b-CAP communication specifications

Version 1.2.1

January 31, 2014

[Remarks]

[Revision history]

Date	Versions	Content
2006-08-02	1.0.0	First edition
2008-01-19	1.1.0	Added supplemental descriptions of UDP specification
2008-12-23	1.1.1	Misprint correction
2009-09-02	1.1.2	Added the setting procedure of Microsoft Network Monitor
2010-08-20	1.1.3	Added descriptions of COM usage
2010-10-22	1.1.4	Added descriptions of PPP
2012-03-23	1.1.5	Misprint correction
2013-01-30	1.2.0	Added descriptions of ZIP compressed packet
2014-01-31	1.2.1	Misprint correction
2014-17-11	1.2.2	Added descriptions of TCP timeout procedure

Contents

1. Introduction

This document specifies the communication protocol of b-CAP.

b-CAP is a communication protocol developed based on the concept of CAP in order to improve the transmission rate; therefore, b-CAP has the same features as CAP series, as follows:

- Having the same service structure as the object of CAO provider
- To invoke function, specify an object by the object ID
- Events are generated from the server-side by polling.

b-CAP is mounted as TCP stream communication. Since b-CAP packet does not contain check code, an error-free protocol is required at the lower-layer protocol.

(Note)

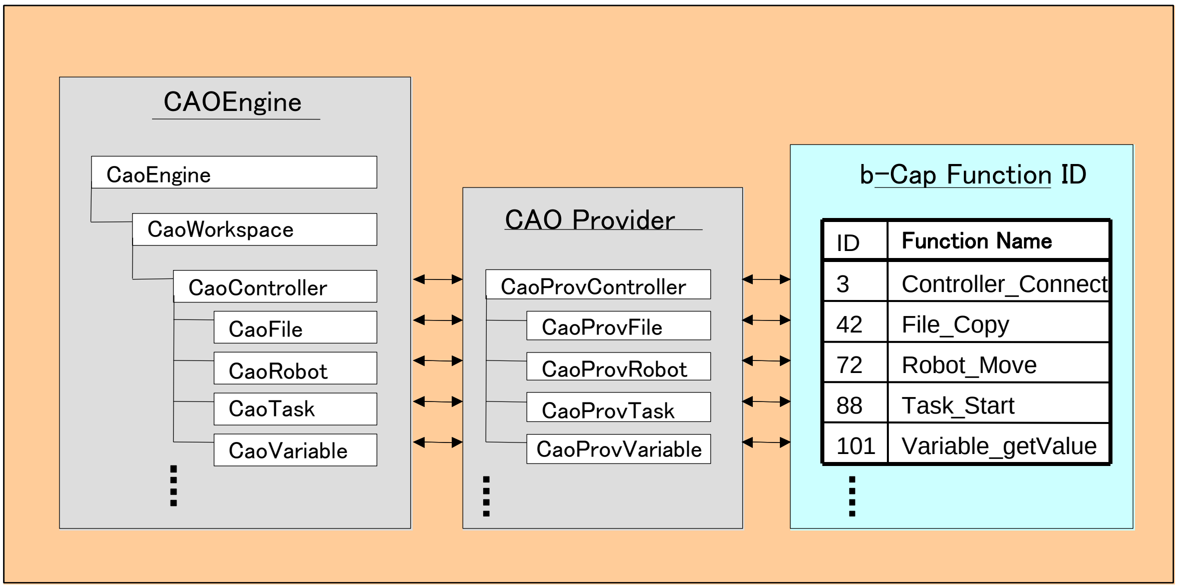
For b-CAP/COM[1], add CRC to the packet. For details, refer to 3.1 .

For b-CAP/UDP and b-CAP/COM, retry processing needs to be installed in the b-CAP client and server application side.

[1] For RS232C communication, b-CAP/COM equips CRC expediently. However, it is recommended to install an error-free protocol, such as PPP (Point to Point Protocol) to use b-CAP with RS232C. Using PPP will facilitate infrared communication or modem connection, not only RS232C.

2. Structure of b-CAP

The figure below shows the service structure of b-CAP that is similar to the object model of CAO provider. One b-CAP message corresponds to one service (function).



ID	Function Name
3	Controller_Connect
42	File_Copy
72	Robot_Move
88	Task_Start
101	Variable_getValue

Figure 2-1 Structure of b-CAP

b-CAP consists of two programs: b-CAP client which requests a service, and b-CAP server which executes a service and returns the result.

b-CAP client creates a request message that stores information necessary for the service required, transmits it to the server side, and then receives the response message to confirm the execution result.

b-CAP server receives the request message from the client, and then executes the service corresponding to function ID. Once the service has been executed, the result and value will be stored into the response message, and then it will be sent to the client side.

For details of the procedure for client and server processing, refer to section 4.

The figure below shows an example of b-CAP connection.

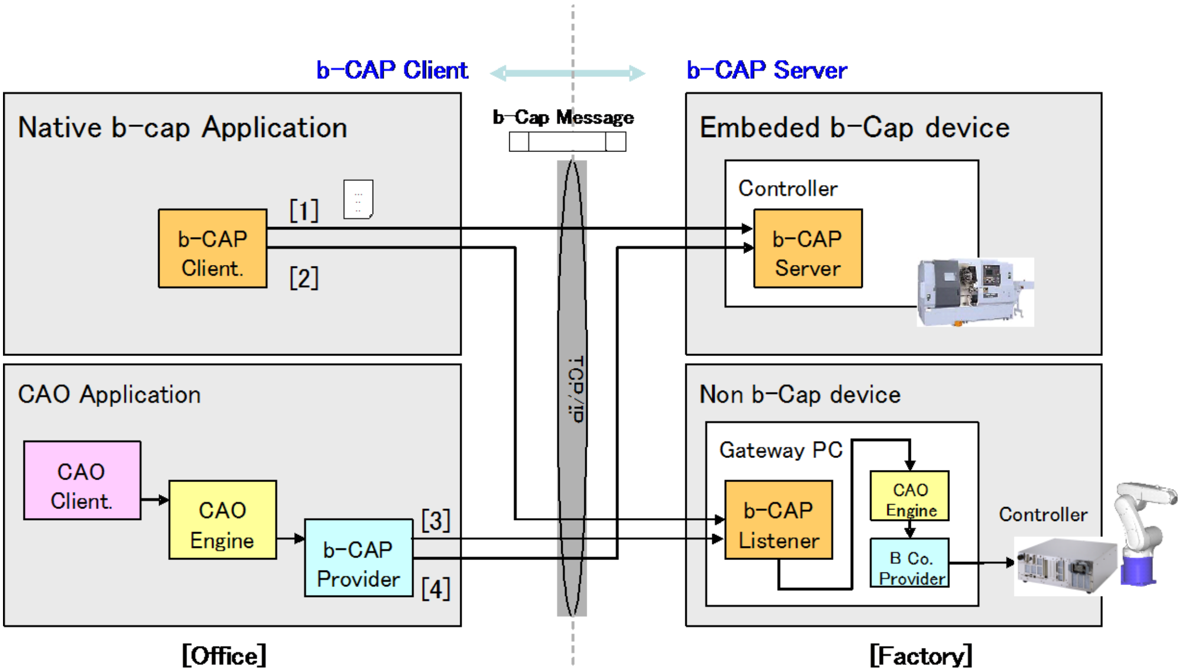


Figure 2-2 Example of connecting b-CAP

3. Packet structure

3.1. Packet structure

The figure below shows the structure of a b-CAP message.

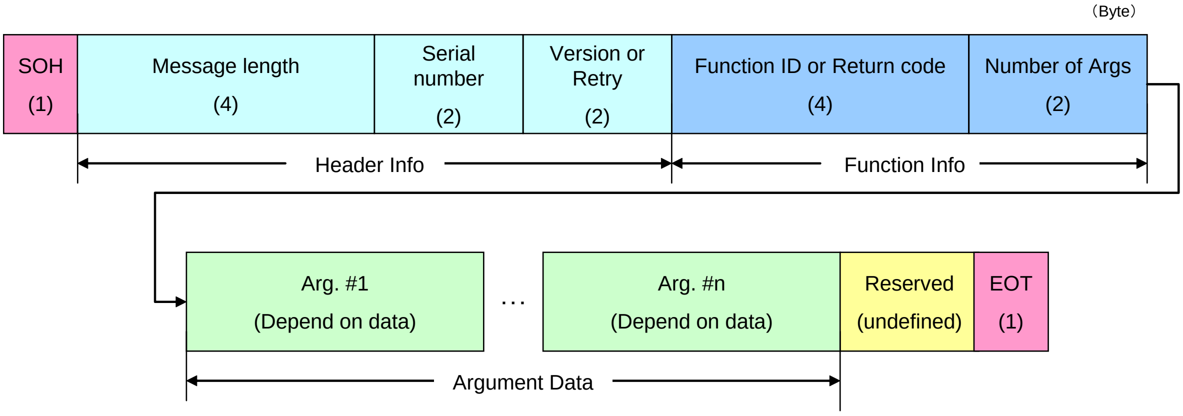


Figure 3-1 Packet structure

The below explains each elements of a packet. The memory image of each data is stored by using Intel convention (little endian).

Header
The start code applied to the top of packet. Use SOH(0x01).

Message length
Data length of the entire packet. Store unsigned long integer (DWORD).
Store the length from the header to the terminator. (For UDP, 504 bytes is the maximum)

Serial number
Serial number of a message. Store unsigned short integer (WORD). The value ranges from one to any figure of WORD_MAX. This value should be the same between the request message and the response message.

Version
Store the protocol version
This value must be "1".
Use this information only with b-CAP/TCP.

Retry
The serial number before the start of retry processing is embedded
Use this information only with b-CAP/UDP or with b-CAP/COM.

Function ID
ID that identifies a call function. Store unsigned long integer (DWORD).
Use only at the request message. (Refer to 3.3.)

Return code
Execution result code of call function. Store unsigned long integer (DWORD).
Use only at the response message. (Refer to 3.4.)

Number of Args
Number of arguments of call function or numbers of output variables of call function. Store unsigned short integer (WORD).

Argument #n
The argument part of N-th argument. Store the argument as VARIANT type image. (Refer to 3.2)

Reserved	System-reserved area. The data length of this area is undefined.
Terminator	End code applied at the end of packet. Use EOT (0x04).

As for b-CAP, the content of "Version or Retry (see Figure 1-3)" and "Reserved" differ depending on the communication device.

The following describes the packet structure in each device.

3.1.1. b-CAP/TCP packet structure

In b-CAP/TCP, parts of packet structures are used as follows.

Version or Retry	>	Version
Reserved	>	Mode

b-CAL/TCP can store data from Function Info to Argument Data by compressed with ZIP format as well.

You can check the packet structure state (compressed or uncompressed) by checking the value of "Mode".

The following figures show the b-CAP/TCP packet structures that are uncompressed and compressed.

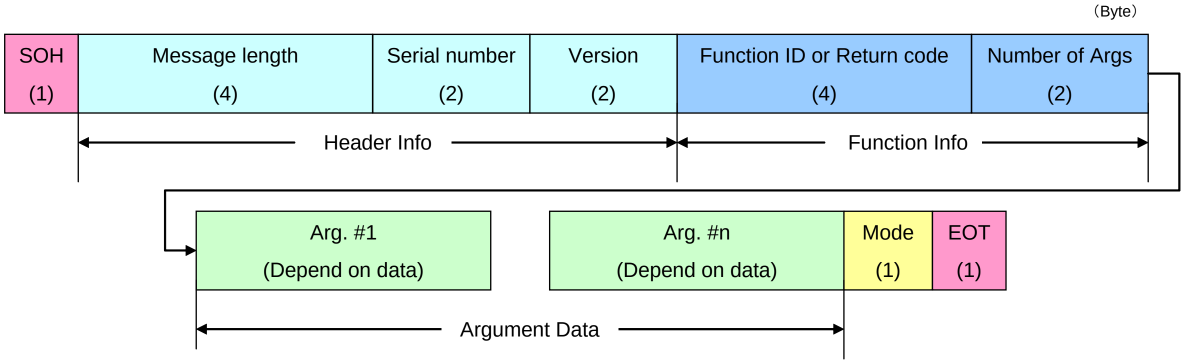


Figure 3-2 b-CAP/CTP packet structure

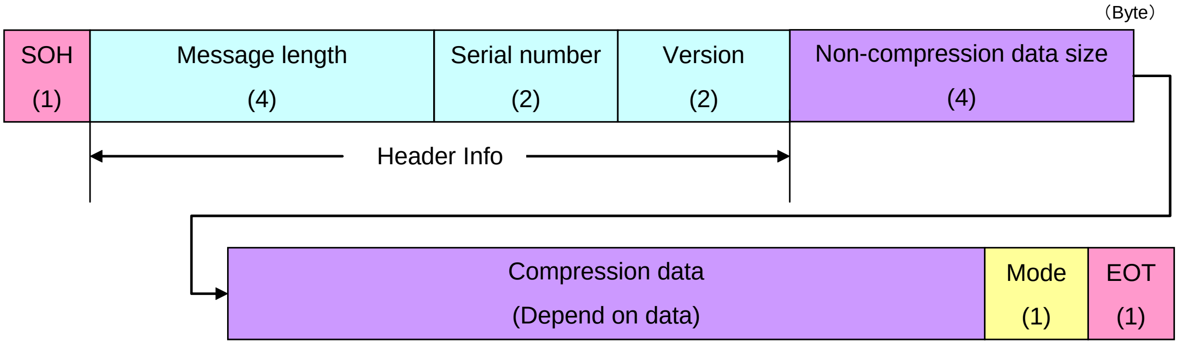


Figure 3-3 b-CAP/TCP packet structure (Compressed)

The following explains the difference between the basic packet elements and b-CAP/TCP packet elements.

Version Enter “1” as a protocol version

Uncompressed data size Data size of when compressed data information is unpacked

Compressed data Binary data where the range from the Function Info to Argument Data are compressed.

Mode State of packet

0 Normal packet

1 ZIP-compressed packet

3.1.2. b-CAP/UDP packet structure

In b-CAP/UDP, parts of packet structures are used as follows.

Version or Retry > Retry

Reserved > None

In b-CAP/UDP, retry packets can be used.

“Retry” stores the value of “Serial Number” of the initial packet. Therefore, at the first time, the value of “Serial Number” corresponds to “Retry”.

The number of maximum retry is determined by the client application and the server application.

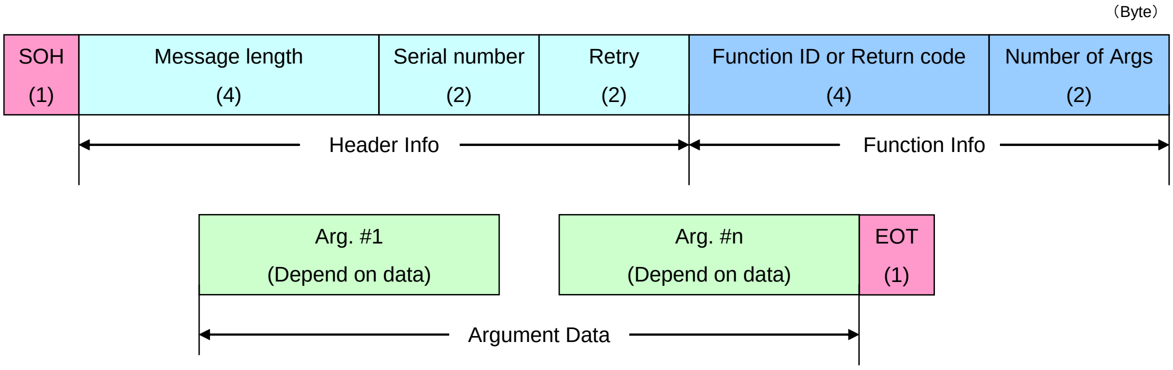


Figure 3-4 b-CAP/UDP packet structure

The following explains the difference between the basic packet elements and b-CAP/UDP packet elements.

Retry The serial number before the start of retry processing is embedded.

Reserved This area is not used in b-CAP/UDP.

Do not add this byte.

3.1.3. b-CAP/COM packet structure

In b-CAP/COM, parts of basic packet structure are used as follows.

Version or Retry	>	Retry
Reserved	>	CRC

In b-CAP/COM, retry packets can be used.

"Retry" stores the value of "Serial Number" of the initial packet. Therefore, at the first time, the value of "Serial Number" is corresponding to "Retry".

The number of maximum retry is determined by the client application and the server application.

In b-CAP/COM, CRC must be added.

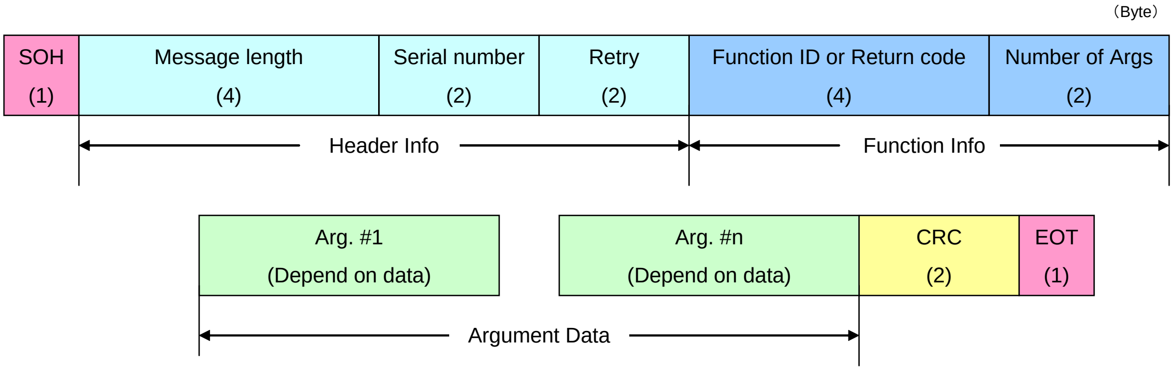


Figure 3-5 b-CAP/COM packet structure

The following explains the difference between the basic packet elements and b-CAP/COM packet element.

Retry The serial number before the start of retry processing is embedded.

CRC Store CRC where the area from the Header Info to Argument Data

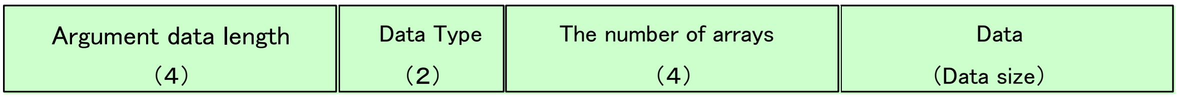
The calculation condition of CRC is shown below.

CRC type	:	CRC-CCITT
Initial value	:	0xFFFF
Output XOR	:	0x0000
Direction of bit transfer	:	Left
Input bit inversion	:	None
Output bit inversion	:	None

3.2. Argument Data part structure

The length of Argument Data part varies depending on the data type to state multiple data types.

The structure of the Argument Data part is shown as follows.



The number in parentheses is data size(byte)

Argument data length	Data Type	The number of arrays	Data
(4)	(2)	(4)	(Data size)

Figure 3-6 Structure of Argument Data part

As the figure above shows, the Argument Data part is composed of the Data Type and the Number of arrays as a common part. As for the Data part, the structure is determined based on information on the common part.

The below explains each element of the Argument Data part.

Argument data length Data type, number of elements, total bytes of data. Store unsigned long integer (DWORD). For about size of each data type, see Table 1.

Data type Data type of argument. Store unsigned short integer (WORD). For about available data type, see Table 1.

Number of elements Number of elements of arguments. Store unsigned long integer (DWORD). This should be “1” when VT_ARRAY is not used for the data type.

Data Data of argument. Available data size differs depending on the data type. For the size of each data, see Table 1. For the data structure stored in the Data part, see Figure 3-7.

Table 1 Available data type

Data type	Value	Size (Byte)	Description
VT_EMPTY	0	0	Empty data
VT_NULL	1	0	NULL value
VT_ERROR	10	4	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 the character) ×2+4	Character string type The first four bytes are occupied by the character string length. The character string is stored with Unicode behind the character string length.
VT_VARIANT	12	-	Variant type The same one as the Argument Data part is stored. This data type can be used only at VT_ARRAY.
VT_ARRAY	0x2000	-	Array This is specified by the logical sum with other data types. The data of the specified type is continuously stored. Only one dimension array can be stored.

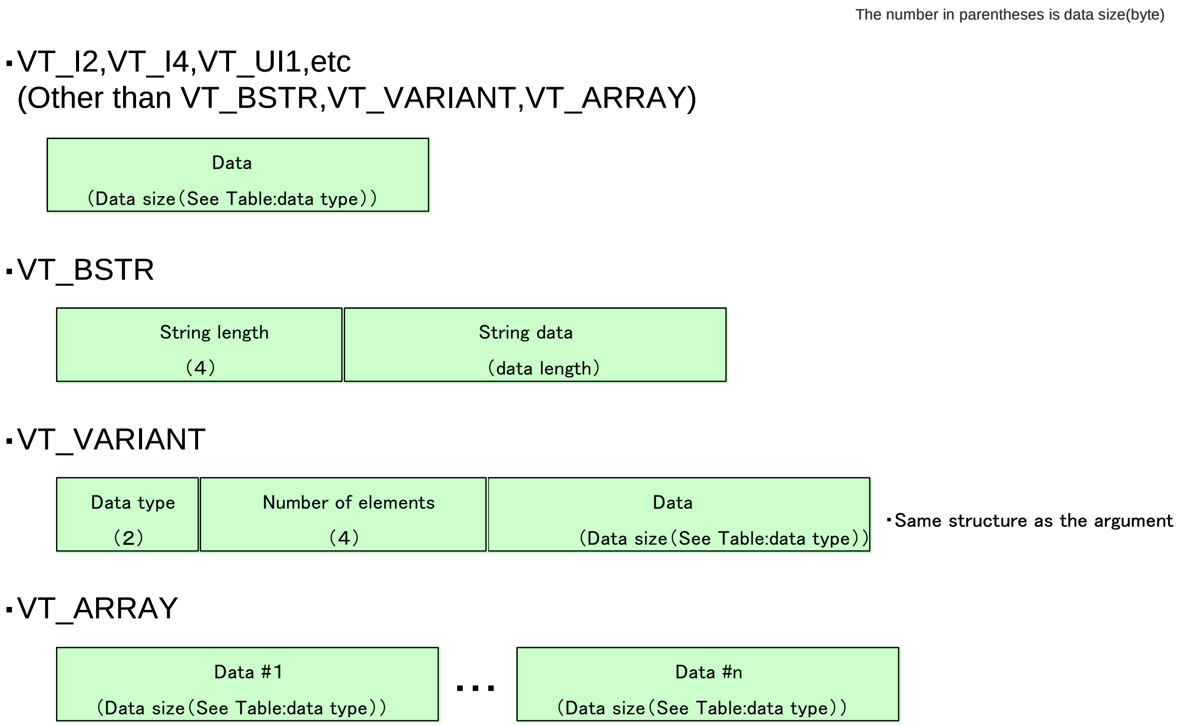

The number in parentheses is data size(byte)
・VT_I2,VT_I4,VT_UI1,etc
(Other than VT_BSTR,VT_VARIANT,VT_ARRAY)
Data
(Data size(See Table:data type))
・VT_BSTR
String length
(4)
String data
(data length)
・VT_VARIANT
Data type
(2)
Number of elements
(4)
Data
(Data size(See Table:data type))
・Same structure as the argument
・VT_ARRAY
Data #1
(Data size(See Table:data type))
・・・
Data #n
(Data size(See Table:data type))

Figure 3-7 Structure of data

3.3. Function ID

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

Table 1 Function ID assignment

Function ID	Description
1 to 137	Predetermined function
138 to 255	Reservation area
256 or higher	User function

To use functions other than predetermined function, assign any functions into user functions.

The below lists predetermined functions of b-CAP.

Table 2 Predetermined function list

Function ID	Function name	Description
1	Service_Start	Start server service
2	Service_Stop	Stop server service
3	Controller_Connect	Connect with controller
4	Controller_Disconnect	Disconnect from controller
5	Controller_GetExtension	Acquire a controller's extension board
6	Controller_GetFile	Acquire a controller's file
7	Controller_GetRobot	Acquire a controller's robot
8	Controller_GetTask	Acquire controller's task
9	Controller_GetVariable	Acquire a controller's variable
10	Controller_GetCommand	Acquire a controller's command
11	Controller_GetExtensionNames	Acquire a list of controller's extension board names
12	Controller_GetFileNames	Acquire a list of controller's file names
13	Controller_GetRobotNames	Acquire a list of controller's robot names
14	Controller_GetTaskNames	Acquire a list of controller's task names
15	Controller_GetVariableNames	Acquire a list of controller's variable identifier
16	Controller_GetCommandNames	Acquire a list of controller's command names
17	Controller_Execute	Execute controller's extension function
18	Controller_GetMessage	Acquire a controller's event message
19	Controller_GetAttribute	Acquire a controller's attribute value
20	Controller_GetHelp	Acquire controller's help character string

21	Controller_GetName	Acquire a controller name
22	Controller_GetTag	Acquire controller's tag information
23	Controller_PutTag	Set controller's tag information
24	Controller_GetID	Acquire a controller ID
25	Controller_PutID	Set a controller ID
26	Extension_GetVariable	Acquire a variable of extension board
27	Extension_GetVariableNames	Acquire a list of variable identifier of extension board
28	Extension_Execute	Execute extension function of extension board
29	Extension_GetAttribute	Acquire an attribute value of extension board
30	Extension_GetHelp	Acquire help character string of extension board
31	Extension_GetName	Acquire an extension board name
32	Extension_GetTag	Acquire tag information on extension board
33	Extension_PutTag	Set tag information on extension board
34	Extension_GetID	Acquire an extension board ID
35	Extension_PutID	Set an extension board ID
36	Extension_Release	Release an extension board
37	File_GetFile	Acquire subordinate files
38	File_GetVariable	Acquire variables of file
39	File_GetFileNames	Acquire a list of subordinate files
40	File_GetVariableNames	Acquire a list of variable names of files
41	File_Execute	Execute an extension function of file
42	File_Copy	Copy a file
43	File_Delete	Delete a file
44	File_Move	Move a file
45	File_Run	Execute a file
46	File_GetDateCreated	Acquire the date of file’s creation
47	File_GetDateLastAccessed	Acquire the final access date of file
48	File_GetDateLastModified	Acquire the last modified date and time of file
49	File_GetPath	Acquire a file path
50	File_GetSize	Acquire file size
51	File_GetType	Acquire a file type
52	File_GetValue	Acquire file contents
53	File_PutValue	Set file contents
54	File_GetAttribute	Acquire a file attribute

55	File_GetHelp	Acquire help character string of file
56	File_GetName	Acquire a file name
57	File_GetTag	Acquire tag information on file
58	File_PutTag	Set tag information on file
59	File_GetID	Acquire a file ID
60	File_PutID	Set a file ID
61	File_Release	Release a file
62	Robot_GetVariable	Acquire variables of robot
63	Robot_GetVariableNames	Acquire a list of variable identifier of robot
64	Robot_Execute	Execute an extension function of robot
65	Robot_Accelerate	Execute ACCEL command of robot
66	Robot_Change	Execute CHANGE command of robot
67	Robot_Chuck	Execute GRASP command of robot
68	Robot_Drive	Execute DRIVE command of robot
69	Robot_GoHome	Execute GOHOME command of robot
70	Robot_Halt	Execute HALT command of robot
71	Robot_Hold	Execute HOLD command of robot
72	Robot_Move	Execute MOVE command of robot
73	Robot_Rotate	Execute ROTATE command of robot
74	Robot_Speed	Execute SPEED/JSPEED command of robot
75	Robot_Unchuck	Execute REELASE command of robot
76	Robot_Unhold	Release HOLD command of robot
77	Robot_GetAttribute	Acquire attribute value of robot
78	Robot_GetHelp	Acquire help character string of robot
79	Robot_GetName	Acquire a robot name
80	Robot_GetTag	Acquire tag information on robot
81	Robot_PutTag	Set tag information on robot
82	Robot_GetID	Acquire a robot ID
83	Robot_PutID	Set a robot ID
84	Robot_Release	Release a robot
85	Task_GetVariable	Acquire a variable of task
86	Task_GetVariableNames	Acquire a list of variable identifier of task
87	Task_Execute	Execute an extension function of task
88	Task_Start	Start a task
89	Task_Stop	Stop a task

90	Task_Delete	Delete a task
91	Task_GetFileName	Acquire a former file name of task
92	Task_GetAttribute	Acquire task attribute
93	Task_GetHelp	Acquire help character string of task
94	Task_GetName	Acquire a task name
95	Task_GetTag	Acquire tag information on task
96	Task_PutTag	Set tag information on task
97	Task_GetID	Acquire a task ID
98	Task_PutID	Set a task ID
99	Task_Release	Release a task
100	Variable_GetDateTime	Acquire time stamp of variable
101	Variable_GetValue	Acquire a variable value
102	Variable_PutValue	Set a variable value
103	Variable_GetAttribute	Acquire attribute value of variable
104	Variable_GetHelp	Acquire help character string of variable
105	Variable_GetName	Acquire a variable name
106	Variable_GetTag	Acquire tag information on variable
107	Variable_PutTag	Set tag information on variable
108	Variable_GetID	Acquire a variable ID
109	Variable_PutID	Set a variable ID
110	Variable_GetMicrosecond	Acquire the time stamp (millisecond) of variable
111	Variable_Release	Release a variable
112	Command_Execute	Execute a command
113	Command_Cancel	Cancel a command
114	Command_GetTimeout	Acquire a time-out time of command
115	Command_PutTimeout	Set a time-out time of command
116	Command_GetState	Acquire command state
117	Command_GetParameters	Acquire command parameters
118	Command_PutParameters	Set command parameters
119	Command_GetResult	Acquire an execution result of command
120	Command_GetAttribute	Acquire attribute value of command
121	Command_GetHelp	Acquire help character string of command
122	Command_GetName	Acquire a command name
123	Command_GetTag	Acquire tag information on command
124	Command_PutTag	Set tag information on command

125	Command_GetID	Acquire a command ID
126	Command_PutID	Set a command ID
127	Command_Release	Release a command
128	Message_Reply	Reply an event message
129	Message_Clear	Clear an event message
130	Message_GetDateTime	Acquire a time stamp of event message
131	Message_GetDescription	Acquire an explanation of event message
132	Message_GetDestination	Acquire a destination of event message
133	Message_GetNumber	Acquire a message number of event message
134	Message_GetSerialNumber	Acquire a serial number of event message
135	Message_GetSource	Acquire a source of event message
136	Message_GetValue	Acquire a value of event message
137	Message_Release	Release an event message

3.4. Return code

In b-CAP, the return code is assigned as follows.

Table 3 Return code assignment

Return code	Description
0x00000000 to 0x8000FFFF	Predetermined return code. Reservation area
0x80010000 to 0x8001FFFF	User definition error

When you create an error code that is other than the “Predetermined error code” shown below, it is recommended to assign any error code within the range of “User definition error” on the table above, to distinguish the error from a system error or a user error.

Table 4 Predetermined return code list

Return code	Error	Description
0x00000000	S_OK	Terminated normally
0x80004001	E_NOTIMPL	Not implemented
0x80004004	E_ABORT	Function processing was interrupted
0x80004005	E_FAIL	Function processing failed
0x80070005	E_ACCESSDENIED	Access failed
0x80070006	E_HANDLE	Illegal handle
0x8007000E	E_OUTOFMEMORY	Memory shortage

0x80070057	E_INVALIDARG	Illegal argument.
0x8000FFFF	E_UNEXPECTED	A fatal error occurred.

4. Communication procedure

4.1. Communication sequence

In b-CAP, there are two types of communication specifications; TCP and UDP. For both specifications, the sequence of b-CAP always starts with the request packet transmission from the client. The server side executes the command (function) corresponding to the request packet, and returns the execution result to the client by putting it to the response packet.

In one session, once a request message is sent, b-CAP must keep waiting to receive a response message in order to synchronize a b-CAP client status. To use multiple request messages, it is recommended to send them through different sessions.

In the server side, there is no regulation with regard to the interval between the request packet reception and the response transmission. Therefore, note that if server side process takes longer than the timeout period of the client side, a timeout may occur on the client side.

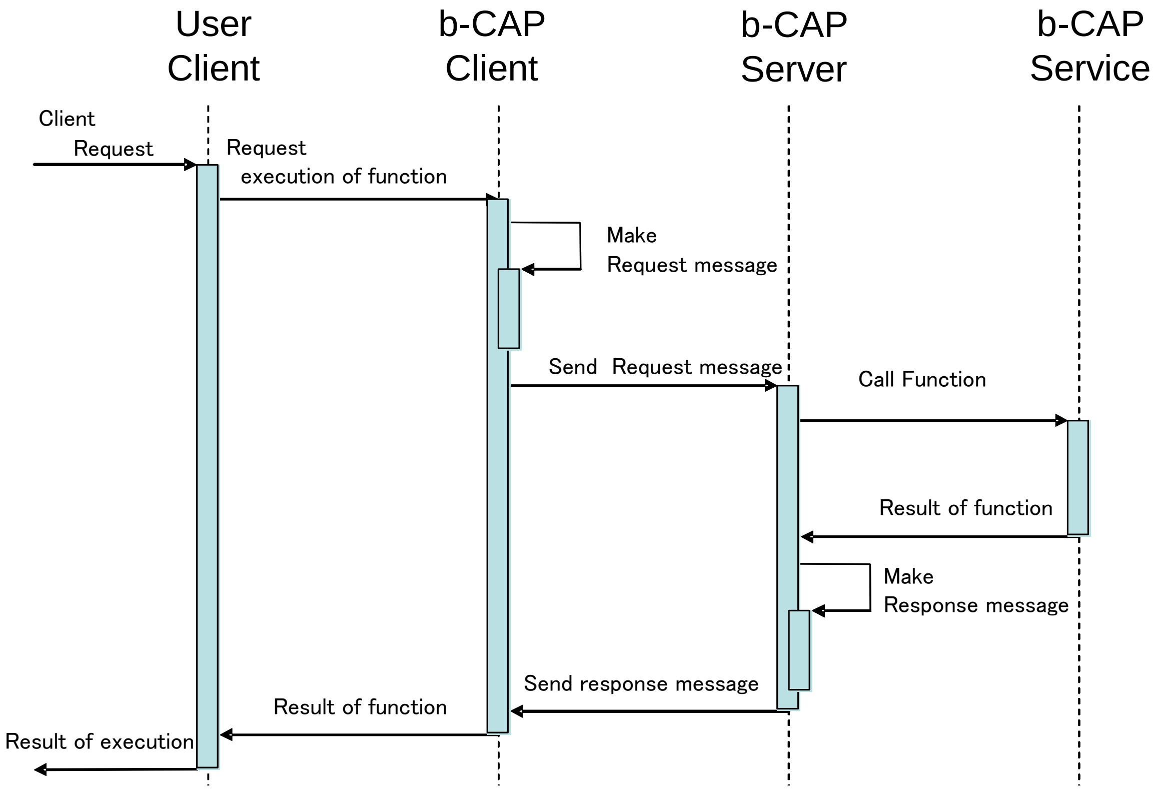


Figure 4-1 Communication sequence

4.2. b-CAP/TCP retry sequence

For b-CAP/TCP, it is not necessary to include the retry processing explicitly in the client program because the retry processing is processed by the TCP protocol stack. However, in the default setting of some OS, the retry time is set relatively long (few hundred milliseconds to few seconds), therefore, it may be inconvenient for some applications. In this case, it is necessary to create a program that intentionally generates a timeout by setting short timeout period explicitly in order to perform retry processing (or error processing) even for TCP. For such application, it is another solution to use UDP which will be explained in the next session.

4.3. b-CAP/UDP retry sequence

For b-CAP/UDP, retry processing shall be implemented in the client/server application side, as shown below.

[Client side]

(1) b-CAP/UDP sends a serial number with counting it up when resending.
=> This step is to discard the response of the previous request that has been received immediately after the resending, when the client assumes that no response is arrived (serial number discrepancy)

(2) At the same time, the initial serial number is stored into the reservation area of b-CAP packet.

[Server side]

(1) If a serial number is in the reservation area, the data is recognized as the retry packet. To allow multiple connections, you need to store the last serial number and its results for every destinations (IP + Port).

(2) In retry packet processing, if the retry packet's serial number is identical with the serial number which has been executed immediately before, the latest result is copied and the copy is returned, without retrying the command, in order to prevent executing a command twice. The serial number of the result is updated to that of the retry packet in this timing.

4.4. Communication procedure of server

The outline of the communication procedure of a server is shown as follows.

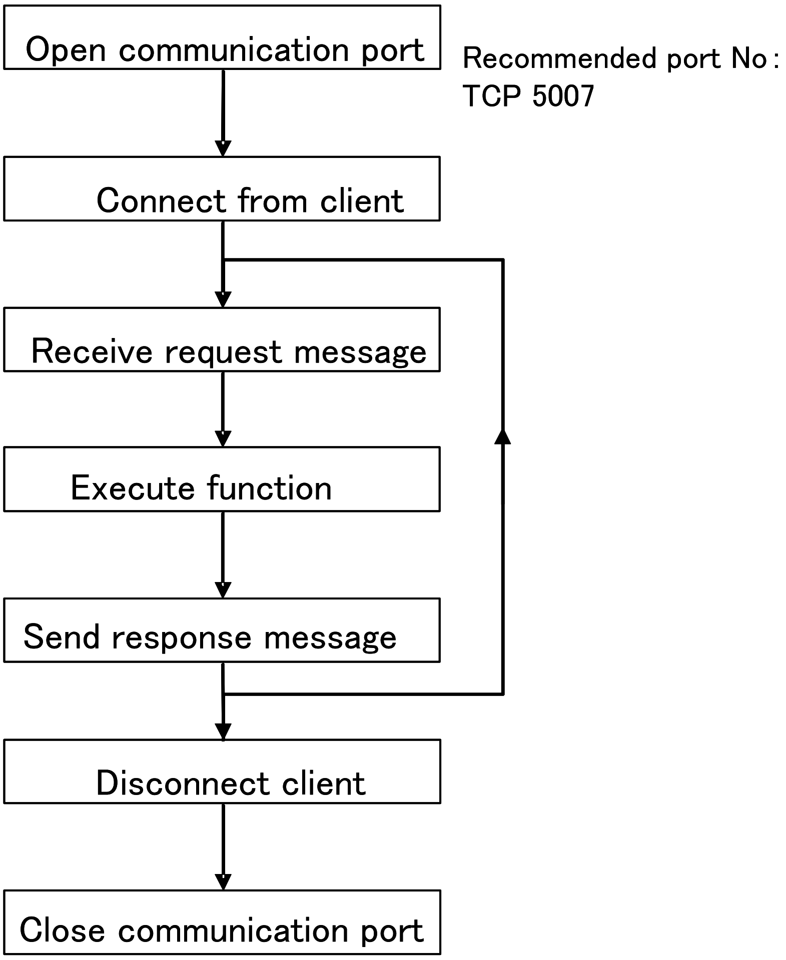


Figure 4-2　Communication procedure of server

Once opening a communication port, the server waits the connection from the client. Recommended TCP port number for opening the communication port is “5007”

After the connection is established, a function corresponding to the request message is executed. The execution result is stored in the response message, and then the message is sent to the client.

4.5. Communication procedure of client

The outline of the communication procedure of a client is shown as follows.

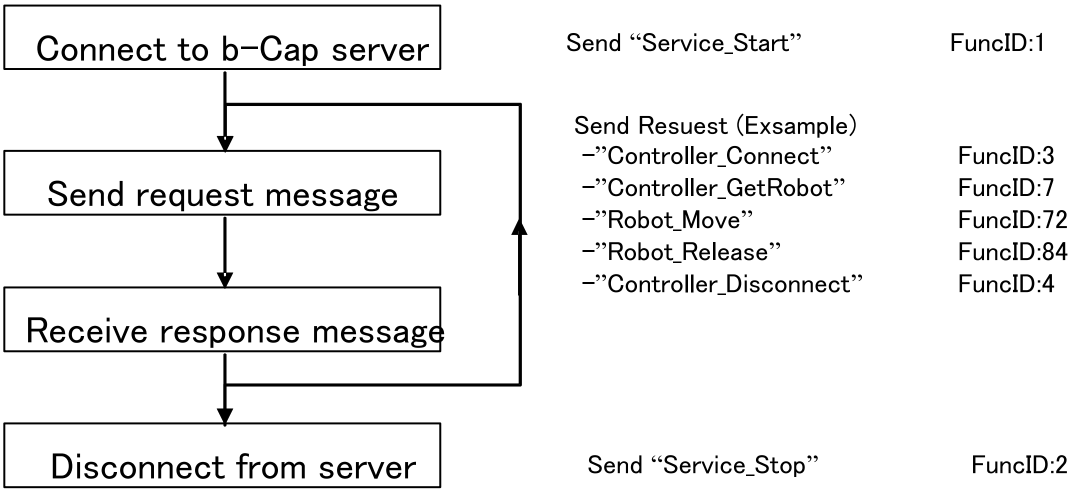

Connect to b-Cap server
Send request message
Receive response message
Disconnect from server
Send "Service_Start" FuncID:1
Send Resuest (Exsample)
-"Controller_Connect" FuncID:3
-"Controller_GetRobot" FuncID:7
-"Robot_Move" FuncID:72
-"Robot_Release" FuncID:84
-"Controller_Disconnect" FuncID:4
Send "Service_Stop" FuncID:2

Figure 4-3 Communication procedure of client

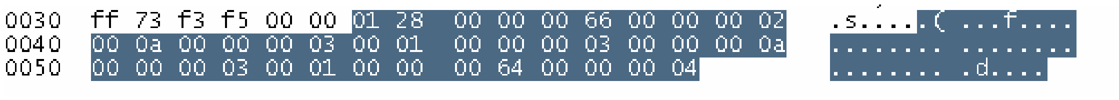
The client connects to the server to establish a session, sends the request message of the function executed, and then waits the execution result from the server.

The client needs to execute the time-out processing when there is no response from the server. Because the response time from the server differs depending on the contents being processed, you need to consider this difference when you set the time-out period.

5. Message sample

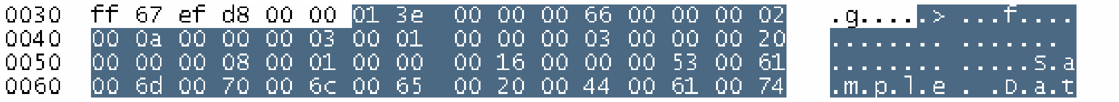
The sample of the communication message is shown as follows.

Variable_putValue（VT_I4　100）



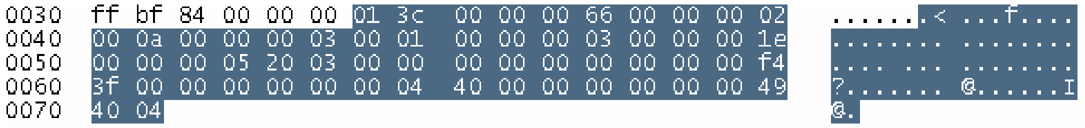
```
0030  ff 73 f3 f5 00 00 01 28  00 00 00 66 00 00 00 02   .s.....( ...f....
0040  00 0a 00 00 00 03 00 01  00 00 00 03 00 00 00 0a   ........ ........
0050  00 00 00 03 00 01 00 00  00 64 00 00 00 04         ........ .d....
```

Variable_putValue（VT_BSTR　“Sample Data”）



```
0030  ff 67 ef d8 00 00 01 3e  00 00 00 66 00 00 00 02   .g.....> ...f....
0040  00 0a 00 00 00 03 00 01  00 00 00 03 00 00 00 20   ........ .......
0050  00 00 00 08 00 01 00 00  00 16 00 00 00 53 00 61   ........ .....S.a
0060  00 6d 00 70 00 6c 00 65  00 20 00 44 00 61 00 74   .m.p.l.e . .D.a.t
```

Variable_putValue（VT_ARRAY|VT_R8　1.25,2.50,50）



```
0030  ff bf 84 00 00 00 01 3c  00 00 00 66 00 00 00 02   .......< ...f....
0040  00 0a 00 00 00 03 00 01  00 00 00 03 00 00 00 1e   ........ ........
0050  00 00 00 05 20 03 00 00  00 00 00 00 00 00 00 f4   .... ... ........
0060  3f 00 00 00 00 00 00 04  40 00 00 00 00 00 00 49   ?....... @......I
0070  40 04                                              @.
```

6. Network monitoring tool

6.1. Microsoft Network Monitor

This is a network monitoring tool that analyzes packet information across the network in real time.

The following shows the setup procedures. This setting allows you to analyze b-CAP packets.

(1) Install Microsoft Network Monitor 3.2.

Install it from the following address or download it from the Internet.

¥¥stream¥pub¥Microsoft¥NetworkMonitor32_x86.exe

Use if you would like to convert the tool into Japanese.

¥¥stream¥pub¥Microsoft¥NetworkMonitor32_x86_Japanese_patch.zip

(2) Start Microsoft Network Monitor 3.2 one time.

(This step will create a file being edited.)

(3) Edit "< my document > ¥Network Monitor 3¥Parsers¥my_sparser.npl".

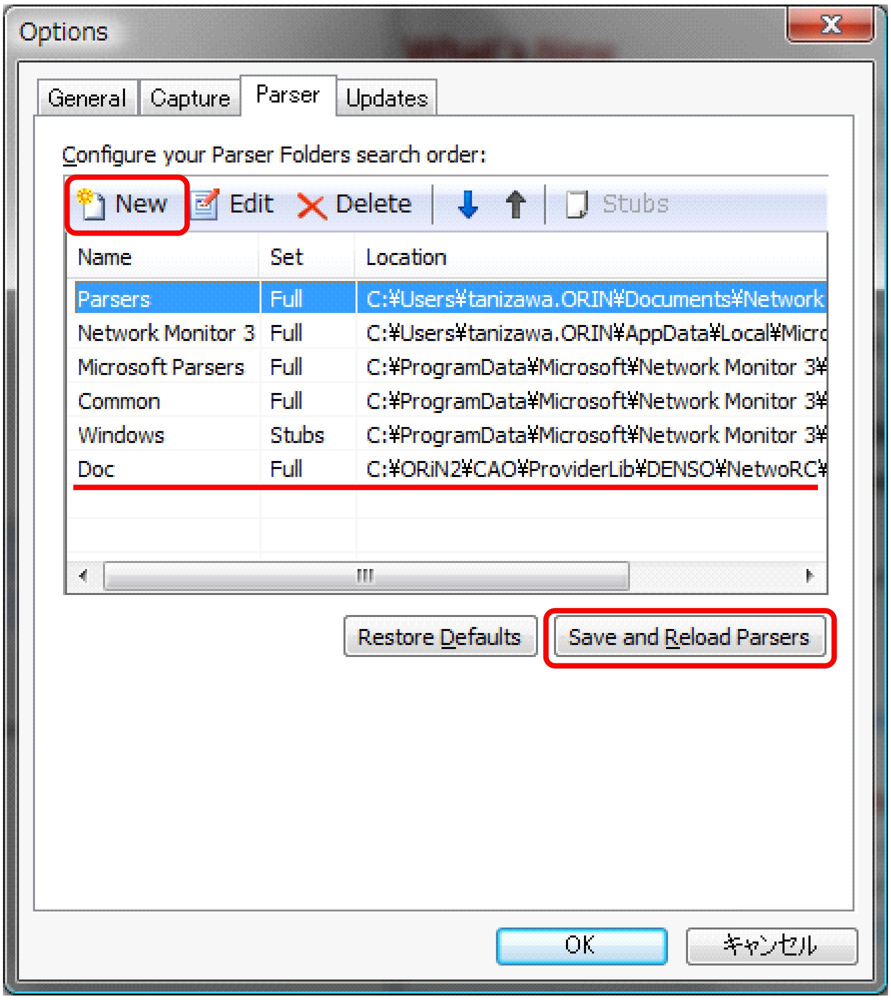
Add the red-inked part as shown below, and then save it.

```
// Personal NPL Files
include "RoboTalk.npl"
```

(4) Change options.

Because an error occurs when Microsoft Network Monitor is started, change the setting.

- Open "Tool" →"Option".
- From the "Parser" tab, click ”New”, and then add "b-CAP.npl".
- Click "Save and Reload Parsers".



Configure your Parser Folders search order:

Name	Set	Location
Parsers	Full	C:¥Users¥tanizawa.ORIN¥Documents¥Network
Network Monitor 3	Full	C:¥Users¥tanizawa.ORIN¥AppData¥Local¥Micro
Microsoft Parsers	Full	C:¥ProgramData¥Microsoft¥Network Monitor 3¥
Common	Full	C:¥ProgramData¥Microsoft¥Network Monitor 3¥
Windows	Stubs	C:¥ProgramData¥Microsoft¥Network Monitor 3¥
Doc	Full	C:¥ORiN2¥CAO¥ProviderLib¥DENSO¥NetwoRC¥

Figure 6-1 Option setting screen of Microsoft Network Monitor