function] => [Extension] => [Extension] => Input “1214” with Add and make “ORiN” effective. [1]

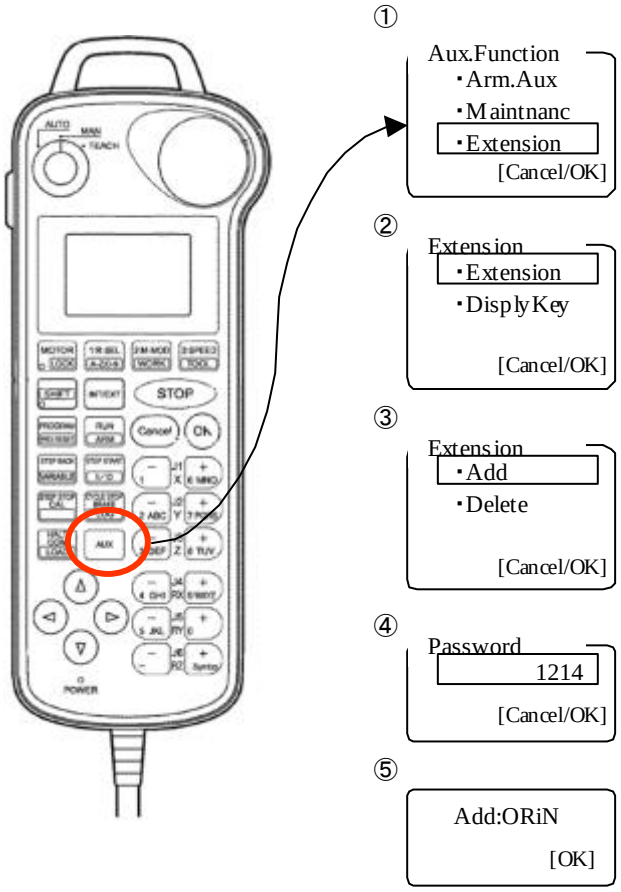


Figure 2-4 ORiN option activation

(3) Set robot controller’s communication permission. When you use Ethernet, go to COM Setting of mini pendant => Set R/W to Ethernet with Permit.

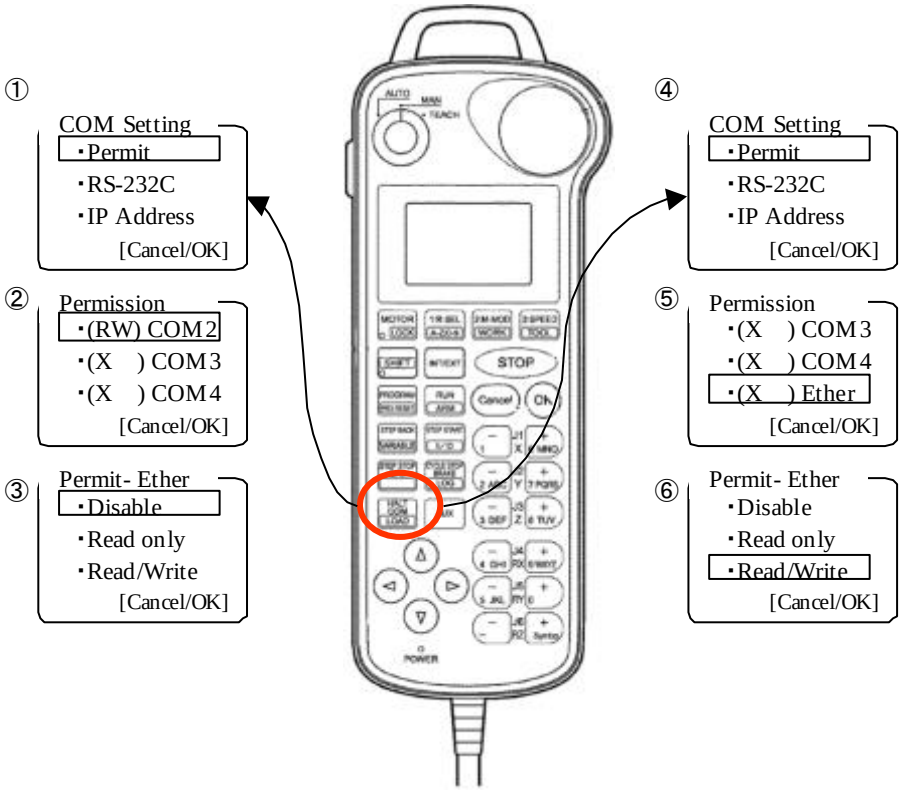


Figure 2-5 Setting of communication permission

(4) To turn ON/OFF the motor of the robot, or to start programs from an b-CAP application, it is necessary to set controller's executable token. When Ethernet is used for communication, go to COM menu of mini pendant and set the executable token to Ethernet in [Ext Run], and also set IP address of client PC in [Client IP] menu.

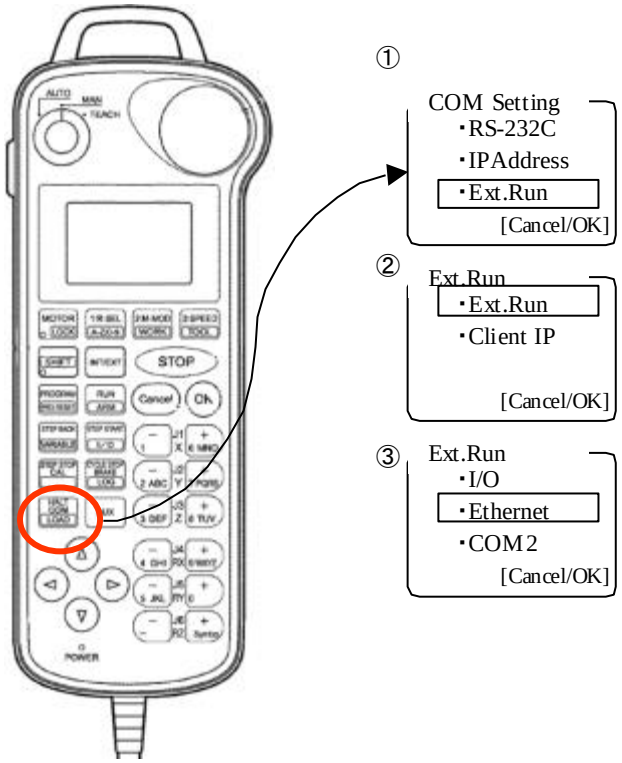


Figure 2-6 Setting of Executable token

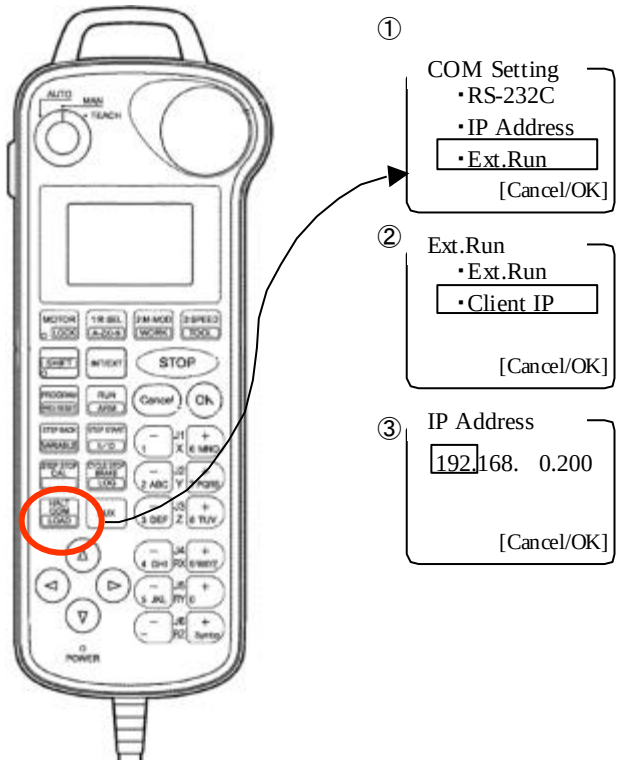


Figure 2-7 Input of IP address of client PC

2.2.3. Preliminary note

b-CAP uses TCP port 5007 of the RC7M/J controller.

If the server communication function of the controller (Access in TP: [F6 Set]—[F5 Set Com.]—[F11 Server]) use the same port, then b-CAP can not work.

2.3. Treatment of special I/O port

DENSO robot controller has a lot of system input signals.

To operate robot from PCs connected to the robot controller using b-CAP, “Step stop (All tasks) signal” and “Instantaneous stop signal” need to be set to enable robot program execution.

Connector assignment and pin assignment of “Step stop (All tasks) signal” and “Instantaneous stop signal” are different depending on the robot controller configuration and I/O assignment. Please confirm the robot controller configuration to make correct I/O treatment to enable robot program execution.

[Note 1] The I/O treatment is not necessary for RC5, because these versions of software does not support running robot program using b-CAP.

[Note 2] The I/O treatment is not necessary, if b-CAP is used only to access variables or files, and if b-CAP is NOT used to control robot motion or robot program (PAC).

2.3.1. I/O treatment for standard configuration controller (without I/O extension board)

Please close Mini I/O general purpose / system I/O connector CN5 – terminal No.11.

By closing the signal, Step stop (All tasks) signal (port number 0) becomes ON, and robot and program execution is enabled.

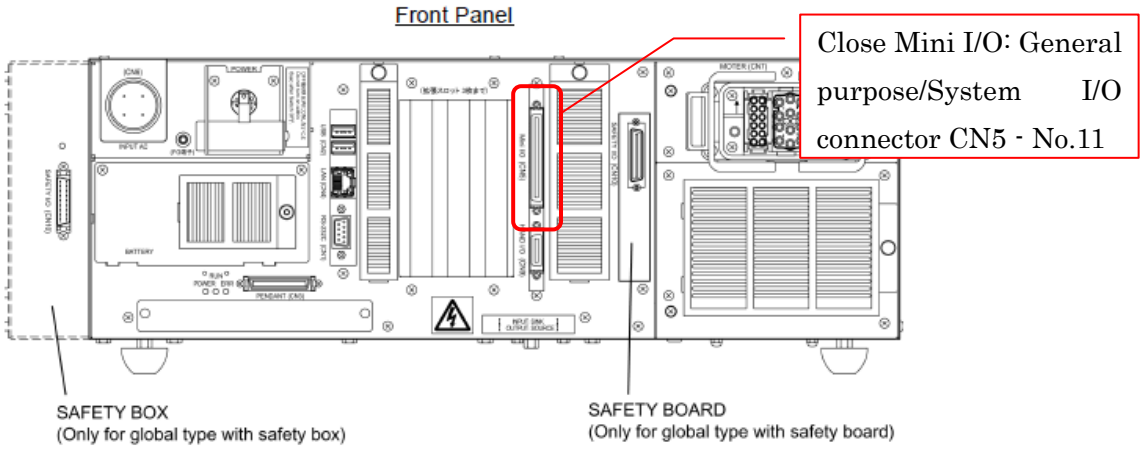


Figure 2-8 Step stop (All tasks) treatment

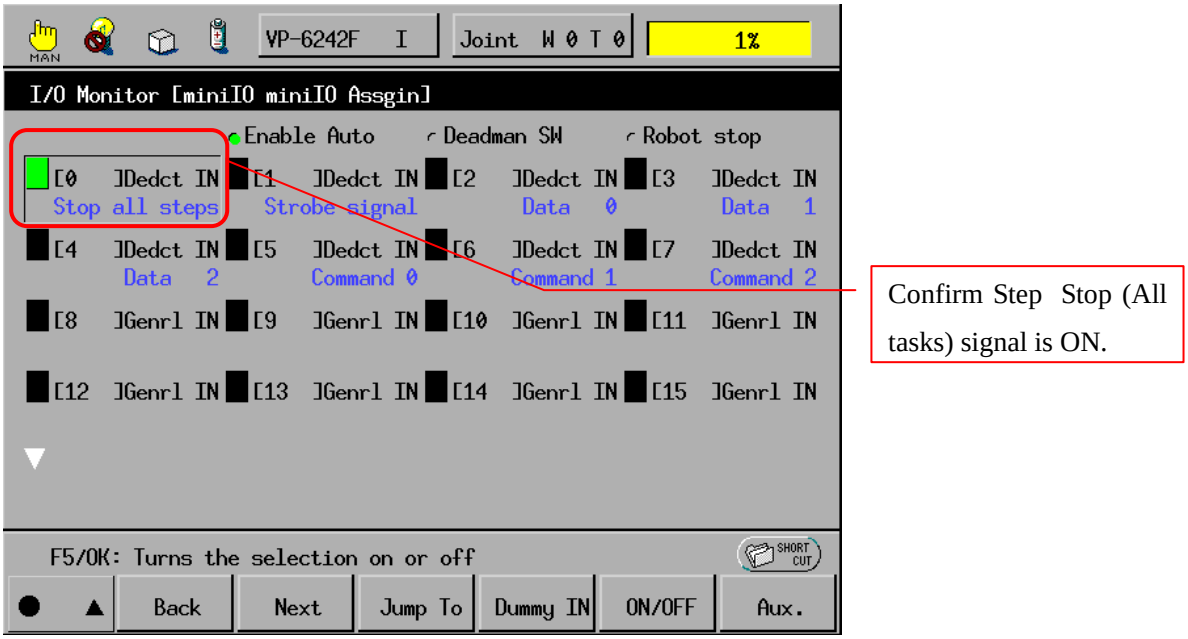


Figure 2-9 Signal confirmation after I/O treatment

This type of I/O does not have Instantaneous stop signal, so no treatment is necessary.

2.3.1.1. Mini I/O All general option

Mini I/O All-general option, available on Version 2.90 or later, is for robot system that does not require special I/O port assigned to mini I/O. By activating the option, all mini I/O ports are assigned as general I/O, and step stop signal wiring becomes not necessary.

To activate the option, setup a robot controller with a teach pendant according to the following procedure.

(5) Set a robot controller to the manual mode.

(6) To activate Mini I/O All-general option, select “Option” => “Function expansion” menu and input password “**6319**”.

(7) Turn off and restart a robot controller.

[Note 1] By activating mini I/O all-general option, a special I/O input assignment of step stop signal is nullified, and the robot cannot be step-stopped by the I/O input.

[Note 2] By activating mini I/O all-general option, step stop wiring becomes not necessary. However, wirings for auto enable input and emergency stop input are still necessary even if the option is activated.

2.3.2. I/O treatment for controllers with I/O extension board

If a robot controller is configures with extension I/O board (parallel I/O, DeviceNet, CC-Link, PROFIBUS, etc.), please refer “Installation and maintenance guide” and “Options Manual – Part2:

RC7M I/O extension board", and turn on "Step stop (All tasks) signal" and "Instantaneous stop signal".

2.4. Robot controller's Executable Token

2.4.1. Basic knowledge concerning robot controller's Executable Token

It is necessary to set the executable token for turning ON/OFF the motor or starting the program from the b-CAP application. (Refer to 2.2.1and 2.2.2). For the safety reason and to meet with "Single point of control" requirement, only the selected equipment can control a robot controller from the outside. Moreover, the robot controller becomes executable only in the external automatic mode as for turning ON/OFF the motor and starting the task from the b-CAP application.

The executable token changes as shown in the following figures.

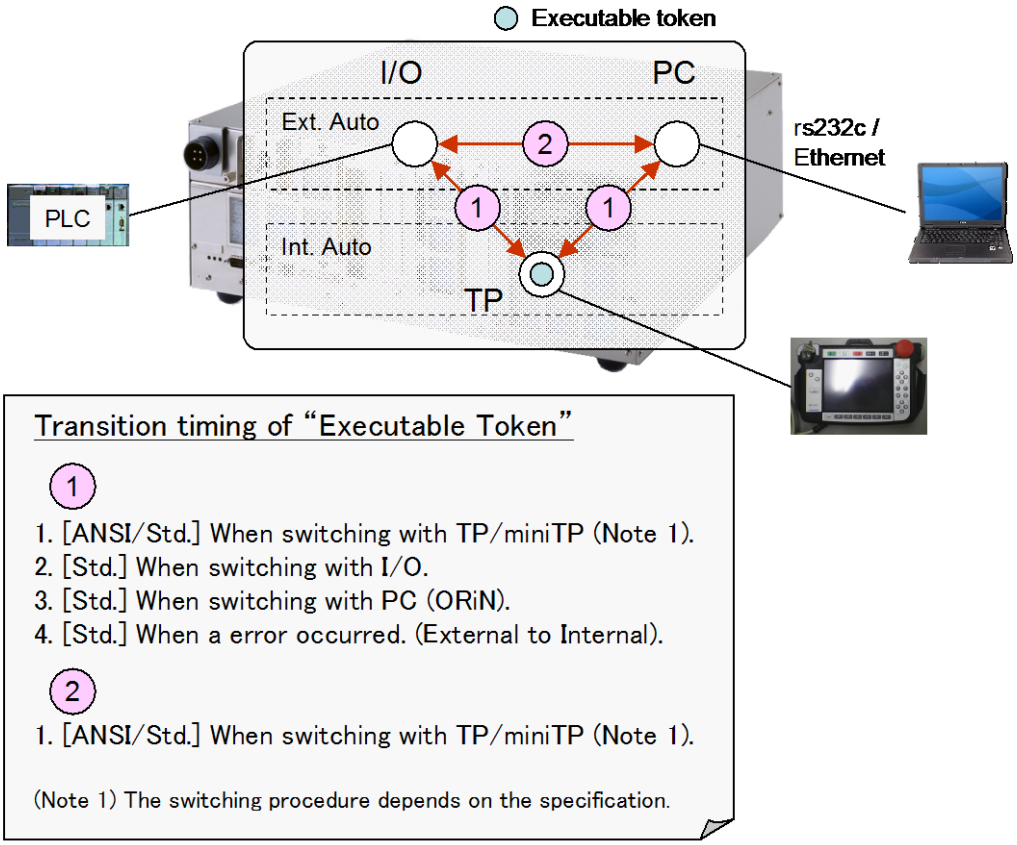


Figure 2-10 Transition of executable token

2.4.2. Notes in ANSI type robot controller

As previously stated, turning ON/OFF the motor and starting a program from the b-CAP application is possible only if the controller is in the external automatic mode.

For the robot controller of the ANSI type, please note that you need to go to "I/O Auxiliary Function" –

“single point of control” menu and select “External automatic operation” to change the robot controller to external automatic mode.

Following is the procedure to select external automatic mode in ANSI type controller. (Note: The ext button on RobMaster does not work for ANSI type robot controllers.)

(1) Mode selection procedure using the teaching pendant

After the following procedure is completed and robot is set to automatic mode, robot mode will be changed to the selected (internal or external) automatic mode.

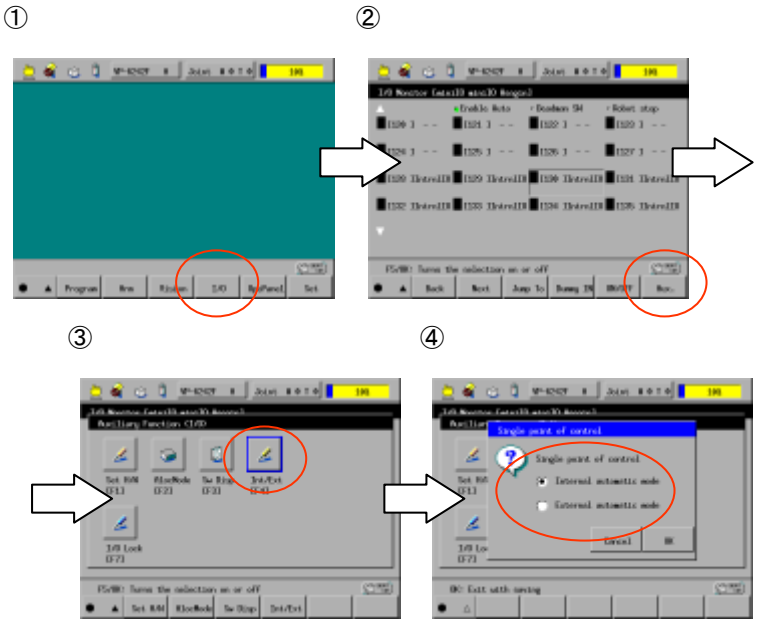


Figure 2-11 Internal/external automatic mode selection by teaching pendant

(2) Mode selection procedure using the mini pendant

After the following procedure is completed and robot is set to automatic mode, robot mode will be changed to the selected (internal or external) automatic mode.

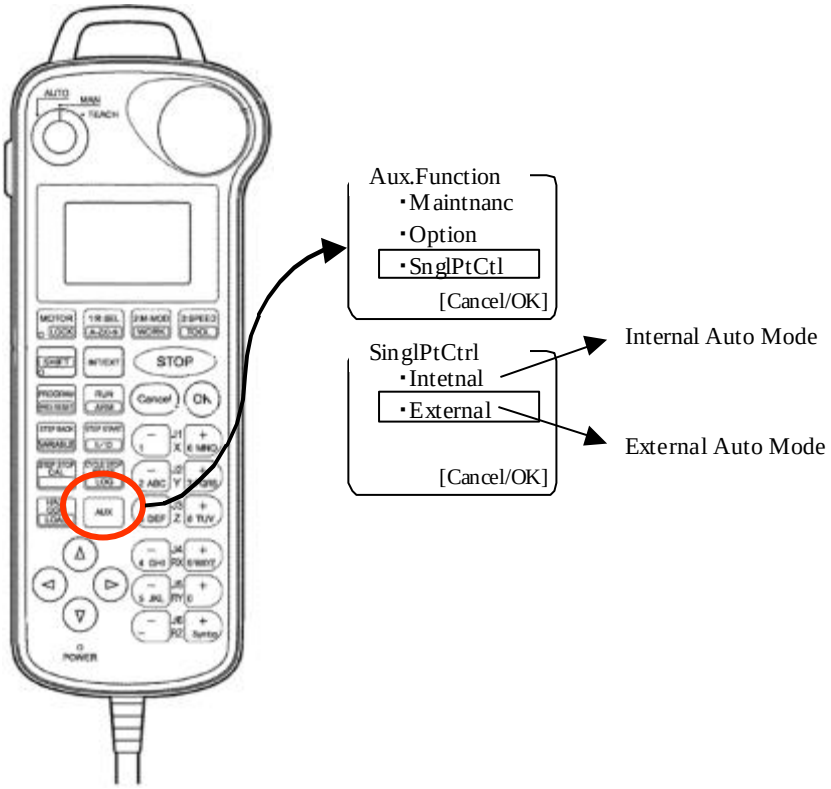


Figure 2-12 Internal/external automatic mode selection by mini pendant

2.5. Transferring PAC program

To execute robot motion commands (e.g. Move, Drive, etc) or to start task by b-CAP, following PAC programs, RobSlave.pac, RobSlave.h and UserExtention.pac, need to be sent to the robot controller and need to be executed.

To transfer PAC program to a robot controller, Using RobMaster.exe tool included ORiN SDK.

Mini pendant (mini TP) or Teach pendant (TP) is necessary for these method.

RobMaster.exe2 is a tool to show robot controller status and to control RobSlave task directly from PC, and the tool program is stored in NetwoRC provider installation folder.

<ORiN2 installation folder>¥CAO¥ProviderLib¥DENSO¥NetwoRC¥Bin

Start RobMaster.exe and follow this procedure to setup the controller.

1. Start robot controller, and change to [manual] mode.
2. Start RobMaster.exe program.
   Start > All Programs > ORiN2 > CAO > ProviderLib > RobMaster
3. Press [Connect] button to connect the program to the robot controller.

[2] <ORiN2 SDK install folder>¥CAO¥ProviderLib¥DENSO¥NetwoRC¥Bin¥RobMaster.exe

4. Press [Setup] button of RobMaster.exe and follow the instructions to send necessary PAC programs to the robot controller.

2.6. Introduction of RobMaster

Bundled tool RobMaster is connected to a robot controller, and offers the following function.

1. Set up a robot controller to be used with ORiN.
2. Start and stop the controller's RobSlave task.
3. Turn on and off the motor power of the controller.
4. Display robot controller error status and clear error.
5. Display robot controller tatues.

Following is the introduction of RobMaster functions.

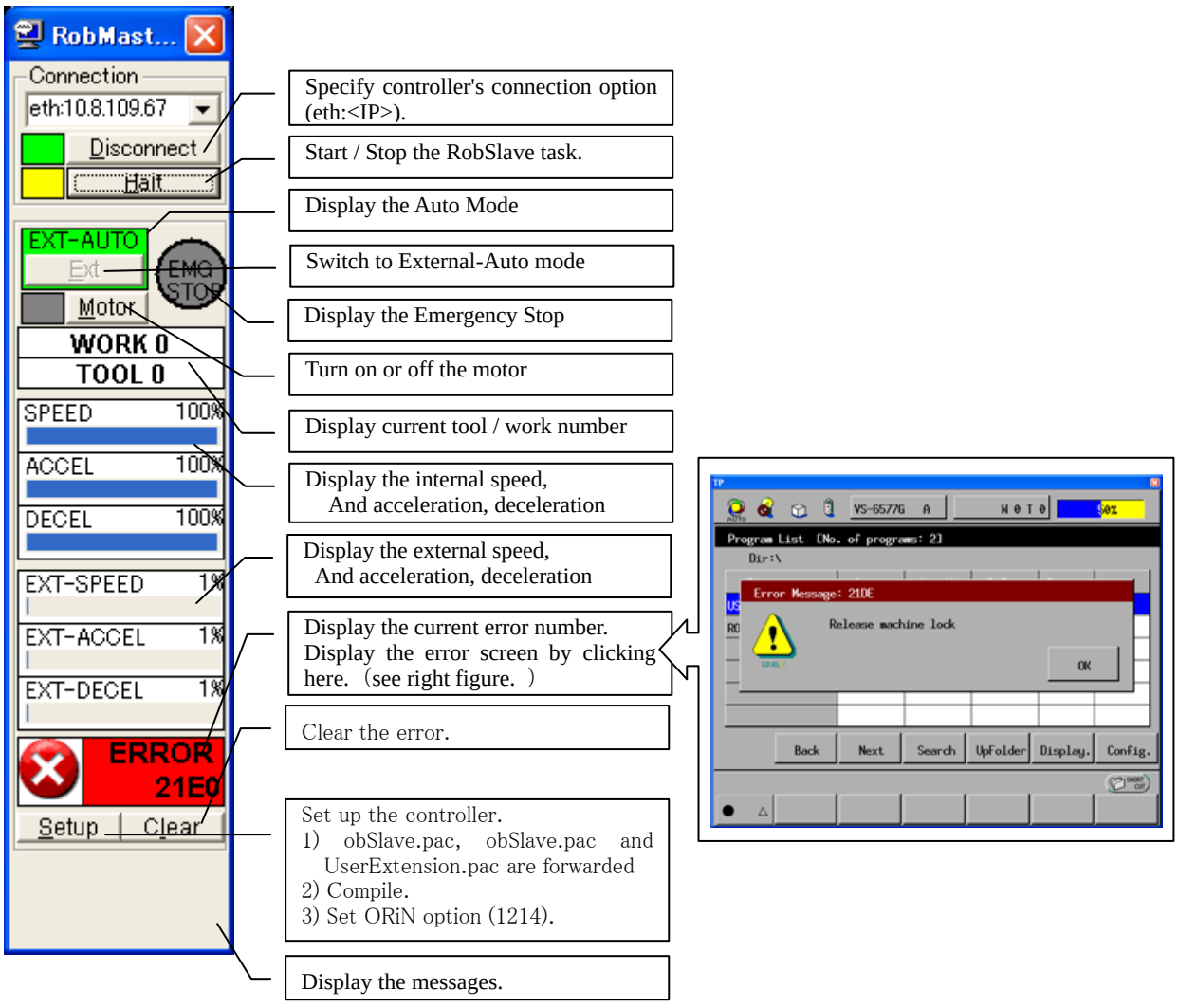


Figure 2-13 Function introduction of RobMaster

2.7. b-CAP Tester

b-CAP tester is a testing tool for sending and receiving b-CAP packet.

b-CAP tester is in the following folder:

ORiN2¥CAP¥b-CAP¥CapLib¥DENSO¥RC7¥Bin

Functions of b-CAP tester are like bellow. (See Fig12)

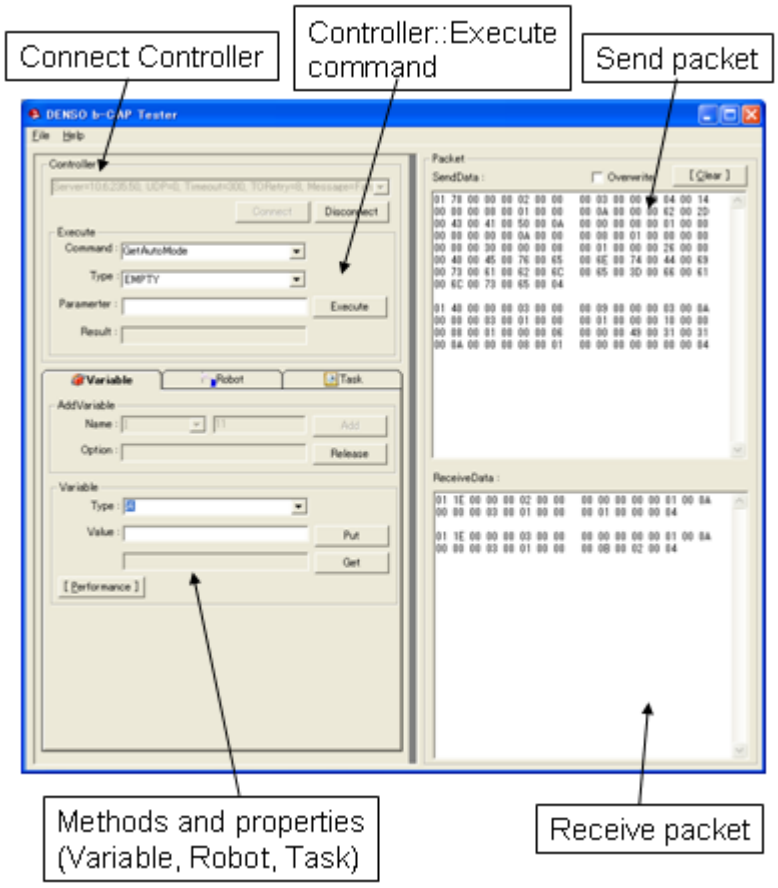


Figure 2-14 Function introduction of b-CAP Tester

3. Structure of b-CAP

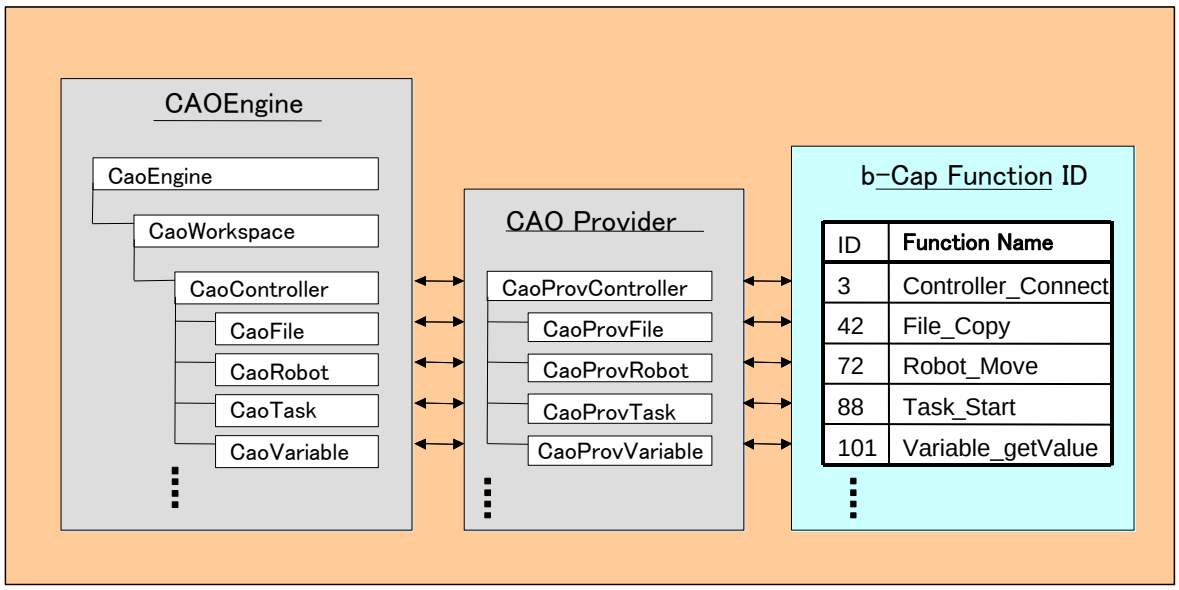


Figure 3-1 The structure of b-CAP

b-CAP consists of 2 programs - b-CAP client (which requests service) and b-CAP server (which performs service and returns the result).

b-CAP client creates a request message which contains necessary information for required service and send it to the server. It confirms the result by receiving the response message.

b-CAP server performs the service which corresponds to the Function ID after receiving the request message from the client. Then it puts the result and the value in a response message, which is sent to the client.

Procedures for client or server are detailed in Section 5 : Communication procedure.

The following is an example of connection of b-CAP.

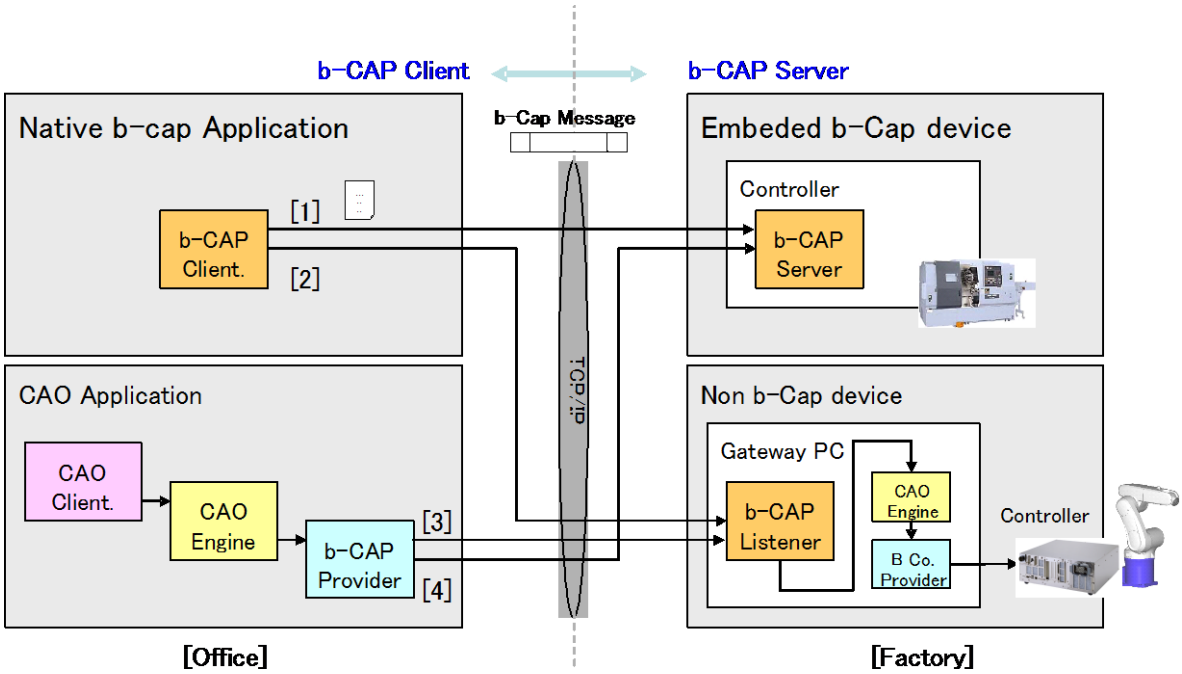


Figure 3-2 Example of b-CAP connection

4. Structure of the packet

4.1. Structure of the packet

The following is the structure of b-CAP message.

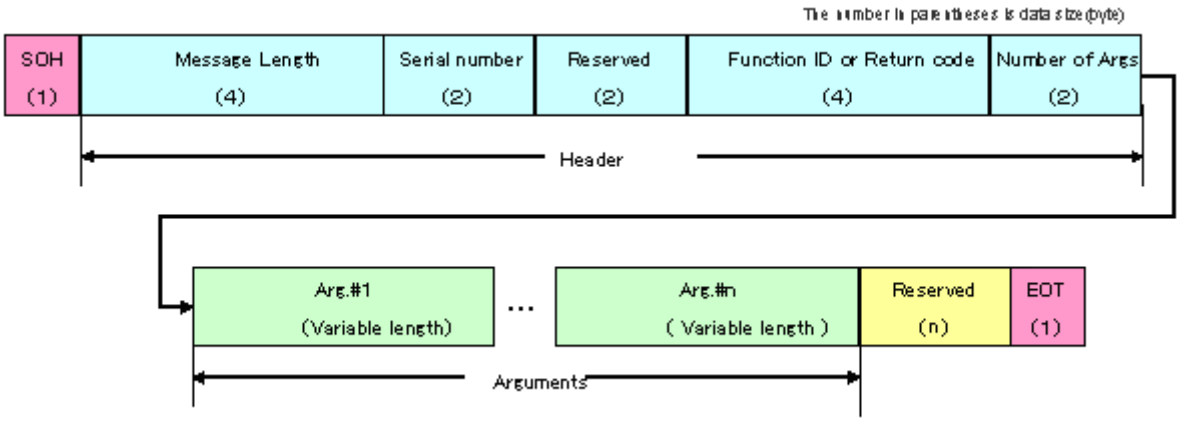


Figure 4-1 b-CAP packet structure

Descriptions of packet elements are listed as follows. The image of each data is stored in Intel format (little endian).

・ Header	Code places at the head of packet. SOH (0x01) is used.
・ Message length	Data length of the whole message. Unsigned long integer (DWORD) is stored. The length from header to terminator is stored.
・ Serial number	Serial number of the message. Unsigned short integer (WORD) is stored. Range of Serial number is 1 to WORD_MAX(0x0001~0xFFFF). The values have to be the same for request and respond messages.
・ Reserved area	Reserved area in the packet. “0” is always stored in this area.
・ Function ID	ID for visited function. Unsigned long integer (DWORD) is stored. Only used for request messages. (See 3.3)
・ Return Code	Code for performance result of visited function. Unsigned long integer (DWORD) is stored. Only used for response messages. (See 3.4)
・ The number of arguments	The number of arguments for visited function, or number of output variable for visited function. Unsigned short integer (WORD) is stored.
・ Argument #n	n-th argument (see 4.2)
・ Reservation area	It is an area that has been reserved with the system. This area is variable length area.

・ Terminator End code at the end of packet. EOT (0x04) is used.

4.2. Structure of argument part

Argument part varies in length depending on the data type. It is created to describe several data type.

The following is the structure of argument part.

The number in parentheses is data size(byte)

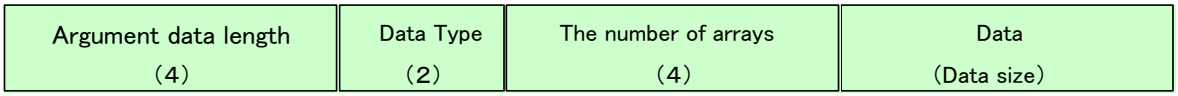


Figure 4-2 Augument data structure

Argument part includes data type and the number of arrays. The structure of data depends on data type and number of arrays.

The following is description of elements of argument part.

・ Argument data length Total bytes of “Data type”, “The number of arrays” and “Data”. But “Argument data length” is NOT included. Unsigned long integer (DWORD) is stored. See Table 4-1 Data type for data size of each type.

・ Data type Data type of argument. Unsigned short integer (WORD) is stored. See Table 4-1 for enabled data type.

・ The number of arrays The number of arrays in an argument.
Unsigned Long integer (DWORD) is stored.
The value is always “1” when VT_ARRAY is not used for data type.

・ Data Data of argument. Varies in size depending on the type.
See for Table 4-1 each data size. See Figure 4-3 for the structure of information stored in data.

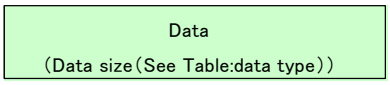
Table 4-1 Data type

Data Type	Value	Size (Byte)	Description
VT_EMPTY	0	0	Empty data
VT_NULL	1	0	NULL value
VT_ERROR	10	2	Error code
VT_UI1	17	1	Binary
VT_I2	2	2	Short integer
VT_UI2	18	2	Unsigned short integer
VT_I4	3	4	Long integer

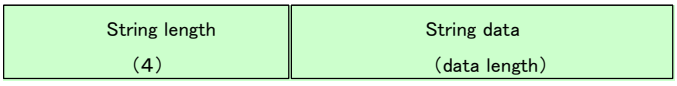
VT_UI4	19	4	Unsigned long integer
VT_R4	4	4	Single-precision floating point
VT_R8	5	8	Double-precision floating point
VT_CY	6	8	Currency type
VT_DATE	7	8	Date type
VT_BOOL	11	2	Boolean type
VT_BSTR	8	(Number of characters) ×2 + 4	String type String type consists of “String length” and “String data”. ”String data” is stored after “string length” in Unicode, where one character is stored with 2 bytes. “String length” describe the number of byte in “String data”
VT_VARIANT	12	-	Variant type Structure is the same as argument part. Used only with VT_ARRAY.
VT_ARRAY	0x2000	-	Array Data type is determined by logical OR. Data of the specified type is stored in a row.

The number in parentheses is data size(byte)

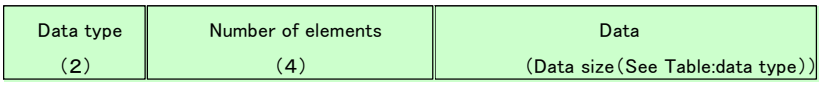
・VT_I2,VT_I4,VT_UI1,etc
(Other than VT_BSTR,VT_VARIANT,VT_ARRAY)



・VT_BSTR



・VT_VARIANT



・Same structure as the argument

・VT_ARRAY

Figure 4-3 Data structure

4.3. Function ID

For b-CAP, function IDs are assigned as follows.

Table 4-2 Function ID assignment

Function ID	Description
1 - 137	For given functions
138 - 255	For reserved area
256 -	For user-defined functions

ID for user-defined functions can be used for functions which are not shown in Table 4-3.

The following is the list of given functions of b-CAP.

Table 4-3 Given functions

Function ID	Function Name	Description
1	Service_Start	Starts the b-CAP service
2	Service_Stop	Stops the b-CAP service
3	Controller_Connect	Connect to a robot controller
4	Controller_Disconnect	Disconnect from the robot controller
7	Controller_GetRobot	Get a robot object
8	Controller_GetTask	Get a task object
9	Controller_GetVariable	Get a variable object
17	Controller_Execute	Execute the extension function of controller
62	Robot_GetVariable	Get a variable object of the robot
64	Robot_Execute	Execute a command of the robot
66	Robot_Change	Execute the change command of the robot
72	Robot_Move	Move the robot
84	Robot_Release	Release the robot object
85	Task_GetVariable	Get a variable object of the task
88	Task_Start	Start a task
89	Task_Stop	Stop a task
99	Task_Release	Release the task object
101	Variable_GetValue	Get a value of the variable
102	Variable_PutValue	Put a value of the variable

111	Variable_Release	Release the variable object

4.4. Return Code

For b-CAP, return codes are assigned as follows.

Table 4-4 Return code assignment

Return code	Description
0x00000000～0x8000FFFF 0x80010000～0x800101FF 0x80070000～0x8007FFFF	For given return codes and reserved area
0x80040200～0x8004FFFF	For user-defined errors

Any code for user-defined errors can be used for errors which are not shown in Table 5.

Table 4-5 Given return codes

Return code	Error	Description
0x00000000	S_OK	OK
0x80004001	E_NOTIMPL	Not implemented
0x80004004	E_ABORT	Function aborted
0x80004005	E_FAIL	Function failed
0x8000FFFF	E_UNEXPECTED	Fatal Error occurred
0x80010001	E_INVALIDRCVPACKET	Invalid packet is received. When this error is occurred, robot controller disconnect from client immediately. Please make sure the packet that you sent.
0x80010002	E_INVALIDSNDPACKET	Invalid packet is sent
0x80010003	E_INVALIDARGTYPE	Invalid argument type
0x80010004	E_ROBOTISBUSY	Robot is busy (Wait for a while)
0x80010005	E_INVALIDCOMMAND	Invalid command string is received
0x80010011	E_PACKETSIZEOVER	Received packet size over (> 16Mbytes)
0x80010012	E_ARGSIZEOVER	An argument siez over of the received packet. (> 16Mbytes)

0x80070005	E_ACCESSDENIED	Access denied
0x80070006	E_HANDLE	Invalid handle
0x8007000E	E_OUTOFMEMORY	Out of memory
0x80070057	E_INVALIDARG	Invalid argument

5. Communication procedure

5.1. Communication sequence

Sequence of b-CAP begins with sending a request packet from a client.

The server performs the request packet function and sends a response packet to the client.

In one session, after the demand message is transmitted, it is necessary to wait for the reception of the response message, and to always take synchronization. When two or more request messages are used, it is preferable to do by two or more sessions.

There is not regulations for time from the reception of the demand packet on the server side to the reply of the answer either. Therefore, it is necessary to note it because the time-out will be generated on the client side when the time-out detection time of the client is short when the processing time of the server side is long.

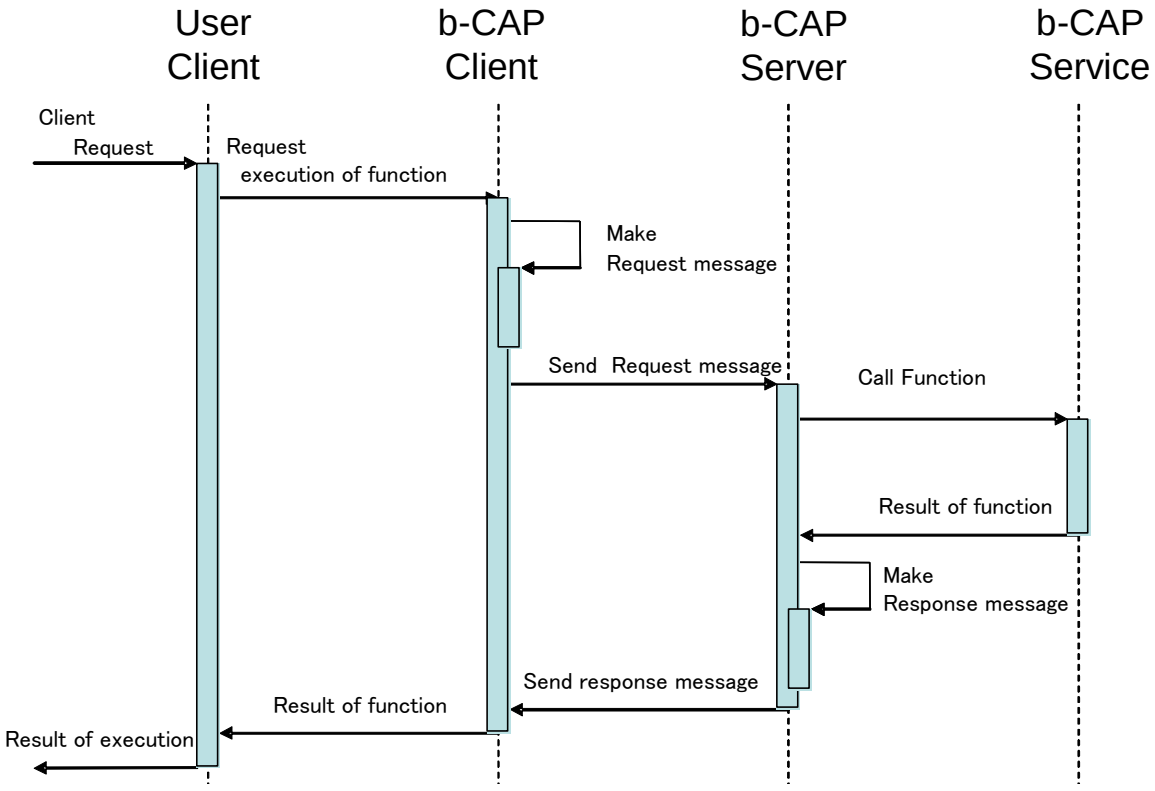


Figure 5-1 Communication sequence

5.2. Communication procedure for server

The following is the outline of communication procedure for server.

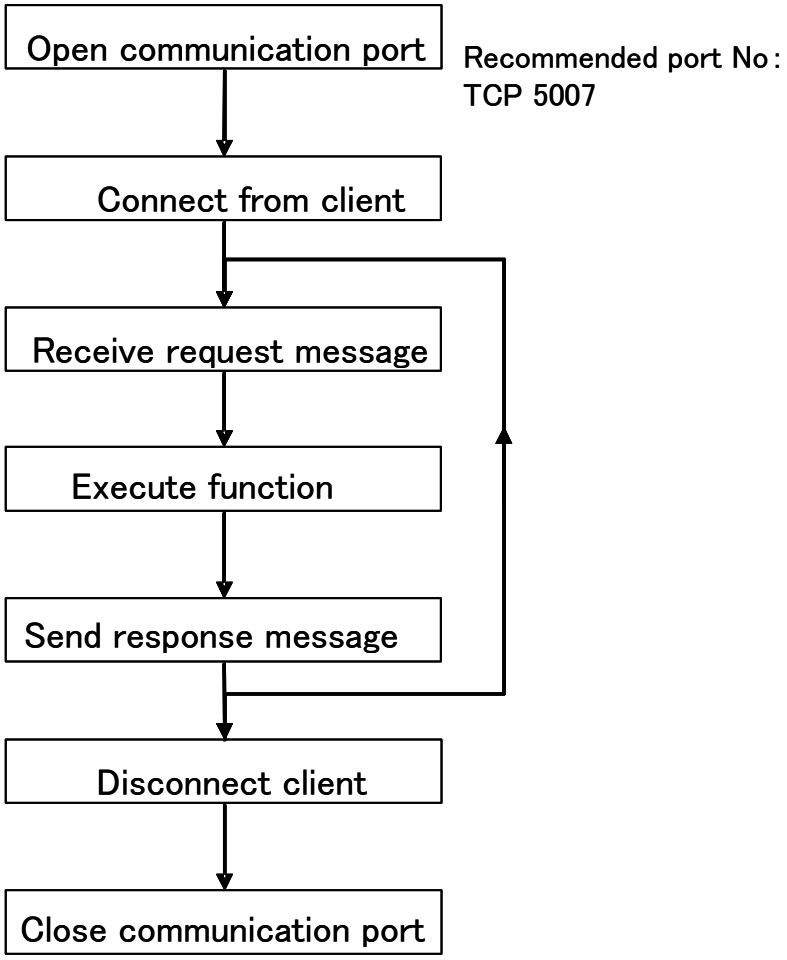


Figure 5-2 Communication procedure of b-CAP server.

Server opens the communication port and waits for client to connect. It is recommended to open TPC port No. 5007.

After the connection, server performs the function which corresponds to the request message. The result is stored in a response message and sent to the client.

5.3. Communication procedure for client

The following is the outline of communication procedure for client.

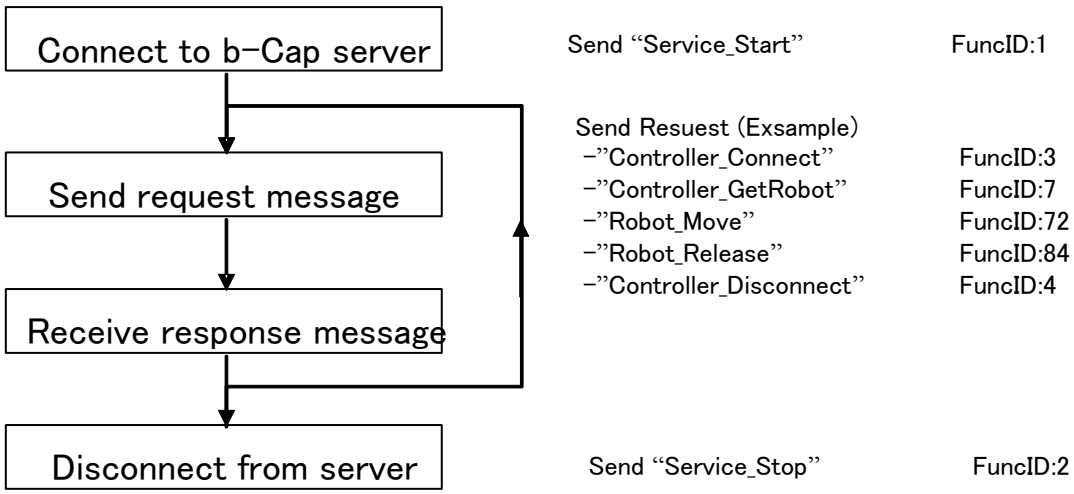


Figure 5-3 Communication procedure of b-CAP client application

Client establishes a session by connecting to the server, then sends a request message of performed function and waits for a response message from the server.

Client needs to perform time-out processing when no response comes from the server. However, response time varies depending on job, therefore, attention needs to be paid to setting the time-out period.

5.4. Application consideration

Inside of the controller, b-CAP communication server is running higher priority than PAC language and the other communication processes.

So, Calling b-CAP in high frequency have harmful effects for execution speed of PAC . And While commucating by b-CAP with the Robot controller in high-frequency, some communication error may occur in Wincaps or some deley may occur in the b-CAP client.

To resolve the problem, reduce b-CAP communication load or stop it temporary.

6. b-CAP communication function

6.1. Start and stop of b-CAP service

6.1.1. Service_Start

Function HRESULT Service_Start()

Function ID 1

Argument No argument

Return Value See Table 4-5 Given return codes

Description Starts b-CAP service

See also Service_Stop

Communication sample 1				
This sample procedure starts the b-CAP server.				
Packet TX	TX(Client->Server): 01 10 00 00 00 01 00 00 00 01 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		
Packet RX	RX(Server->Client): 01 10 00 00 00 01 00 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

6.1.2. Service_Stop

Function HRESULT Service_Stop ()

Function ID 2

Argument No argument

Return Value See Table 4-5 Given return codes

Description Stops b-CAP service

See also Service_Stop

<table>
<tr><td colspan="5">Communication sample 1</td></tr>
<tr><td colspan="5">This sample procedure stops the b-CAP server.</td></tr>
<tr><td>Packet
TX</td><td colspan="4">TX(Client->Server):
01 10 00 00 00 08 00 00 00 02 00 00 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td colspan="4">Binary</td></tr>
<tr><td>No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="4">-</td></tr>
<tr><td>Packet
RX</td><td colspan="4">RX(Server->Client):
01 10 00 00 00 08 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td colspan="4">Binary</td></tr>
<tr><td>No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="4">-</td></tr>
</table>

6.2. Controller objects

6.2.1. Controller_Connect

Function	HRESULT Controller_Connect ()			
Function ID	3			
Argument	[in]	BSTR	bstrCtrlName	Controller name(Not used)
	[in]	BSTR	bstrProvName	Provider name (Not used)
	[in]	BSTR	bstrPcName	Provider execution machine name(Not used)
	[in]	BSTR	bstrOption	Option character string = "<option1>, <option2>, …" (Not used)
	[out]	long	hController	Handle of controller
Return Value	See Table 4-5 Given return codes			
Description	Gets the handle of the controller object “hController” by connecting to the controller			
See also	Controller_Disconnect			

<table>
<tr><td colspan="2">Communication sample 1</td></tr>
<tr><td colspan="2">This function returns the handle of the controller - hCOntroller.</td></tr>
<tr><td>Packet
TX</td><td>TX(Client->Server):
01 48 00 00 00 02 00 00 00 03 00 00 00 04 00 0A
00 00 00 08 00 01 00 00 00 00 00 00 0A 00 00
00 08 00 01 00 00 00 00 00 00 00 0A 00 00 00 08
00 01 00 00 00 00 00 00 00 0A 00 00 00 08 00 01
00 00 00 00 00 00 00 04</td></tr>
</table>

	Name	Description	Type	Value
		Binary		
	bstrCtrlName	The name of the controller. (Not used in RC7)	VT_BSTR	Null String
		00 00 00 08 00 01 00 00 00 00 00 00 00 0A		
	bstrProvName	The name of the provider. (Not used in RC7)	VT_BSTR	Null String
		00 08 00 01 00 00 00 00 00 00 00 0A 00 00		
	bstrPcName	The name of the client PC. (Not used in RC7)	VT_BSTR	Null String
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		
	bstrOption	The connecting option. (Not used in RC7)	VT_BSTR	Null String
		00 00 00 00 00 00 00 0A 00 00 00 08 00 01		
Packet RX	RX(Server->Client): 01 1E 00 00 00 02 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	hController	The handle of controller	VT_I4	0x000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		

6.2.2. Controller_Disconnect

Function HRESULT Controller_Disconnect ()

Function ID 4

Argument [in] long hController Handle of controller

Return Value See Table 4-5 Given return codes

Description Disconnects from the controller specified by “hController”

See also Controller_Connect

Communication sample 1
This function disconnects from the controller that indicated by handle of controller - hCOntroller.

<table>
<tr><td>Packet
TX</td><td colspan="3">TX(Client->Server):
01 1E 00 00 00 03 00 00 00 04 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 01 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>hController</td><td>The handle of the controller
(See also Controller_Connect)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td colspan="4">00 00 00 03 00 01 00 00 00 01 00 00 00 04 0A</td></tr>
<tr><td>Packet
RX</td><td colspan="3">RX(Server->Client):
01 10 00 00 00 03 00 00 00 00 00 00 00 04</td></tr>
<tr><td rowspan="3"></td><td>Name</td><td>Description</td><td>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>

6.2.3. Controller_GetVariable

Function	HRESULT Controller_GetVariable ()			
Function ID	9			
Argument	[in]	long	hController	Handle of controller
	[in]	BSTR	bstrName	Variable name
	[in]	BSTR	bstrOption	Option character string
	[out]	long	hVariable	Handle of variable
Return Value	See Table 4-5 Given return codes			
Description	Gets the handle of the controller system variable object “hVariable”			
	By using this handle, The application software can accesses to resources of the variable.			
See also	Variable_GetValue			
	Variable_PutValue			
	Robot_GetVariable			
	Task_GetVariable			

In this function, variables in controller can be obtained by using the following variable names for “bstrName”

Table 6-1 User-accessible variables for controller class

Variable Name	Data type	Description	Attribute	
			get	put

I example) “I10”	VT_I4	Variable I Put ID No. of variable (0 -) after the variable name.	∨	∨
F example) “F10”	VT_R4	Variable F Put ID No. of variable (0 -) after the variable name.	∨	∨
D example) “D10”	VT_R8	Variable D Put ID No. of variable (0 -) after the variable name.	∨	∨
V example) “V10”	VT_ARRAY | VT_R4	Variable V Put ID No. of variable (0 -) after the variable name. The number of data arrays is 3.	∨	∨
P example) “P10”	VT_ARRAY | VT_R4	Variable P Put ID No. of variable (0 -) after the variable name. The number of data arrays is 7.	∨	∨
J example) “J10”	VT_ARRAY | VT_R4	Variable J Put ID No. of variable (0 -) after the variable name. The number of data arrays is 6 or 4 depend ed on the robot type.	∨	∨
T example) “T10”	VT_ARRAY | VT_R4	Variable T. Put ID No. of variable (0 -) after the variable name. The number of data arrays is 10.	∨	∨
S example) “S10”	VT_BSTR	Variable S (String). Put ID No. of variable (0 -) after the variable name.	∨	∨
IO example) “IO10”	VT_BOOL	Variable IO Put ID No. of variable (0 -) after variable name.	∨	∨
TOOL example) “TOOL63”	VT_ARRAY | VT_R4	Tool settings: Put ID No. of variable (0 -63) after variable name. NOTE: ID No.0(TOOL0) is read only.	∨	∨

WORK example) “WORK7”	VT_ARRAY | VT_R4	Work settings: Put ID No. of variable (0 -7) after variable name. NOTE: ID No.0(WORK0) is read only.	∨	∨
AREA example) “AREA7”	VT_ARRAY | VT_R4	AREA settings: Put ID No. of variable (0 -7) after variable name.	∨	∨
_ITP example) “_ITP10”	VT_I4	ITP Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_PAC example) “_PAC10”	VT_I4	PAC Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_DIO example) “_DIO10”	VT_I4	DIO Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_ARM example) “_ARM10”	VT_I4	ARM Configurations: Put ID No. of variable (0 -) after variable name.	∨	∨

		NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.		
_SRV example) "_SRV10"	VT_I4	SRV Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_SPD example) "_SPD10"	VT_I4	SPD Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_VIS example) "_VIS10"	VT_I4	VIS Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨
_COM example) "_COM10"	VT_I4	COM Configurations: Put ID No. of variable (0 -) after variable name. NOTE1) ID No.0 is the numbers of this configurations. NEVER PUT VALUE. NOTE2) By putting wrong value,Controller may be dameged critically.	∨	∨

@MODE	VT_I2	1: Manual 2: Teach check 3: Internal automatic 4: External Automatic	∨	-
@EMERGENCY_STOP	VT_BOOL	true = Emergency stop switch is ON. false = Emergency stop switch is OFF.	∨	-
@ERROR_CODE	VT_I4	ID No. of the error which is occurring Returns "0" when no error occurs.	∨	-
@ERROR_DESCRIPTION	VT_BSTR (In version 1.0.9(Controller V2.624) and the earlier verson ,This data type was VT_ARRAY | VT_UI1)	Error description of the error which is occurring now. Note : When no error had been occurred, This variable returns "Undefined error".	∨	-
@AUTO_ENABLE	VT_I2	1 = Auto Enable signal is ON. 0 = Auto Enable signal is OFF.	∨	-
@PROTECTIVE_STOP	VT_I2	1 = Protective stop is ON. 0 = Protective stop is OFF.	∨	-
@DEADMAN_SW	VT_I2	1 = Deadman switch is ON. 0 = Deadman switch is OFF.	∨	-
@VERSION	VT_BSTR	Version string of the robot controller	∨	-

Communication sample 1

This function returns the handle of the variable - hVariable.

By using this hanlde, application can access to variable.

Packet

TX

TX(Client->Server):

```
01 3E 00 00 00 05 00 00    00 09 00 00 00 03 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 49 00 31 00 0A
00 00 00 08 00 01 00 00    00 00 00 00 00 04
```

Name	Description	Type	Value
	Binary		
hController	The handle of the controller (See also Controller_Connect)	VT_I4	0x0000001

		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrName	The variable name	VT_BSTR	"I1"
		00 08 00 01 00 00 00 04 00 00 00 49 00 31 00 0E 00 00		
	bstrOption	bstrOption	VT_BSTR	Null String
		00 00 00 08 00 01 00 00 00 00 00 00 00 0A		
Packet RX	RX(Server->Client): 01 1E 00 00 00 05 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 01 00 02 00 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable "I1" is returned.	VT_I4	0x00020001
		00 00 00 03 00 01 00 00 00 01 00 02 00 04 0A		

6.2.4. Controller_Execute

Function HRESULT Controller_Execute()

Function ID 17

Argument	[in]	long	hController	Handle of controller
	[in]	BSTR	bstrCommand	Command name
	[in]	VARIANT	vntParam	Parameter
	[out]	long	pVal	Result value

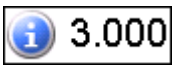
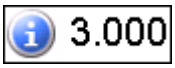
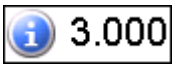
Return Value See Table 4-5 Given return codes

Description Execute the command of the controller "hController"

In this function, commands can be executed by using the following command names for "bstrCommand".

Table 6-2 Implemented command list

Command	Parameter	Return value	Operation
GetAutoMode	None	VT_I2: Mode(0:Unkown 1:Internal auto 2:External auto)	Get auto mode

PutAutoMode	VT_I2: Mode(1:Internal auto 2:External auto)	None	Set auto mode
ClearError	:VT_I4: ErrorCode	VT_I2: Result Code	Clear error Note that "ClearError" is time-consuming command. It may take about 1 sec.
SuspendAll	None	None	Suspend all pac programs.
ResetAll	None	None	Kill and Reset all Pac programs.
StartLog 	None	None	Execute StartLog
StopLog 	None	None	Execute StopLog
ClearLog 	None	None	Execute ClearLog

Communication sample 1

This function executes a command of a controller.

In this sample, "ClearError" is executed.

Packet TX

TX(Client->Server):

```
01 4E 00 00 00 0D 00 00   00 11 00 00 00 03 00 0A
00 00 00 03 00 01 00 00   00 01 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 0A
00 00 00 03 00 01 00 00   00 00 00 00 00 04
```

Name	Description	Type	Value
	Binary		
hController	The handle of the controller (See also Controller_Connect)	VT_I4	0x0000001
	`00 00 00 03 00 01 00 00  00 01 00 00 00  0A`		
bstrCommand	The command string	VT_BSTR	"ClearError"
	`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`		

	vntParam	The Error code that should be cleared. (In RC7, This value is not used, please use 0x00000000)	VT_I4	0x00000000
		00 00 00 03 00 01 00 00 00 00 00 00 00 04 0A		
Packet RX	RX(Server->Client): 01 1C 00 00 00 46 00 00 00 00 00 00 00 01 00 08 00 00 00 02 00 01 00 00 00 01 00 04			
	Name	Description	Type	Value
		Binary		
	Result	Result of ErrorClear (1 = Succeeded)	VT_I2	0x0001
		00 00 00 02 00 01 00 00 00 01 00 01 00 08		

6.2.5. Controller_GetTaskNames

Function HRESULT Controller_GetTaskNames ()

Function ID 14

Argument				
	[in]	long	hController	Handle of controller
	[in]	BSTR	bstrOption	Option character string
	[out]	VARIANT	TaskNames	the list of pac tasks.

Return Value See Table 4-5 Given return codes

Description Get the list of pac tasks.

Note 1) In thing function, Tye type of result(=list of pac tasks) value is **VT_VARIANT | VT_ARRAY.**

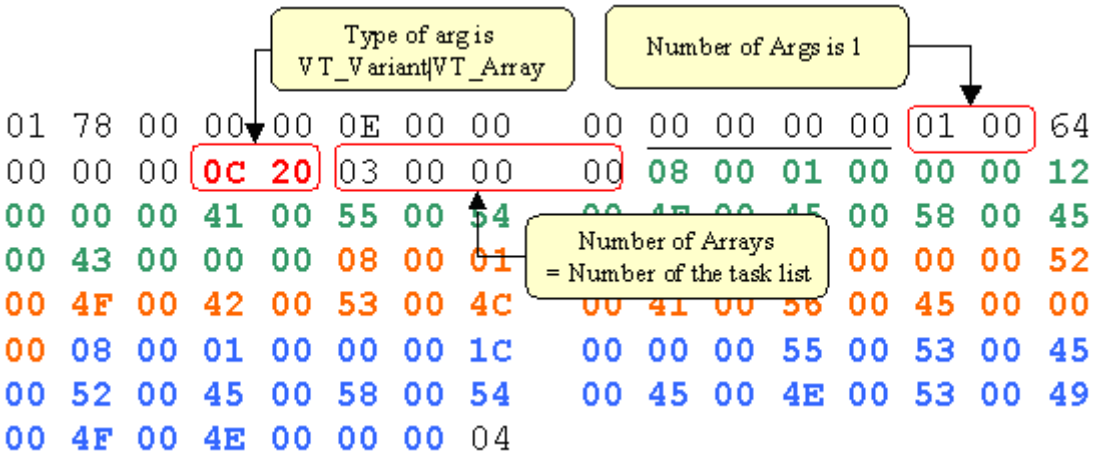
Note 2) When many pac task is in the controller, then returned packet became large. The client program should prepare enough receive buffers.

Communication sample 1				
This function returns the list of pac tasks.				
Packet TX	TX(Client->Server): 01 2C 00 00 00 0E 00 00 00 0E 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	hController	The handle of the controller (See also Controller_Connect)	VT_I4	0x0000001

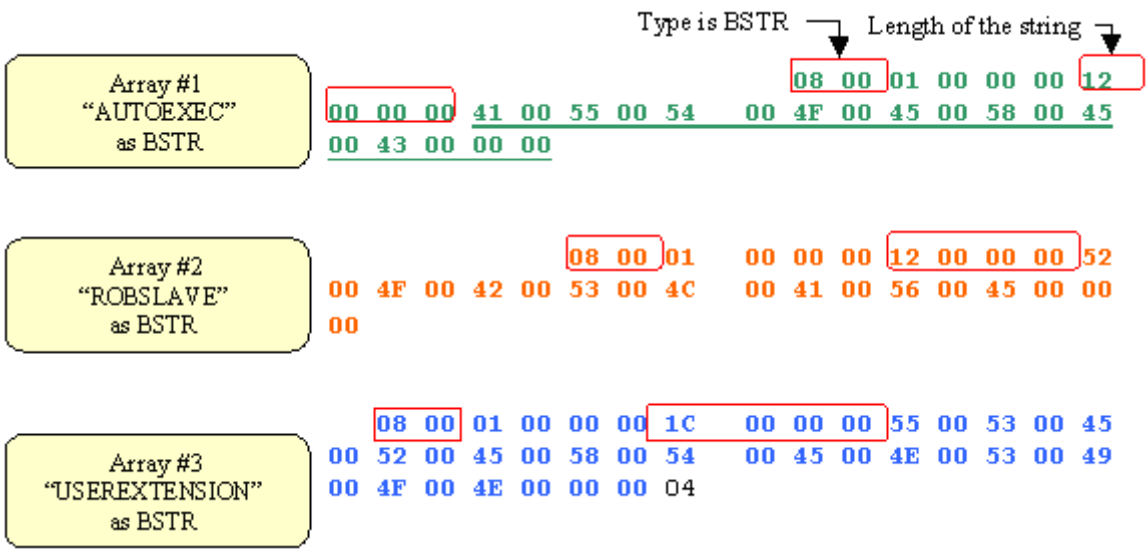
<table>
<tr><td rowspan="4"></td><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 01 00 00 00 0A</td></tr>
<tr><td rowspan="2">bstrOption</td><td>The option string</td><td>VT_BSTR</td><td>Null string</td></tr>
<tr><td colspan="3">00 08 00 01 00 00 00 00 00 00 00 0A 00 00</td></tr>
<tr><td>Packet
RX</td><td colspan="4">RX(Server->Client):
01 78 00 00 00 0E 00 00 00 00 00 00 00 00 00 00 00 01 00 64
00 00 00 0C 20 03 00 00 00 08 00 01 00 00 00 12
00 00 00 41 00 55 00 54 00 4F 00 45 00 58 00 45
00 43 00 00 00 08 00 01 00 00 00 12 00 00 00 52
00 4F 00 42 00 53 00 4C 00 41 00 56 00 45 00 00
00 08 00 01 00 00 00 1C 00 00 00 55 00 53 00 45
00 52 00 45 00 58 00 54 00 45 00 4E 00 53 00 49
00 4F 00 4E 00 00 00 04</td></tr>
<tr><td></td><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td></td><td colspan="4">Binary</td></tr>
<tr><td></td><td>TaskNames</td><td>The list of pac tasks.</td><td>VT_ARRAY |
VT_VARIANT</td><td></td></tr>
<tr><td></td><td colspan="4">00 00 00 0C 20 03 00 00 00 08 00 01 00 00 00 12
00 00 00 41 00 55 00 54 00 4F 00 45 00 58 00 45
00 43 00 00 00 08 00 01 00 00 00 12 00 00 00 52
00 4F 00 42 00 53 00 4C 00 41 00 56 00 45 00 00
00 08 00 01 00 00 00 1C 00 00 00 55 00 53 00 45
00 52 00 45 00 58 00 54 00 45 00 4E 00 53 00 49
00 4F 00 4E 00 00 00 64</td></tr>
</table>

This function retuns pac task list as VT_VARIANT|VT_ARRAY.

The structure of this received packet is below.



The structure of the argument of this Variant is below.



6.3. Robot objects

6.3.1. Controller_GetRobot

Function	HRESULT Controller_GetRobot ()			
Function ID	7			
Argument	[in]	long	hController	Handle of controller
	[in]	BSTR	bstrName	Name of robot
	[in]	BSTR	bstrOption	Option character string
	[out]	long	hRobot	Handle of robot
Return Value	See Table 4-5 Given return codes			
Description	Get the handle of robot object "hRobot"			
	By using this handle, The application software can accesses to resources of the robot.			
See also	Controller_Connect			

Communication sample 1				
This function returns the handle of robot - hRobot.				
Packet TX	TX(Client->Server): 01 3A 00 00 00 10 00 00 00 07 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		

	hController	The handle of the controller (See also Controller_Connect)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00		0A
	bstrName	The name of the robot (In RC7, This value is not used)	VT_BSTR	Null String
		00 08 00 01 00 00 00 00 00 00 00		0A 00 00
	bstrOption	bstrOption	VT_BSTR	Null String
		00 01 00 00 00 00 00 00 00		0A 00 00 00 08
Packet RX	RX(Server->Client): 01 1E 00 00 00 10 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of the robot	VT_I4	0x000001
		00 00 00 03 00 01 00 00 00 01 00 00 00		0A

6.3.2. Robot_Release

Function HRESULT Robot_Release ()

Function ID 84

Argument				
	[in]	long	hRobot	Handle of robot

Return Value See Table 4-5 Given return codes

Description Release the robot objets specified by “hRobot”

6.3.3. Robot_GetVariable

Function HRESULT Robot_GetVariable ()x

Function ID 62

Argument				
	[in]	long	hRobot	Handle of robot
	[in]	BSTR	bstrName	Variable name
	[in]	BSTR	bstrOption	Option character string
	[out]	long	hVariable	Handle of variable

Return Value See Table 4-5 Given return codes

Description Get the handle of robot system variable object “hVariable”

By using this handle, The application software can accesses to resources of the variable.

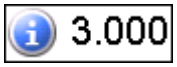
See also Variable_GetValue

Variable_PutValue

Controller_GetVariable

Task_GetVariable

In this function, information on robots can be obtained by using the following variable names for "bstrName".

Variable Name	Data type	Description	Attribute	
			get	put
@CURRENT_POSITION	VT_ARRAY | VT_R4	Current position of the robot (X, Y, Z, Ry, Ry, Rz, Fig)	∨	-
@CURRENT_ANGLE	VT_ARRAY | VT_R4	Current position of the robot （Angle of each axis）	∨	-
@CURRENT_TRANS	VT_ARRAY | VT_R4	Current position of the robot expressed in T Type. （X,Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig）	∨	-
@CURRENT_TOOL	VT_I2	Currently used tool number	∨	-
@CURRENT_WORK	VT_I2	Currently used work number	∨	-
@SPEED	VT_R4	Internal speed	∨	-
@ACCEL	VT_R4	Internal acceleration	∨	-
@DECEL	VT_R4	Internal deceleration	∨	-
@EXTSPEED	VT_R4	External speed	∨	-
@EXTACCEL	VT_R4	External acceleration	∨	-
@EXTDECEL	VT_R4	External deceleration		
@SERVO_ON	VT_I2	Servo state, 0=OFF,1= ON	∨	-
@TYPE	VT_I4	Robot type	∨	-
@HIGH_CURRENT_POSITION 	VT_ARRAY | VT_R4	Current robot position expressed in P Type with time stamp. (X, Y, Z, Ry, Ry, Rz, Fig, time stamp) The value is indefinite under machine-locked	∨	-

		because it is retrieved from an encoder directly. When the controller is not in machine-lock mode, the current encoder value is returned. (update resolution: 500 microsecond) When the controller is in machine-lock mode, the internal position target value is returned. (update resolution: 8 msec) [Note1]On machine-lock mode, current position is indefinite. [Note2]The resolution of the time stamp is 23 bit. Therefore the value is reset to zero when it is over 0x0x7FFFFF.		
@HIGH_CURRENT_ANGLE 3.000	VT_ARRAY | VT_R4	Current position of the robot expressed in J Type with time stamp. （Angle of each axis, time stamp） For function specification, please refer to @HIGH_CURRENT_POSITION.	∨	-
@HIGH_CURRENT_TRANS 3.000	VT_ARRAY | VT_R4	Current position of the robot expressed in T Type. （X,Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, time stamp） For function specification, please refer to @HIGH_CURRENT_POSITION.	∨	-

Communication sample 1				
This function returns the handle of variable - hVariable.				
Packet TX	TX(Client->Server): 01 5C 00 00 00 12 00 00 00 3E 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 2C 00 00 00 08 00 01 00 00 00 22 00 00 00 40 00 43 00 55 00 52 00 52 00 45 00 4E 00 54 00 5F 00 50 00 4F 00 53 00 49 00 54 00 49 00 4F 00 4E 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		

	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 0A		
	bstrName	The name of the varibale	VT_BSTR	"@CURRENT_ POSITION"
		00 08 00 01 00 00 00 22 00 00 00 40 00 43 00 55 00 52 00 52 00 45 00 4E 00 54 00 5F 00 50 00 4F 00 53 00 49 00 54 00 49 00 4F 00 4E 00 2C 00 00		
	bstrOption	Option	VT_BSTR	Null String
		00 08 00 01 00 00 00 00 00 00 00 0A 00 00		
Packet RX	RX(Server->Client): 01 1E 00 00 00 12 00 00 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 00 00 00 30 00 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of the variable	VT_I4	0x300000
		00 00 00 03 00 01 00 00 00 00 00 30 00 0A		

6.3.4. Robot_Move

Function	HRESULT Robot_Move ()			
Function ID	72			
Argument	[in]	long	hRobot	Handle of robot
	[in]	long	lComp	Interpolation 1:MOVE P 2:MOVE L
	[in]	VARIANT	vntPose	Pose data (Only BSTR)
	[in]	BSTR	bstrOpt	Motion option
Return Value	See Table 4-5 Given return codes			
Description	Robot specified by the handle of robot object "hRobot" moves to the specified coordinates. This method corresponds to MOVE instruction PAC language. Note 1) This function require to run RobSlave.pac			

The form and the meaning are specified as follows.

In case of VT_BSTR

・ If Comp = 1, 2

"[<@pass motion beginning displacement>] < pose >"

e.g.) "P1", "@P T100", "@E J520"

<pose> : "<variable type><variable number>" or
"[<variable type>](<element1>,<element2>,…)"
: <variable type> : One character either 'P', 'T' or 'J'
<variable number> : a decimal number
<element n> : an element of variable either 'P', 'T' or 'J'
P(<x>,<y>,<z>,<rx>,<ry>,<rz>,<fig>)
J(<j1>,<j2>,<j3>,<j4>,<j5>,<j6>,<j7>,<j8>)
T(<x>,<y>,<z>,<ox>,<oy>,<oz>,<ax>,<ay>,<az>,<fig>)

<@pass motion beginning displacement> : "@0", "@P", "@E", or "@<*value*>"

Example 1. Move 1, “@P P1", "NEXT" ‘ MOVE P, @P P1, NEXT

Example 2. Move 2, “P1” ‘ MOVE L, P1

Example 3. Move 1, “@0 J1” ‘ MOVE P, @0 J1

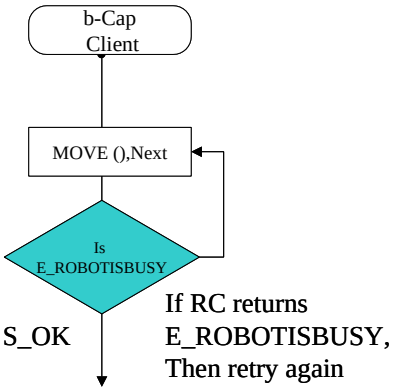
Example 4. Move 2, “@0 (307.1856,-157.8244,107.0714,160,0,0,1)”
‘ MOVE L,@0 P(307.1856,-157.8244,107.0714,160,0,0,1)

Example 5. Move 1, “@P J(60,50,40,30,20,10)”
‘ MOVE L,@P J(60,50,40,30,20,10)

If <NEXT option> is added, the robot proceeds to the next no-movement instruction without waiting for movement to finish.

If <NEXT option> is added, the robot proceeds to the next no-movement instruction without waiting for movement to finish.

When two or more motion method (e.g. Approach/Move/Drive or other commands that use RobSlave.PAC) is executed consecutively, the latter motion method returns “E_ROBOTISBUSY” (0x80010004) status until the preceding motion method execution ends. If application needs to execute Move method continuously, retry until the method returns S_OK.



The following table shows the PAC MOVE command supported by Move method.

Table 6-3 Move command list

Division	PAC command	Move method of b-CAP
MOVE P,…	MOVE P, P[n1]	Move 1, “P<n1>”
	MOVE P, @P P[n1]	Move 1, “@P P<n1>”
	MOVE P, @E P[n1]	Move 1, “@E P<n1>”
	MOVE P, T[n1]	Move 1, “T<n1>”
	MOVE P, @P T[n1]	Move 1, “@P T<n1>”
	MOVE P, @E T[n1]	Move 1, “@E T<n1>”
	MOVE P, J[n1]	Move 1, “J<n1>”
	MOVE P, @P J[n1]	Move 1, “@P J<n1>”
	MOVE P, @E J[n1]	Move 1, “@E J<n1>”
MOVE L,…	MOVE L, P[n1]	Move 2, “P<n1>”
	MOVE L, @P P[n1]	Move 2, “@P P<n1>”
	MOVE L, @E P[n1]	Move 2, “@E P<n1>”
	MOVE L, T[n1]	Move 2, “T<n1>”
	MOVE L, @P T[n1]	Move 2, “@P T<n1>”
	MOVE L, @E T[n1]	Move 2, “@E T<n1>”
	MOVE L, J[n1]	Move 2, “J<n1>”
	MOVE L, @P J[n1]	Move 2, “@P J<n1>”
	MOVE L, @E J[n1]	Move 2, “@E J<n1>”

< n1 >: integer 0-65535 as index number of variable

Communication sample 1 - MOVE 1, “P12”
This sample executes a move command “MOVE 1, “P12””.

<table>
<tr><td>Packet
TX</td><td>TX(Client->Server):
01 4E 00 00 00 39 0A 00 00 48 00 00 00 04 00 0A
00 00 00 03 00 01 00 00 00 01 00 00 00 0A 00 00
00 03 00 01 00 00 00 01 00 00 00 10 00 00 00 08
00 01 00 00 00 06 00 00 00 50 00 31 00 32 00 0A
00 00 00 08 00 01 00 00 00 00 00 00 00 00 04
<table>
<tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>The handle of robot
(See also Controller_GetRobot)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td></td><td>00 00 00 03 00 01 00 00 00 01 00 00 00 0A</td><td></td><td></td></tr>
<tr><td>lComp</td><td>Interpolation(MOVE P(= PTP))</td><td>VT_I4</td><td>1</td></tr>
<tr><td></td><td>00 03 00 01 00 00 00 01 00 00 00 0A 00 00</td><td></td><td></td></tr>
<tr><td>vntPose</td><td>Destination position</td><td>VT_BSTR</td><td>“P12”</td></tr>
<tr><td></td><td>00 01 00 00 00 06 00 00 00 50 00 31 00 32 00 10 00 00 00 08</td><td></td><td></td></tr>
<tr><td>bstrOpt</td><td>Option</td><td>VT_BSTR</td><td>Null String</td></tr>
<tr><td></td><td>00 00 00 08 00 01 00 00 00 00 00 00 00 00 0A</td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td>Packet
RX</td><td>RX(Server->Client):
01 10 00 00 00 39 0A 00 00 00 00 00 00 00 00 04
<table>
<tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>No-Args</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td>-</td><td></td><td></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Communication sample 2 - MOVE 2, “@P P12”, “NEXT”</td></tr>
<tr><td>This sample executes a move command “MOVE 2, “@P P12”, “NEXT””.</td></tr>
<tr><td>Packet
TX</td><td>TX(Client->Server):
01 5C 00 00 00 38 0A 00 00 48 00 00 00 04 00 0A
00 00 00 03 00 01 00 00 00 01 00 00 00 0A 00 00
00 03 00 01 00 00 00 02 00 00 00 16 00 00 00 08
00 01 00 00 00 0C 00 00 00 40 00 50 00 20 00 50
00 31 00 32 00 12 00 00 00 08 00 01 00 00 00 08
00 00 00 4E 00 45 00 58 00 54 00 04
<table>
<tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>The handle of robot
(See also Controller_GetRobot)</td><td>VT_I4</td><td>0x0000001</td></tr>
</table>
</td></tr>
</table>

		00 00 00 03 00 01 00 00 00 01 00 00 0A		
	lComp	Interpolation(MOVE L(= CP))	VT_I4	2
		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		
	vntPose	Destination position	VT_BSTR	“@P P12”
		00 01 00 00 00 0C 00 00 00 40 00 50 00 20 00 50 00 31 00 32 00 16 00 00 00 08		
	bstrOpt	Option	VT_BSTR	“NEXT”
		00 00 00 4E 00 45 00 58 12 00 00 00 08 00 54 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 38 0A 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

Communication sample 3 - MOVE 2,“@P P(544.2,-79.2,136.6,0,0,3.9,0)”, “NEXT”				
This sample executes a move command “MOVE 2, “@P P(X,Y,Z,,,Fig)”, “NEXT””.				
Packet TX	TX(Client->Server): 01 92 00 00 00 48 0A 00 00 48 00 00 00 04 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 0A 00 00 00 03 00 01 00 00 00 02 00 00 00 4C 00 00 00 08 00 01 00 00 00 42 00 00 00 40 00 50 00 20 00 50 00 28 00 35 00 34 00 34 00 2E 00 32 00 2C 00 2D 00 37 00 39 00 2E 00 32 00 2C 00 31 00 33 00 36 00 2E 00 36 00 2C 00 30 00 2C 00 30 00 2C 00 33 00 2E 00 39 00 2C 00 30 00 29 00 12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 0A		
	lComp	Interpolation(MOVE L(= CP))	VT_I4	2
		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		

	vntPose	Destination position	VT_BSTR	"@P P(544.2,-79.2,136.6,0,0,3.9,0)"
		00 01 00 00 00 42 00 00 4C 00 00 00 08 00 40 00 50 00 20 00 50 00 28 00 35 00 34 00 34 00 2E 00 32 00 2C 00 2D 00 37 00 39 00 2E 00 32 00 2C 00 31 00 33 00 36 00 2E 00 36 00 2C 00 30 00 2C 00 30 00 2C 00 33 00 2E 00 39 00 2C 00 30 00 29 00		
	bstrOpt	Option	VT_BSTR	"NEXT"
		00 01 00 00 00 08 00 00 12 00 00 00 08 00 4E 00 45 00 58 00 54 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 48 0A 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

6.3.5. Robot_Execute

Function	HRESULT Robot_Execute()			
Function ID	64			
Argument	[in]	long	hRobot	Handle of robot
	[in]	BSTR	bstrCommand	Command name
	[in]	VARIANT	vntParam	Parameter
	[out]	VARIANT	pVal	Result
Return Value	See Table 4-5 Given return codes			
Description	Execute the command of the robot "hRobot"			

In this function, commands can be executed by using the following command names for "bstrCommand".

Table 6-4 Implemented command list

Command	Parameter	Return value	Operation
Speed	VT_R4: Internal speed(0.1 to 100.0)	None	Internal speed is set. (Note: This function require to run RobSlave.pac)
ExtSpeed	VT_R4|ARRAY:	None	External speed and

	(<external speed>,<external accelertion> ,<external deceleration>) <external speed >=VT_R4:(0.1 to 100.0) <external acceleration>=VT_R4:(0.0001 to 100.0) <external deceleration>=VT_R4:(0.0001 to 100.0)		external acceleration and external deceleration are set. ExtAccel and ExtDecel are optional
Motor	VT_I2: State(1:ON / 0:OFF)	None	Motor ON/OFF Note that "Motor ON/OFF" is time-consuming command. It may take about 3 sec.
DefTool	VT_R4|ARRAY: (<ToolNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ >)	None	For define a tool coordinate in AutoMode (Note: This function require to run RobSlave.pac)
Defwork	VT_R4|ARRAY: (<WorkNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ >)	None	For define a work coordinate in AutoMode (Note: This function require to run RobSlave.pac)
DefArea	VT_R4|ARRAY: (<AreaNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ>,<VX>,<VY>,<VZ>, <IONo>,<P-Variable No>,<ErrorOut>, <Enable or Disable >)	None	For define and enable/disable a area in AutoMode (Note: This function require to run RobSlave.pac)
MotionComp	None	VT_I2: Mode(0:running / 1:done)	Get MotionComp (Note: This function require to run RobSlave.pac)

			(Note: This function require to run RobSlave.pac)
MotionSkip	None	None	MotionSkip (Note: This function require to run RobSlave.pac)
Interrupt	VT_I2: (0:OFF 1:ON)	None	INTERRUPT ON/OFF (Note: This function require to run RobSlave.pac)
J2P	<J type:POSEDATA>	<Ptype:VT_R4|VT_ARRAY>	J2P [3]
J2T	<J type:POSEDATA>	<Ttype:VT_R4|VT_ARRAY>	J2T [3]
P2J	<P type:POSEDATA>	<Jtype:VT_R4|VT_ARRAY>	P2J [3]
P2T	<P type:POSEDATA>	<Ttype:VT_R4|VT_ARRAY>	P2T [3]
T2J	<T type:POSEDATA>	<Jtype:VT_R4|VT_ARRAY>	T2J [3]
T2P	<T type:POSEDATA>	<Ptype:VT_R4|VT_ARRAY>	T2P [3]
TINV	<T type:POSEDATA>	<Ttype:VT_R4|VT_ARRAY>	TINV [3]
NORMTRN	<T type:POSEDATA>	<Ttype:VT_R4|VT_ARRAY>	NORMTRN [3]
Approach	VT_BSTR: Interpolation method(=1:P, 2:L), base pose,	None	APPROACH <Interpolation method>, <pose variable

[3] In the version 2.801 and the previous robot controller, The data type of the option parameter had to be specified as VT_VARIANT(VT_R4|VT_ARRAY) or VT_VARIANT(VT_BSTR). In this function, these data types of the option parameters are supported for the compatibility. Therefore in the version 2.802 and the later controller, these should be specified as VT_R4|VT_ARRAY or VT_BSTR like 'Communication sample 6-a – P2J'.

	[@pass] approach distance [,option =“NEXT”] e.g.) “1,P1,@P 100,NEXT” See section 6.3.5.1 for more detail information.		type><pose variable number>, <pass> <approach distance>[, NEXT] (Note: This function require to run RobSlave.pac)
Depart	VT_BSTR: Interpolation method(=1:P, 2:L), [@pass] depart distance, [,option =“NEXT”] e.g.) “1,@P 100,NEXT” See section 6.3.5.2 for more detail information.	None	DEPART <Interpolation method >, <pass> <depart distance>[, NEXT] (Note: This function require to run RobSlave.pac)
DriveAEx	VT_BSTR: [@pass] (<Axis No.1>,<value>), [(<Axis No.1>,<value>)], … [(<Axis No.8>,<value>)] [,option :VT_BSTR “NEXT”] e.g.) “@P (1,20),(2,30)”	None	DRIVEA <Pass start displacement> (<Axis1 number>, <Axis1 Coordinate>), (<Axis2 number>, <Axis2 Coordinate>),…(<Axis8 number>, <Axis8 Coordinate>) [,NEXT] (Note: This function require to run RobSlave.pac)
DriveEx	VT_BSTR [@pass] (<Axis No.1>,<value>), [(<Axis No.1>,<value>)], … [(<Axis No.8>,<value>)] [,option :VT_BSTR “NEXT”]	None	DRIVE <Pass start displacement> (<Axis1 number>, <Axis1 Coordinate>), (<Axis2 number>, <Axis2 Coordinate>),…(<Axis8

	e.g.) "@P (1,20),(2,30)"		number>, <Axis8 Coordinate>) [,NEXT] (Note: This function require to run RobSlave.pac)
UserExt	VT_R4|ARRAY: (<CommandNumber>, <Parameter1>,<Parameter2>, <Parameter3>,･･･,<Parameter9>) Or VT_I4: CommandNumber Or VT_VARIANT|ARRAY(VT_I4, VT_R4|ARRAY>): CommandNumber(VT_I4), Parameter1,Parameter2,Parameter3, …,Parameter9(VT_R4|ARRAY) See section 6.3.5.3 for more detail information.	VT_R4 | VT_ARRAY	UserExtension (Note: This function require to run RobSlave.pac)
slvChangeMode 3.000	VT_I2 or VT_I4: 0x0 : Stop SlaveMode 0x1 : Synchronization(P type) 0x2 : Synchronization (J type) 0x3 : Synchronization (T type) 0x101 : Asynchronization(P type) 0x102 : Asynchronization (J type) 0x103 : Asynchronization(T type)	None	Switch the Slave Mode. Note: When switch to the Slave mode, this command wait until stop of the robot motion. (The b-CAP Slave function)
slvMove 3.000	VT_R4|ARRAY : P/J/T type Or VT_R8|ARRAY : P/J/T type	<Current position(J type): VT_R4|ARRAY>	Execute "SlaveMove". The type of this argument (P/J/T) is specified by the "SlvChangeMode". (The b-CAP Slave function)
slvGetMode 3.000	None	<SlaveMode:VT_I2> = 0x0 : Stop slave mode 0x1 :Synchronization (P type) 0x2 : Synchronization	Get current SlaveMode. (The b-CAP Slave function)

		(J type) 0x3 : Synchronization (T type) 0x101 : Asynchronization(P type) 0x102 : Asynchronization (J type) 0x103 : Asynchronization(T type)	
GetSrvData Reserved	<DataNumber:VT_I4>	< Servo Data : VT_R4|VT_ARRAY >	Gets the internal servo data specified by <DataNumber> into <InternalServoData>.
SetSrvData Reserved	<VT_VARIANT|ARRAY>: (<DataNumber:VT_I4>, < Servo Data :VT_R4|TV_ARRAY>)	None	Sets the internal servo data specified by <DataNumber>.

Communication sample 1 – SPEED 50.0				
This sample changes the internal robot speed.				
Packet TX	TX(Client->Server): 01 44 00 00 00 5C 00 00 00 40 00 00 00 03 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 53 00 50 00 45 00 45 00 44 00 0A 00 00 00 04 00 01 00 00 00 00 00 48 42 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	Command string	VT_BSTR	“SPEED”
		00 08 00 01 00 00 00 0A 00 00 00 53 00 50 00 45 00 45 00 44 00 14 00 00		
	vntParam	Parameter of command	VT_R4	50.0
		00 48 42 0A 00 00 00 04 00 01 00 00 00 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 5C 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value

		Binary		
	No-Args	-	-	-
		-		

Communication sample 2 - EXTSPEED 50.0				
This sample changes the external speed and acceleration and deceleration.				
Packet TX	TX(Client->Server): 01 4A 00 00 00 8A 03 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 1A 00 00 00 08 00 01 00 00 00 10 00 00 00 45 00 58 00 54 00 53 00 50 00 45 00 45 00 44 00 0A 00 00 00 04 20 01 00 00 00 00 00 48 42 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	Command string	VT_BSTR	“EXTSPEED”
		00 08 00 01 00 00 00 10 00 00 00 45 00 58 00 54 00 53 00 50 00 45 00 45 00 44 00 1A 00 00		
	vntParam	Parameter of command	VT_R4	50.0
		00 48 42 0A 00 00 00 04 00 01 00 00 00 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 8A 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

Communication sample 3 - EXTSPEED 50.0,10.0, 3.0				
This command “EXTSPEED” can change the speed and acceleration and deceleration individually.				
Packet TX	TX(Client->Server): 01 52 00 00 00 5E 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 1A 00 00 00 08 00 01 00 00 00 10 00 00 00 45 00 58 00 54 00 53 00 50 00 45 00 45 00 44 00 12 00 00 00 04 20 03 00 00 00 00 00 48 42 00 00 20 41 00 00 40 40 04			

	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 **01 00 00 00** 0A		
	bstrCommand	Command string	VT_BSTR	“EXTSPEED”
		00 08 00 01 00 00 00 10 00 00 00 **45 00 58 00 54 00 53 00 50 00 45 00 45 00 44 00** 1A 00 00		
	vntParam	Parameter of command	VT_R4	50.0,10.0, 3.0
		20 03 00 00 00 **00 00 48 42 00 00 20 41 00 00 40 40** 12 00 00 00 04		
Packet RX	RX(Server->Client): 01 10 00 00 00 8A 00 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

Communication sample 4 – Motor ON/OFF				
This command turns on or off the motor.				
Packet TX	TX(Client->Server): 01 42 00 00 00 8B 03 00 00 40 00 00 00 03 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 **4D 00 4F 00 54** **00 4F 00 52 00** 08 00 00 00 02 00 01 00 00 00 **01** **00** 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 **01 00 00 00** 0A		
	bstrCommand	Command string	VT_BSTR	“MOTOR”
		00 08 00 01 00 00 00 0A 00 00 00 **4D 00 4F 00 54 00 4F 00 52 00** 14 00 00		
	vntParam	Parameter of command	VT_I2	0x0001

		08 00 00 00 02 00 01 00 00 00 01 00
Packet RX	RX(Server->Client): 01 10 00 00 00 8B 00 00 00 00 00 00 00 00 00 04	
	Name	Description / Binary	Type	Value
	No-Args	-	-	-
		-		

Communication sample 5 - MotionComp				
This sample gets value of MotionComp.				
Packet TX	TX(Client->Server): 01 4C 00 00 00 04 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 1E 00 00 00 08 00 01 00 00 00 14 00 00 00 4D 00 4F 00 54 00 49 00 4F 00 4E 00 43 00 4F 00 4D 00 50 00 06 00 00 00 01 00 01 00 00 00 04			
	Name	Description / Binary	Type	Value
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	Command string	VT_BSTR	“MOTIONCOMP”
		00 08 00 01 00 00 00 14 00 00 00 4D 00 4F 00 54 00 49 00 4F 00 4E 00 43 00 4F 00 4D 00 50 00 1E 00 00		
	vntParam	Option parameter	VT_NULL	-
		00 00 00 01 00 01 00 00 00 06		
Packet RX	RX(Server->Client): 01 1C 00 00 00 12 00 00 00 00 00 00 00 01 00 08 00 00 00 02 00 01 00 00 00 01 00 04			
	Name	Description / Binary	Type	Value
	Pval	Mode	VT_I2	1
		00 00 00 02 00 01 00 00 00 01 00 08		

Communication sample 6-a – P2J

This sample gets result value of P2J.

J2P,J2T,P2J,P2T,T2J,T2P,TINV,NORMTRN has same structure.

In the version 2.802 and the later robot controller, The data type of the option parameter should be specified as VT_BSTR or VT_R4|VT_ARRAY like below.

Packet TX

TX(Client->Server):

```
01 58 00 00 00 0A 00 00    00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00    00 01 00 00 00 10 00 00
00 08 00 01 00 00 00 06    00 00 00 50 00 32 00 4A
00 22 00 00 00 04 20 07    00 00 00 B5 56 28 44 F6
FF 80 43 00 00 96 43 00    00 00 00 00 00 00 00 45
B5 A6 42 00 00 00 00 04
```

Name	Description	Type	Value
	Binary		
hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
	00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
bstrCommand	Command string	VT_BSTR	“P2J”
	00 08 00 01 00 00 00 06 00 00 00 50 00 32 00 4A 10 00 00 00		
vntParam	Option parameter	VT_R4|ARRAY Note: VT_BSTR is also valid as the type, then Variable can be specified like ‘P1’.	673.3548 257.9997 300 0 0 83.35404 0
	22 00 00 00 04 20 07 00 00 00 B5 56 28 44 F6 FF 80 43 00 00 96 43 00 00 00 00 00 00 00 00 45 B5 A6 42 00 00 00 00		

Packet RX

RX(Server->Client):

```
01 2A 00 00 00 0E 00 00    00 00 00 00 00 01 00 16
00 00 00 04 20 04 00 00    00 00 00 90 C1 38 29 82
42 00 00 96 43 1A 18 11    42 04
```

Name	Description	Type	Value
	Binary		

	Pval	Result	VT_R4|ARRAY	-18 65.08051 300 36.27354
		00 00 00 04 20 04 00 00 00 00 00 90 C1 38 29 82 (16) 42 00 00 96 43 1A 18 11 42		

Communication sample 6-b – P2J				
This sample gets result value of P2J. J2P,J2T,P2J,P2T,T2J,T2P,TINV,NORMTRN has same structure.				
Packet TX	TX(Client->Server): 01 5E 00 00 00 0E 00 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 10 00 00 00 08 00 01 00 00 00 06 00 00 00 50 00 32 00 4A 00 28 00 00 00 0C 00 01 00 00 00 04 20 07 00 00 00 92 8C D2 43 06 1A 85 43 4C BB 47 44 F9 F1 AB 42 2A EF 08 42 78 3B 04 43 00 00 A0 40 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 (0A)		
	bstrCommand	Command string	VT_BSTR	“P2J”
		00 08 00 01 00 00 00 06 00 00 00 50 00 32 00 4A 00 (10 00 00)		
	vntParam	Option parameter	VT_VARIANT (VT_R4|ARRAY) Note:If VT_VARIANT (VT_BSTR) is used as a type, then Variable can be specified like ‘P1’.	421.0982, 266.2033 798.9265 85.9726 34.23356 5
		28 00 00 00 0C 00 01 00 00 00 00 04 20 07 00 00 00 92 8C D2 43 06 1A 85 43 4C BB 47 44 F9 F1 AB 42 2A EF 08 42 78 3B 04 43 00 00 A0 40		

<table>
<tr><td>Packet
RX</td><td colspan="3">RX(Server->Client):
01 32 00 00 00 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 1E
00 00 00 04 20 06 00 00 00 E0 FD EF 41 4A FD EF
41 B2 FD EF 41 BC FD EF 41 F2 FD EF 41 3B F9 EF
41 04</td></tr>
<tr><td rowspan="3"></td><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td colspan="3">Binary</td></tr>
<tr><td>pVal</td><td>Result value</td><td>VT_R4|ARRAY</td><td>29.99896
29.99898
29.99889
29.99889
29.999
29.99669</td></tr>
<tr><td colspan="4">1E
00 00 00 04 20 06 00 00 00 E0 FD EF 41 4A FD EF
41 B2 FD EF 41 BC FD EF 41 F2 FD EF 41 3B F9 EF
41 04</td></tr>
</table>

Communication sample 7 – DriveEX

This sample execute “DriveAEx” command. This sample execute below:

DriveAEx @P (1,10),(2,10),(3,10),(4,10),(5,10),(6,10),NEXT

And “DriveEx “also has same parameters.

Packet TX

TX(Client->Server):

```
01 B8 00 00 00 04 00 00   00 40 00 00 00 04 00 0A
00 00 00 03 00 01 00 00   00 01 00 00 00 1A 00 00
00 08 00 01 00 00 00 10   00 00 00 44 00 72 00 69
00 76 00 65 00 41 00 45   00 78 00 62 00 00 00 08
00 01 00 00 00 58 00 00   00 40 00 50 00 20 00 28
00 31 00 2C 00 31 00 30   00 29 00 2C 00 28 00 32
00 2C 00 31 00 30 00 29   00 2C 00 28 00 33 00 2C
00 31 00 30 00 29 00 2C   00 28 00 34 00 2C 00 31
00 30 00 29 00 2C 00 28   00 35 00 2C 00 31 00 30
00 29 00 2C 00 28 00 36   00 2C 00 31 00 30 00 29
00 12 00 00 00 08 00 01   00 00 00 08 00 00 00 4E
00 45 00 58 00 54 00 04
```

Name	Description	Type	Value
	Binary		
hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
	`00 00 00 03 00 01 00 00  00 01 00 00 0A`		
bstrCommand	Command string	VT_BSTR	“DriveAEx”
	`00 08 00 01 00 00 00 06  00 00 00 50 00 32 00 4A 10 00 00 00`		
bstrPosition	Destination position	VT_BSTR	“@P (1,10),(2,10),(3,10),(4,10),(5,10),(6,10)”

	`00 01 00 00 00 58 00 00  00 40 00 50 00 20 00 28`
	`00 31 00 2C 00 31 00 30  00 29 00 2C 00 28 00 32`
	`00 2C 00 31 00 30 00 29  00 2C 00 28 00 33 00 2C`
	`00 31 00 30 00 29 00 2C  00 28 00 34 00 2C 00 31`
	`00 30 00 29 00 2C 00 28  00 35 00 2C 00 31 00 30`
	`00 29 00 2C 00 28 00 36  00 2C 00 31 00 30 00 29`
	`00 62 00 00 00 08 00`		

bstrOption	Option parameter	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

RX(Server->Client):

```
01 10 00 00 00 04 00 00   00 00 00 00 00 00 00 04
```

Name	Description	Type	Value
	Binary		

	No Args.	-	-	-
		-		

Communication sample 8 – slvMove			
This sample execute “slvMove” command : SlvMove 12.6334, 4.920117, 233.132, 4.918625, 0, 0, 0, 0			
Packet TX	TX(Client->Server): 01 64 00 00 00 A0 08 00 00 40 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 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 26 00 00 00 04 20 08 00 00 00 66 22 4A 41 9A 71 9D 40 C8 21 69 43 61 65 9D 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04		
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 (0A)		
	bstrCommand	Command string	VT_BSTR	“slvMove”
		00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 (18 00 00) 00 4D 00 6F 00 76 00 65 00		
	vntParam	Option parameter	VT_R4 | ARRAY	12.6334, 4.920117, 233.132, 4.918625, 0, 0, 0, 0
		00 00 00 66 22 4A 41 9A 71 9D 40 C8 21 69 43 61 (26 00 00 00 04 20 08) 65 9D 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
Packet RX	RX(Server->Client): 01 3A 00 00 00 A0 08 00 00 00 00 00 00 01 00 26 00 00 00 04 20 08 00 00 00 33 FD 1F 41 99 F9 9F 40 4E 00 66 43 92 F3 9F 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04		
	Name	Description	Type	Value
		Binary		
	pVal	Current position(J type)	VT_R4 | ARRAY	9.9993162, 4.9992185, 230.00119, 4.9984827, 0, 0, 0, 0
		00 00 00 04 20 08 00 00 00 33 FD 1F 41 99 F9 9F (26) 40 4E 00 66 43 92 F3 9F 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

6.3.5.1. Approach

Approach is implemented as a command of the Robot_Execute()

The parameters are like below.

Argument	[in]	long	hRobot	Handle of robot
	[in]	BSTR	bstrCommand	Command name “Approach”
	[in]	BSTR	vntParam	Approach parameters in BSTR

The form and the meaning of Approach parameters are specified as follows.

Type of vntParam : VT_BSTR

<Interpolation 1 or 2>,<base pose>,<[@pass motion beginning displacement] Approach distance>, [NEXT]

e.g). "1, P10,@P 123.4 " ‘ Approach P,P10,@P 123.4

"2, P10,@E 123.4 ,NEXT" ‘ Approach L,P10,@E 123.4,NEXT

"1, (450,250,400,20,-10,170,5), 123" ‘ Approach P, (450,250,400,20,-10,170,5), 123,NEXT

<pose> : "<variable type><variable number>" or
"[<variable type>](<element1>,<element2>,…)"
<variable type> : One character either 'P', 'T' or 'J'
<variable number> : a decimal number
<element n> : an element of variable either 'P', 'T' or 'J'
P(<x>,<y>,<z>,<rx>,<ry>,<rz>,<fig>)
J(<j1>,<j2>,<j3>,<j4>,<j5>,<j6>,<j7>,<j8>)
T(<x>,<y>,<z>,<ox>,<oy>,<oz>,<ax>,<ay>,<az>,<fig>)

<@pass motion beginning displacement> : "@0", "@P", "@E", or "@<*value*>"

If <NEXT option> is added, the robot proceeds to the next no-movement instruction without waiting for movement to finish.

When two or more motion method (e.g. Approach/Move/Drive or commands that use RobSlave.PAC) is executed consecutively, the latter motion method returns “E_ROBOTISBUSY” (0x80010004) status until the preceding motion method execution ends. If application needs to execute Move method continuously, retry until the method returns S_OK.

6.3.5.2. Depart

Depart is implemented as a command of the Robot_Execute()

The parameters are like below.

Argument	[in]	long	hRobot	Handle of robot
	[in]	BSTR	bstrCommand	Command name “Depart”
	[in]	BSTR	vntParam	Depart parameters in BSTR

The form and the meaning of Depart parameters are specified as follows.

Type of vntParam : VT_BSTR

<Interpolation 1 or 2>,<[@pass motion beginning displacement] Depart distance>, [NEXT]

e.g). "1, @P 123.4 " ‘ Depart P,@P 123.4

"2, @E 123.4 ,NEXT" ‘ Depart L,@E 123.4,NEXT

"1, 123" ‘ Depart P, 123,NEXT

<@pass motion beginning displacement>= "@0", "@P", "@E"

If <NEXT option> is added, the robot proceeds to the next no-movement instruction without waiting for movement to finish.

When two or more motion method (e.g. Approach/Move/Drive or commands that use RobSlave.PAC) is executed consecutively, the latter motion method returns “E_ROBOTISBUSY” (0x80010004) status until the preceding motion method execution ends. If application needs to execute Move method continuously, retry until the method returns S_OK.

6.3.5.3. UserExtension

The user can enhance an original command by defining an additional command in the UserExtention program of the UserExtention.pac file, and describing the execution code corresponding to it.

Communication sample –SYSSTATE command by using UserExtension.
This sample describes how to use UserExt Following is a concrete code example when SYSSTATE command (Command No is 10055 (= 0x2747))are used as a user defined command.

<table>
<tr><td>Packet
TX</td><td>TX(Client->Server):
01 48 00 00 00 11 00 00 00 40 00 00 00 03 00 0A
00 00 00 03 00 01 00 00 00 01 00 00 00 18 00 00
00 08 00 01 00 00 00 0E 00 00 00 55 00 73 00 65
00 72 00 45 00 78 00 74 00 0A 00 00 00 03 00 01
00 00 00 47 27 00 00 04

<table><tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr><tr><td></td><td colspan="3">BInary</td></tr><tr><td>hRobot</td><td>The handle of robot
(See also Controller_GetRobot)</td><td>VT_I4</td><td>0x0000001</td></tr><tr><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 01 00 00 00 0A</td></tr><tr><td>bstrCommand</td><td>Command string</td><td>VT_BSTR</td><td>“UserExt”</td></tr><tr><td></td><td colspan="3">00 08 00 01 00 00 00 0E 00 00 00 18 00 00
00 72 00 45 00 78 00 74 00 55 00 73 00 65</td></tr><tr><td>CommandNo
and Parameters</td><td>Parameters</td><td>VT_I4</td><td>0x2747</td></tr><tr><td></td><td colspan="3">00 00 00 47 27 00 00 0A 00 00 00 03 00 01</td></tr></table></td></tr>
<tr><td>Packet
RX</td><td>RX(Server->Client):
01 42 00 00 00 11 00 00 00 00 00 00 00 01 00 2E
00 00 00 04 20 0A 00 00 00 00 90 01 45 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
BF 04

<table><tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr><tr><td></td><td colspan="3">Binary</td></tr><tr><td>pVal</td><td>Result</td><td>ARRAY | R4</td><td>2073
0
0
0
0
0
0
0
-1</td></tr><tr><td></td><td colspan="3">00 00 00 04 20 0A 00 00 00 00 90 01 45 00 00 00 2E
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
BF</td></tr></table></td></tr>
</table>

6.3.6. Robot_Change

Function	HRESULT Robot_Change()			
Function ID	66			
Argument	[in]	long	hRobot	Handle of robot
	[in]	BSTR	bstrName	Tool or user coordinate name
Return Value	See Table 4-5 Given return codes			
Description	Change the tool and user coordinate of the robot "hRobot".			
	Note 1) This function require to run RobSlave.pac.			

In this function, commands can be executed by using the following command names for "bstrName".

Table 6-5 Implemented command list

Name	Operation
"Tool=n" Or "Tooln"	Change the tool In this command, The number of the tool is included in bstrName, in this case, n=0 to 63. Note: This function require to run RobSlave.pac
"Work=n" Or "Workn"	Change the user coordinate In this command, The number of the tool is included in bstrName, in this case, n=0 to 7. Note: This function require to run RobSlave.pac

Communication sample 1 – ChangeTool into Tool 7				
This sample change the number of tool.				
Packet TX	TX(Client->Server): 01 36 00 00 00 98 04 00 00 42 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 16 00 00 00 08 00 01 00 00 00 0A 00 00 00 54 00 4F 00 4F 00 4C 00 37 00 04			
	Name	Description	Type	Value
		Binary		
	hRobot	The handle of robot (See also Controller_GetRobot)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrName	Name string	VT_BSTR	"TOOL7"

		16 00 00 00 08 00 01 00 00 00 0A 00 00 00 54 00 4F 00 4F 00 4C 00 37 00
Packet RX	RX(Server->Client): 01 10 00 00 98 04 00 00 00 00 00 00 00 00 00 04	
	Name	Description / Binary / Type / Value
	No-Args	- / - / - / -

6.4. Task object

Task object allows starting and stopping PAC program in the robot controller.

6.4.1. Controller_GetTask

Gets the handle of the task object specified by the task name.

Function HRESULT Controller_GetTask ()

Function ID 8

Argument				
	[in]	long	hController	Handle of controller
	[in]	BSTR	bstrName	Task name(PAC Program name)
	[in]	BSTR	bstrOption	Option character string (Not used)
	[out]	long	hTask	Handle of task

Return Value See Table 4-5 Given return codes

Description Gets the handle of task object “hTask”

See also Controller_connect

Communication sample 1 – Controller_GetTask(“RobSlave”)				
This function returns a handle of the task.				
Packet TX	TX(Client->Server): 01 4A 00 00 00 99 04 00 00 08 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 01 00 00 00 1A 00 00 00 08 00 01 00 00 00 10 00 00 00 52 00 6F 00 62 00 53 00 6C 00 61 00 76 00 65 00 0A 00 00 00 08 00 01 00 00 00 00 00 00 00 04			
	Name	Description / Binary	Type	Value
	hController	The handle of controller (See also Controller_Connect)	VT_I4	0x0000001
		0A 00 00 00 03 00 01 00 00 00 01 00 00 00		
	bstrName	Task name	VT_BSTR	“ROBSLAVE”

<table>
<tr><td rowspan="5"></td><td></td><td colspan="3">1A 00 00
00 08 00 01 00 00 00 10 00 00 00 52 00 6F 00 62
00 53 00 6C 00 61 00 76 00 65 00</td></tr>
<tr><td rowspan="3">bstrOption</td><td>bstrOption</td><td>VT_BSTR</td><td>Null string</td></tr>
<tr><td colspan="3">0A 00 00 00 08
00 01 00 00 00 00 00 00 00</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Packet
RX</td><td colspan="4">RX(Server->Client):
01 1E 00 00 00 99 04 00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00 00 3B 11 00 00 04</td></tr>
<tr><td>Name</td><td>Description</td><td>Type</td><td>Value</td></tr>
<tr><td></td><td>Binary</td><td></td><td></td></tr>
<tr><td rowspan="3">hTask</td><td>The handle of task</td><td>VT_I4</td><td>0x0000113b</td></tr>
<tr><td colspan="3">01 00 0A
00 00 00 03 00 01 00 00 00 3B 11 00 00</td></tr>
<tr><td colspan="4"></td></tr>
</table>

6.4.2. Task_Release

Function	HRESULT Task_Release ()			
Function ID	99			
Argument	[in]	long	hTask	Handle of task
Return Value	See Table 4-5 Given return codes			
Description	Releases the task object specified by “hTask”			

6.4.3. Task_GetVariable

Function	HRESULT Task_GetVariable ()			
Function ID	85			
Argument	[in]	long	hTask	Handle of controller
	[in]	BSTR	bstrName	Variable name
	[in]	BSTR	bstrOption	Option character string
	[out]	long	hVariable	Handle of variable
Return Value	See Table 4-5 Given return codes			
Description	Get the handle of Task system variable object “hVariable”			
See also	Variable_GetValue Variable_PutValue Controller_GetVariable Robot_GetVariable			

In this function, information about task can be obtained by using the following variable names.

Variable Name	Data type	Description	Attribute	
			get	put
@STATUS	VT_I4	State of task. 1: DORMANT 2: READY 3: RUN 4: WAIT 6: SUSPEND 0: NON_EXISTENT	∨	-
@PRIORITY	VT_I4	Priority of task.	∨	-
@LINE_NO	VT_I4	Line number of currently executing main program.	∨	-
@CYCLE_TIME	VT_I4	One cycle execution time of task.	∨	-

Communication sample 1 – Task_GetVariable (“@STATUS”)

This function returns the handle of a variable.

Packet TX

TX(Client->Server):

```
01 48 00 00 00 A0 04 00   00 55 00 00 00 03 00 0A
00 00 00 03 00 01 00 00   00 3B 11 00 00 18 00 00
00 08 00 01 00 00 00 0E   00 00 00 40 00 53 00 54
00 41 00 54 00 55 00 53   00 0A 00 00 00 08 00 01
00 00 00 00 00 00 00 04
```

Name	Description	Type	Value
	Binary		
hTask	The handle of task (See also Controller_Connect)	VT_I4	0x0000113b
	`00 00 00 03 00 01 00 00  00 3B 11 00 00  0A`		
bstrName	Variale name	VT_BSTR	“@STATUS
	`00 08 00 01 00 00 00 0E  00 00 00 40 00 53 00 54  18 00 00` `00 41 00 54 00 55 00 53  00`		
bstrOption	bstrOption	VT_BSTR	Null string
	`0A 00 00 00 08 00 01 00 00 00 00 00 00 00`		

Packet RX

RX(Server->Client):

```
01 1E 00 00 00 A0 04 00   00 00 00 00 00 01 00 0A
00 00 00 03 00 01 00 00   00 3B 11 40 00 04
```

Name	Description	Type	Value

		Binary		
	hVariable	The handle of task	VT_I4	0x0040113b
		00 00 00 03 00 01 00 00 00 3B 11 40 00 01 00 0A		

6.4.4. Task_Start

Function HRESULT Task_Start ()

Function ID 88

Argument

[in]	long	hTask	Handle of task
[in]	long	lMode	Start mode 1: 1 Cycle execution 2: Cyclic execution 3: 1 Step forward ~~4: 1 Step backword,~~ 5: Resume all
[in]	BSTR	bstrOpt	Option character string (Not used)

Return Value See Table 4-5 Given return codes

Description Start PAC program corresponding to the task “hTask”.

If this method is called with mode 5 (Resume), then all suspended programs in the robot controller are resumed.

See also Controller_GetTask

Task_Stop

Communication sample 1 – Task_Start - Cyclic execution				
This function starts a task.				
Packet TX	TX(Client->Server): 01 3A 00 00 00 A1 04 00 00 58 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 3B 11 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			
	Name	Description	Type	Value
		Binary		
	hTask	The handle of task (See also Controller_GetTask)	VT_I4	0x0000113b
		00 00 00 03 00 01 00 00 00 3B 11 00 00 0A		
	lMode	lMode(2: Cyclic execution)	VT_I4	2

		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		
	bstrOpt	bstrOption	VT_BSTR	Null string
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		
Packet RX	RX(Server->Client): 01 10 00 00 00 A1 04 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

6.4.5. Task_Stop

Function HRESULT Task_stop

Function ID 89

Argument

[in]	long	hTask	Handle of task
[in]	long	lMode	Stop mode 1: Suspend 2: Step stop 3: Cycle stop 4: Reset 5: Suspend all (Continuous stop)
[in]	BSTR	bstrOpt	Option character string (Not used)

Return Value See Table 4-5 Given return codes

Description Stop program corresponding to the task “hTask”.
If this method is called with mode 5 (Suspend), then all programs in the robot controller are suspended.

See also Controller_GetTask
Task_Start

Communication sample 1 – Task_stop - Cycle stop				
This function stops a task.				
Packet TX	TX(Client->Server): 01 3A 00 00 00 A5 04 00 00 59 00 00 00 03 00 0A 00 00 00 03 00 01 00 00 00 3B 11 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			
	Name	Description	Type	Value

		Binary		
	hTask	The handle of task (See also Controller_GetTask)	VT_I4	0x0000113b
		00 00 00 03 00 01 00 00 00 **01 00 00 00**		
	lMode	lMode(3: Cycle stop)	VT_I4	3
		00 03 00 01 00 00 00 **03 00 00 00**		
	bstrOpt	bstrOption	VT_BSTR	Null string
		00 01 00 00 00 00 00 00 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 A5 04 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

6.5. Variable object

Variable object allows to read or write variables.

6.5.1. Variable_Release

Function HRESULT Variable_Release ()

Function ID 111

Argument [in] long hVar Handle of Variable

Return Value See Table 4-5 Given return codes

Description Releases the variable object specified by “hVar”

6.5.2. Variable_GetValue

Gets the value of a variable that specified by the handle of the variable.

Function HRESULT Variable_GetValue ()

Function ID 101

Argument [in] long hVariable Handle of variable

[out] VARIANT pVal Value

Return Value See Table 4-5 Given return codes

Description Gets the value of the variable object specified by hVariable

See also Controller_GetVariable

Robot_GetVariable

Task_GetVariable

Communication sample 1 – Variable_GetValue (“I40”)				
This function returns a value of the variable “I40”.				
Packet TX	TX(Client->Server): 01 1E 00 00 00 AD 04 00 00 65 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 28 00 02 00 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable	VT_I4	0x0020028
		00 00 00 03 00 01 00 00 00 28 00 02 00 0A		
Packet RX	RX(Server->Client): 01 1E 00 00 00 AD 04 00 00 00 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 78 56 34 12 04			
	Name	Description	Type	Value
		Binary		
	pVal	The value of variable	VT_I4	0x12345678
		00 00 00 03 00 01 00 00 00 78 56 34 12 0A		

Communication sample 2 – Variable_GetValue (“D40”)				
This function returns a value of the variable “D40”.				
Packet TX	TX(Client->Server): 01 1E 00 00 00 09 00 00 00 65 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 28 00 04 00 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable	VT_I4	0x40028
		00 00 00 03 00 01 00 00 00 28 00 04 00 0A		
Packet RX	RX(Server->Client): 01 22 00 00 00 09 00 00 00 00 00 00 00 01 00 0E 00 00 00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09 40 04			
	Name	Description	Type	Value
		Binary		
	pVal	The value of variable	VT_R8	3.1415
		00 00 00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09 40 0E		

Communication sample 3 – Variable_GetValue (“P40”)

This function returns a value of the variable “P40”.

Packet TX

TX(Client->Server):

```
01 1E 00 00 00 11 00 00    00 65 00 00 00 01 00 0A
00 00 00 03 00 01 00 00    00 28 00 06 00 04
```

Name	Description	Type	Value
	Binary		
hVariable	The handle of variable	VT_I4	0x60028

```
                                                                  0A
00 00 00 03 00 01 00 00    00 28 00 06 00
```

Packet RX

RX(Server->Client):

```
01 36 00 00 00 11 00 00    00 00 00 00 00 01 00 22
00 00 00 04 20 07 00 00    00 00 00 20 41 00 00 A0
41 00 00 F0 41 00 00 20    42 00 00 48 42 00 00 70
42 00 00 A0 40 04
```

Name	Description	Type	Value
	Binary		
Pval	The value of variable “P40”	VT_R4|ARRAY	(10,20,30,40,50,60,5)

```
                                                                  22
00 00 00 04 20 07 00 00    00 00 00 20 41 00 00 A0
41 00 00 F0 41 00 00 20    42 00 00 48 42 00 00 70
42 00 00 A0 40
```

Communication sample 4 – Variable_GetValue (“@ERROR_DESCRIPTION”)

This sample procedure gets the value of controller varibale “@ERROR_DESCRIPTION”.

Packet TX

TX(Client->Server):

```
01 1E 00 00 00 19 00 00    00 65 00 00 00 01 00 0A
00 00 00 03 00 01 00 00    00 04 00 01 00 04
```

Name	Description	Type	Value
	Binary		
HVariable	The handle of the variable	VT_I4	0x10004

```
                                                                  0A
00 00 00 03 00 01 00 00    00 04 00 01 00
```

Packet RX

RX(Server->Client):

```
01 56 00 00 00 05 00 00    00 00 00 00 00 01 00 42
00 00 00 08 00 01 00 00    00 38 00 00 00 4A 00 31
00 20 00 65 00 6E 00 63    00 6F 00 64 00 65 00 72
00 20 00 64 00 61 00 74    00 61 00 20 00 6E 00 6F
00 74 00 20 00 72 00 65    00 63 00 65 00 69 00 76
00 65 00 64 00 04
```

	Name	Description	Type	Value
		Binary		
	pVal	Message string of a error. NOTE: This is BSTR.	VT_BSTR	"J1 encoder data not received"
		00 00 00 08 00 01 00 00 00 38 00 00 00 4A 00 31 01 00 42 00 20 00 65 00 6E 00 63 00 6F 00 64 00 65 00 72 00 20 00 64 00 61 00 74 00 61 00 20 00 6E 00 6F 00 74 00 20 00 72 00 65 00 63 00 69 00 76 00 65 00 64 00 04		

Communication sample 5 – Variable_GetValue ("@HIGH_CURRENT_POSITION")				
This sample procedure gets the value of robot varibale "@HIGH_CURRENT_POSITION".				
Packet TX	TX(Client->Server): 01 1E 00 00 00 20 00 00 00 65 00 00 00 01 00 0A 00 00 00 03 00 01 00 00 00 0B 02 03 00 04			
	Name	Description	Type	Value
		Binary		
	HVariable	The handle of the variable	VT_I4	0x30020B
		00 00 00 03 00 01 00 00 00 0B 02 03 00 0A		
Packet RX	RX(Server->Client): 01 3A 00 00 00 06 00 00 00 00 00 00 00 01 00 26 00 00 00 04 20 08 00 00 00 BF 29 57 C3 27 C3 AB 41 EC A5 73 43 00 00 00 00 00 00 00 00 9C 23 4F 43 00 00 00 00 00 00 D0 4A 04			
	Name	Description	Type	Value
		Binary		
	pVal	Current position(P type) with time stamp	VT_R4|ARRAY	-215.1631, 21.47029, 243.6481, 0, 0, 207.1391, 0, 6815744
		00 00 00 04 20 08 00 00 00 BF 29 57 C3 27 C3 AB 01 00 26 41 EC A5 73 43 00 00 00 00 00 00 00 00 9C 23 4F 43 00 00 00 00 00 00 D0 4A		

6.5.3. Variable_PutValue

Function HRESULT Variable_PutValue ()

Function ID 102

Argument [in] long hVariable Handle of variable

	[in]	VARIANT newVal	Value
Return Value	See Table 4-5 Given return codes		
Description	Writes a value of variable object specified by “hVariable”		
See also	Controller_GetVariable		
	Robot_GetVariable		
	Task_GetVariable		

Communication sample 1 – Variable_PutValue (“I40”)				
This function puts a new value into the variable “I40” that indicated by the handle.				
Packet TX	TX(Client->Server): 01 2C 00 00 00 34 03 00 00 66 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 28 00 02 00 0A 00 00 00 03 00 01 00 00 00 78 56 34 12 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable	VT_I4	0x0020028
		00 00 00 03 00 01 00 00 00 28 00 02 00 0A		
	newVal	New value for putting	VT_I4	0x12345678
		00 03 00 01 00 00 00 78 56 34 12 0A 00 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 34 03 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

Communication sample 2 – Variable_PutValue (“D40”)				
This function puts a new value into the variable “D40” that indicated by the handle.				
Packet TX	TX(Client->Server): 01 30 00 00 00 39 03 00 00 66 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 28 00 04 00 0E 00 00 00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09 40 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable	VT_I4	0x0040028

		00 00 00 03 00 01 00 00 00 28 00 04 00 00 0A		
	newVal	New value for putting	VT_R8	3.1415
		00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09 40 0E 00 00		
Packet RX	RX(Server->Client): 01 10 00 00 00 39 03 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

Communication sample 3 – Variable_PutValue (“P40”)				
This function puts a new value into the variable “P40” that indicated by the handle.				
Packet TX	TX(Client->Server): 01 44 00 00 00 3C 03 00 00 66 00 00 00 02 00 0A 00 00 00 03 00 01 00 00 00 28 00 06 00 22 00 00 00 04 20 07 00 00 00 00 00 20 41 00 00 A0 41 00 00 F0 41 00 00 20 42 00 00 48 42 00 00 70 42 00 00 A0 40 04			
	Name	Description	Type	Value
		Binary		
	hVariable	The handle of variable	VT_I4	0x0060028
		00 00 00 03 00 01 00 00 00 28 00 06 00 00 0A		
	newVal	New value for putting	VT_R4|ARRAY	(10,20,30,40,50,60,5)
		22 00 00 00 04 20 07 00 00 00 00 00 20 41 00 00 A0 41 00 00 F0 41 00 00 20 42 00 00 48 42 00 00 70 42 00 00 A0		
Packet RX	RX(Server->Client): 01 10 00 00 00 39 03 00 00 00 00 00 00 00 00 04			
	Name	Description	Type	Value
		Binary		
	No-Args	-	-	-
		-		

7. Outline of robot operation command execution

The execution of the robot motion commands of the Robot object (Move, Approach, Depart, Drive, Speed, Tool, Work) has been achieved in the following way. b-CAP server (is inside of the RC7M/J controller) communicate with a PAC program RobSlave (RobSlave.pac) running on the robot controller, and the PAC program executes specified PAC command. Controller global variables I[0], T[0] and T[1] are used for communication between b-CAP server and RobSlave. I[0] is uses to express the executing command status, e.g., running, completed, or error. T[0] is used to pass command and parameters from b-CAP server to RobSlave. T[1] is used to return value from RobSlave to b-CAP client program via b-CAP server.

Following diagram shows the execution procedure of robot motion command.

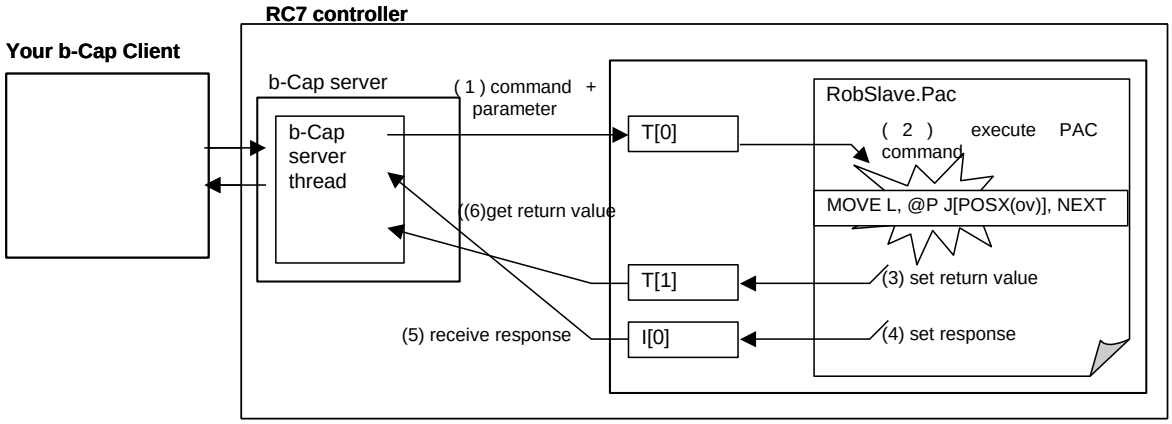


Figure 7-1 Execution procedure of robot motion command

【 attention 】

All global variables (I,F,D,V,P,J,T,S) from [0] to [9] have been reserved with the system.

Please do not access these variables in the user's program.