b-CAP 通信仕様書

Version 1.2.2

July 11, 2014

【備考】

【改版履歴】

日付	版数	内容
2006-08-02	1.0.0	初版
2008-01-19	1.1.0	UDP 仕様の説明補足
2008-12-23	1.1.1	誤植修正
2009-09-02	1.1.2	Microsoft Network Monitor の設定手順追加
2010-08-20	1.1.3	COM 使用の説明追記
2010-10-22	1.1.4	PPP の説明追記
2012-03-23	1.1.5	誤植修正
2013-01-30	1.2.0	ZIP 圧縮パケットの説明追記
2014-01-31	1.2.1	誤植修正
2014-07-11	1.2.2	TCP タイムアウト処理について追記

目次

1. はじめに

本仕様書は, b-CAP の通信プロトコルを規定するものである.

b-CAP は, CAP の概念を踏襲しつつ, 通信速度の向上を狙ったプロトコルである. これにより, b-CAP は CAP ファミリと同様な以下の特徴を持っている.

- CAO プロバイダのオブジェクトモデルと同様なサービス構造
- オブジェクト ID で対象オブジェクトを指定しての関数呼び出し
- サーバからのイベントをポーリングで実現

b-CAP は TCP ストリーム通信として実装される. これは b-CAP のパケットにチェックコードが存在しないため, 下位層のプロトコルでエラーフリーなプロトコルが必要なためである.

(補足)

b-CAP/COM[1]の場合, パケットに CRC を付加する. 詳細については 3.1 を参照してください.

b-CAP/UDP および b-CAP/COM の場合のリトライ処理は b-CAP クライアント&サーバアプリ側で実装しなくてはならない.

[1] b-CAP/COM は RS232C 通信を想定して便宜的に CRC を付加しているが, 本来は RS232C であっても PPP (Point to Point Protocol)等のエラーフリーのプロトコルをインストールして b-CAP を使うことを推奨する. PPP を活用した方が, RS232C 直接接続に限らず, 赤外線やモデム接続等も容易になる.

2. b-CAP の構造

b-CAP は，CAO プロバイダのオブジェクトモデルと同様なサービス構造を持ち，1つの b-CAP メッセージが1つのサービス（関数）に対応する．この構造を以下に図で示す．

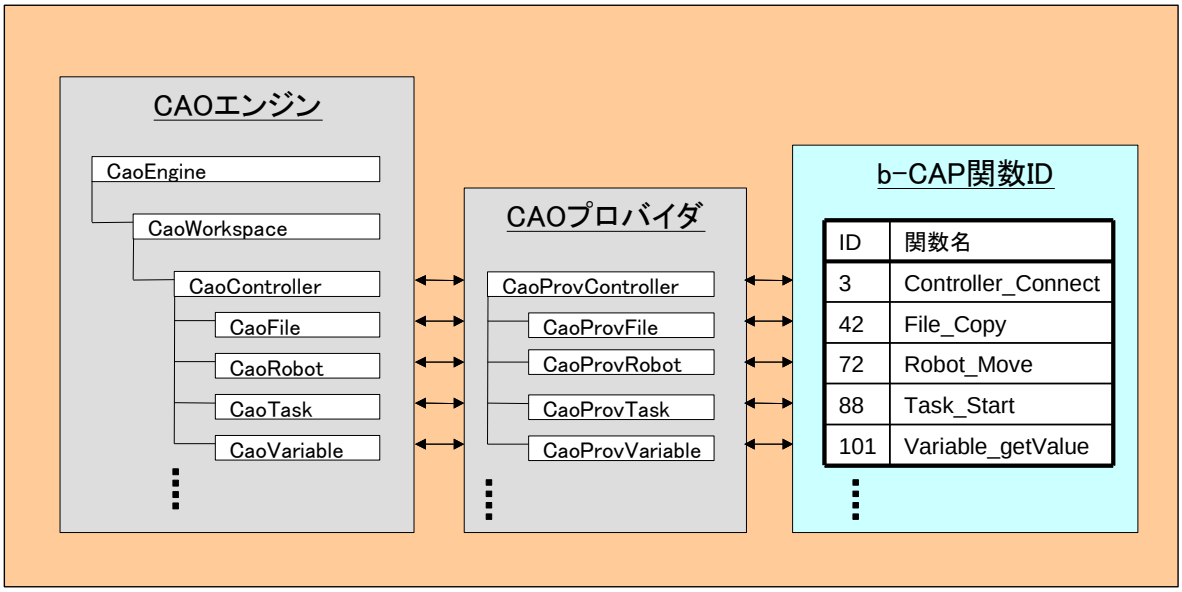


図 1-1 b-CAP の構造

b-CAP は，サービスを要求する b-CAP クライアントとサービスを実行し結果を返す b-CAP サーバの 2 つのプログラムで構成される．

b-CAP クライアントは要求するサービスに必要な情報を格納した要求メッセージを作成し，サーバ側に送信し，応答メッセージを受信して実行結果を確認する．

b-CAP サーバでは，クライアントからの要求メッセージを受信し，関数 ID に対応するサービスを実行する．サービス実行後は，結果及び値を応答メッセージに格納し，クライアント側に送信する．

クライアント，サーバの処理手順については，4 で詳細を説明する．

以下に b-CAP による接続例を示す．

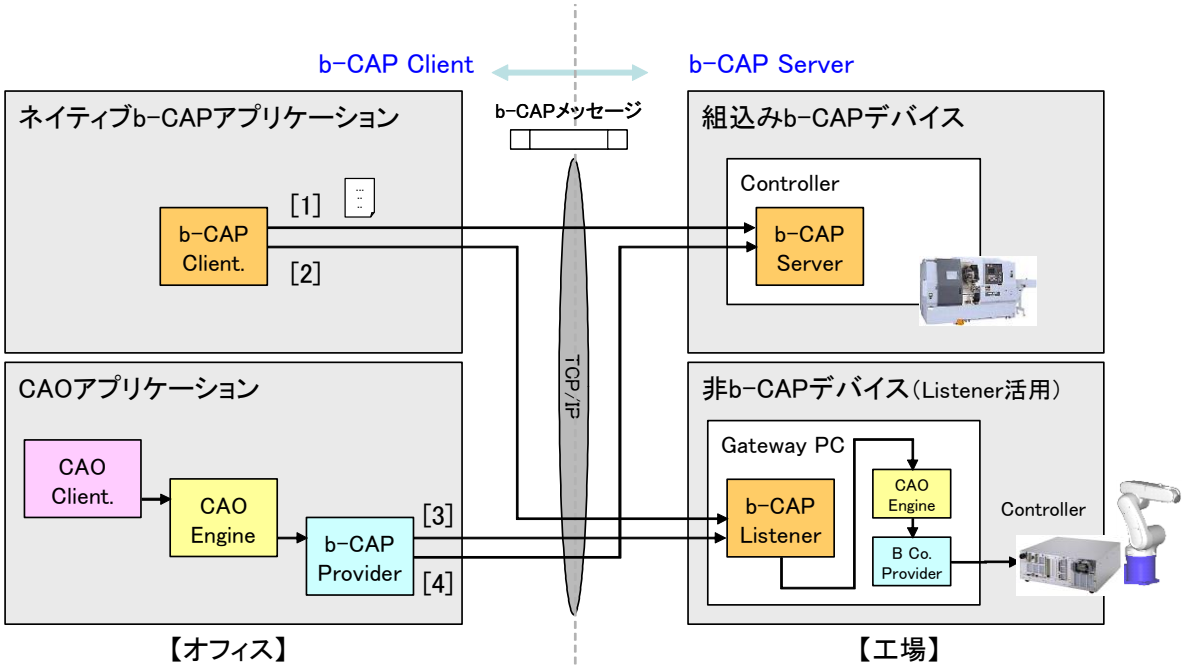

b-CAP Client
b-CAP Server
b-CAPメッセージ
ネイティブb-CAPアプリケーション
[1]
b-CAP Client.
[2]
組込みb-CAPデバイス
Controller
b-CAP Server
TCP/IP
CAOアプリケーション
CAO Client.
CAO Engine
b-CAP Provider
[3]
[4]
非b-CAPデバイス(Listener活用)
Gateway PC
b-CAP Listener
CAO Engine
B Co. Provider
Controller
【オフィス】
【工場】

図 1-2 b-CAP 接続例

3. パケット構造

3.1. パケット構造

b-CAP のメッセージは次のような構造になっている.

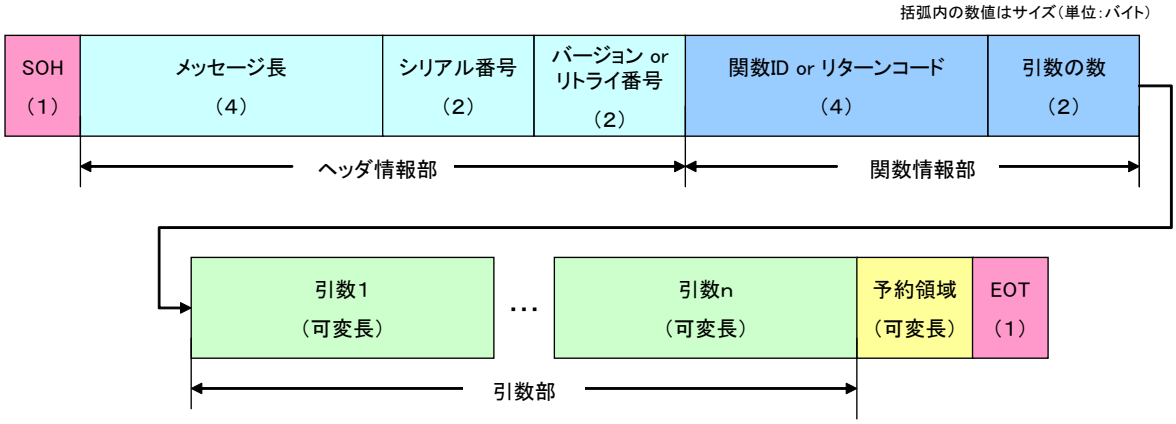


図 1-3 パケット構造

以下にパケットの各要素の説明を示す. ここで各データのメモリイメージはインテル方式(リトルエンディアン)で格納される.

ヘッダ	パケットの先頭につけるスタートコード. SOH(0x01)を使用する.
メッセージ長	パケット全体のデータ長. 符号なし長整数型(DWORD)を格納する. ヘッダからターミネータまでの長さを格納する(UDP の場合は最大 504 バイト).
シリアル番号	メッセージのシリアル番号. 符号なし短整数型(WORD)を格納する. 1 から WORD_MAX の任意の数字. この値は要求メッセージと応答メッセージで一致させる.
バージョン	プロトコルバージョンを格納する. この値は常に1を設定する. この情報は b-CAP/TCP でのみ使用する.
リトライ	リトライ処理開始前のシリアル番号を埋め込む. この情報は b-CAP/UDP, b-CAP/COM でのみ使用する.
関数 ID	呼び出し関数を識別する ID. 符号なし長整数型(DWORD)を格納する. 要求メッセージのときのみ使用する. (3.3 参照)
リターンコード	呼び出し関数の実行結果コード. 符号なし長整数型(DWORD)を格納する. 応答メッセージのときのみ使用する. (3.4 参照)
引数の数	呼び出し関数の引数の数, 又は呼び出し関数の出力変数の数. 符号なし短

	整数型(WORD)を格納する.
引数 n	n 番目の引数部. 引数を VARIANT 型のイメージで格納する. (3.2 参照)
予約領域	システムで予約されている領域です. この領域は可変長です.
ターミネータ	パケットの最後につけるエンドコード. EOT(0x04)を使用する.

b-CAP は通信デバイスによって"バージョン or リトライ"と"予約領域"の内容が異なる.
以下には各デバイスにおけるパケット構造を示す.

3.1.1. b-CAP/TCP のパケット構造

b-CAP/TCP では, 基本的なパケット構造の一部を以下のように使用する.

バージョン or リトライ番号 → バージョン
予約領域 → モード

また b-CAP/TCP では関数情報部から引数部までの情報を ZIP 圧縮して格納することができる.
パケット構造が圧縮・非圧縮のどちらであるかは"モード"の値から判別することができる.
以下に b-CAP/TCP の圧縮・非圧縮時パケット構造を示す.

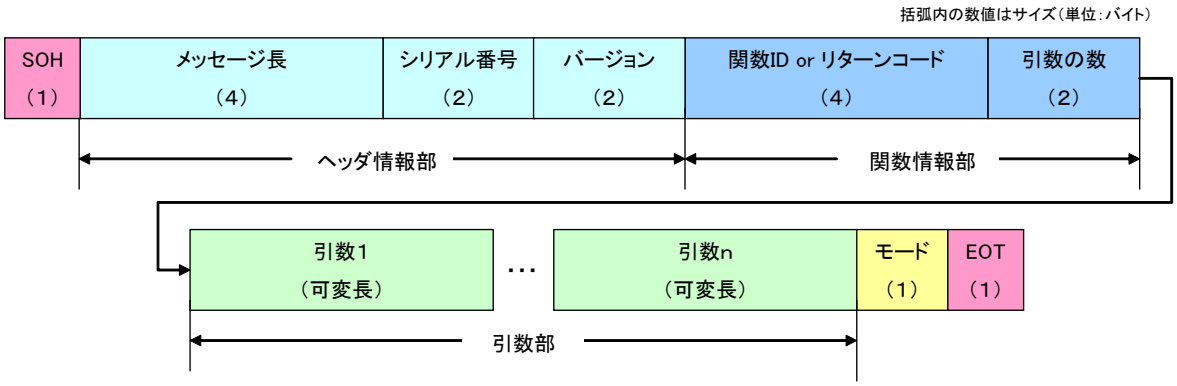


図 1-4 b-CAP/TCP パケット構造

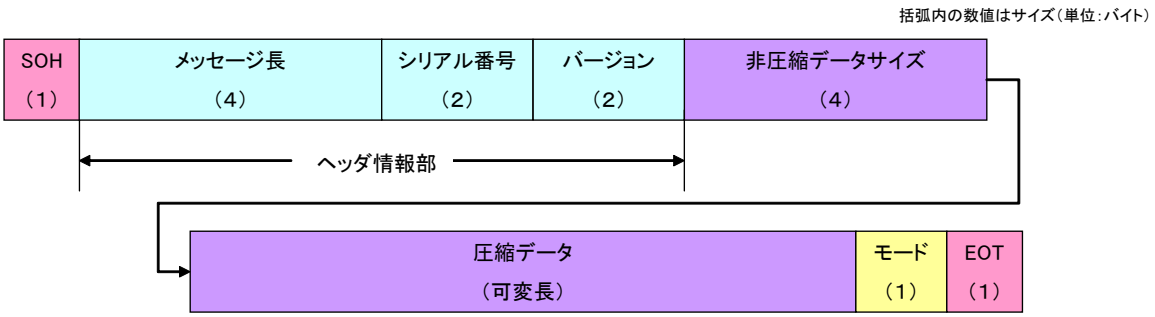


図 1-5 b-CAP/TCP 圧縮時のパケット構造

以下に基本のパケット要素と異なる部分の説明を示す.

バージョン	プロトコルバージョンとして 1 を格納します.
非圧縮データサイズ	圧縮データ情報を解凍したときのサイズ
圧縮データ	関数情報部から引数部までを圧縮したバイナリデータ
モード	パケットの状態
	0 通常パケット
	1 ZIP 圧縮パケット

3.1.2. b-CAP/UDP のパケット構造

b-CAP/UDP では, 基本的なパケット構造の一部を以下のように使用する.

バージョン or リトライ番号	→	リトライ番号
予約領域	→	なし

b-CAP/UDP では, リトライパケットを使用できる.

“リトライ番号”に初回パケットの“シリアル番号”の値を格納する. このため, 初回は“リトライ番号”と“シリアル番号”の値は一致する.

リトライ可能な回数については, クライアント&サーバアプリで決定する.

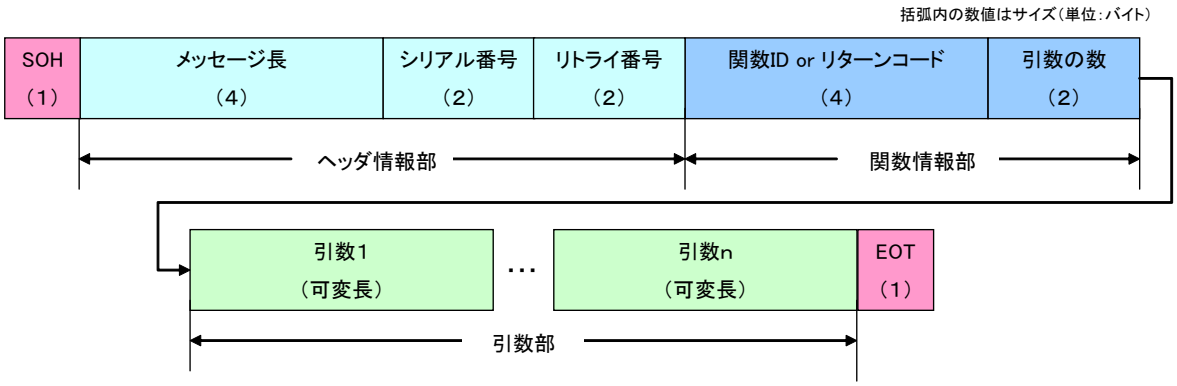


図 1-6 b-CAP/UDP パケット構造

以下に基本のパケット要素と異なる部分の説明を示す.

リトライ番号	リトライ前のシリアル番号を埋め込みます.
予約領域	b-CAP/UDP ではこの領域は使用しません.
	このバイトを付加しないでください.

3.1.3. b-CAP/COM のパケット構造

b-CAP/COM では，基本的なパケット構造の一部を以下のように使用する．

バージョン or リトライ番号 → リトライ番号

予約領域 → CRC

b-CAP/COM では，リトライパケットを使用できる．

“リトライ番号”に初回パケットの“シリアル番号”の値を格納する．このため，初回は“リトライ番号”と“シリアル番号”の値は一致する．

リトライ可能な回数については，クライアント&サーバアプリで決定する．

また b-CAP/COM では，CRC を付加する必要がある．

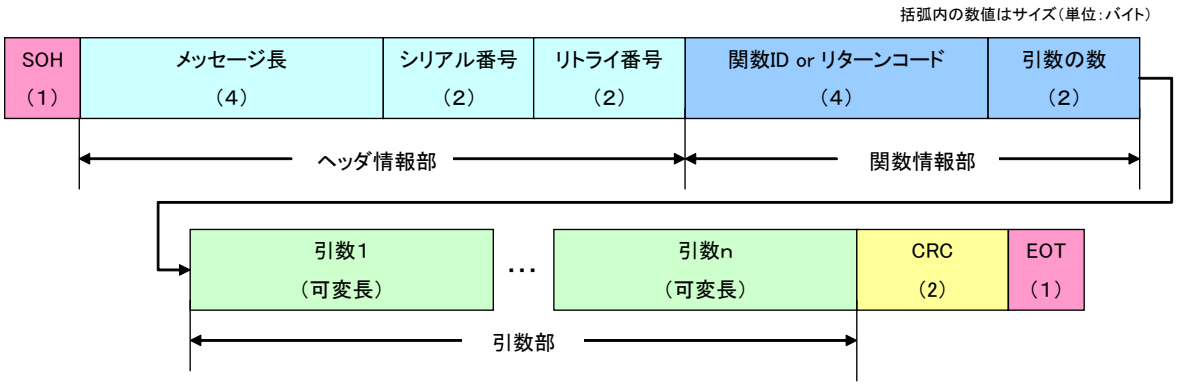


図 1-7 b-CAP/COM パケット構造

以下に基本のパケット要素と異なる部分の説明を示す．

リトライ番号　　リトライ前のシリアル番号を埋め込みます．

CRC　　CRC にはヘッダ情報部から引数部までの CRC を格納する．

CRC の計算条件を以下に示す．

CRC タイプ	:	CRC-CCITT
初期値	:	0xFFFF
出力 XOR	:	0x0000
ビット送り方向	:	左
入力ビット反転	:	なし
出力ビット反転	:	なし

3.2. 引数部構造

引数部はデータ型によって可変長の構造をとり，複数のデータ型を表記できるように作成されている．
以下に引数部の構造を示す．

括弧内の数値はサイズ（単位：バイト）

図 1-8　引数部の構造

引数部は共通部分としてデータ型と要素数で構成される．データ部は共通部の情報を元に構造が決定される．
以下に引数部の各要素の説明を示す．

引数データ長	データ型，要素数，データの合計バイト数．符号なし長整数型（DWORD）を格納する．データ型ごとのサイズは表 1 のようになる．
データ型	引数のデータ型．符号なし短整数型（WORD）を格納する．使用できるデータ型は表 1 のようになる．
要素数	引数の要素数．符号なし長整数型（DWORD）を格納する．データ型に VT_ARRAY が使用されていないときは，常に 1 にする．
データ部	引数のデータ．データ型によって使用するサイズが異なる．各データのサイズは表 1 を参照すること．また，データ部に格納される情報の構造については，図 1-9 を参照すること．

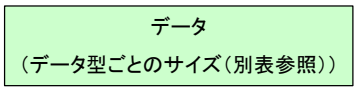
表 1 使用可能なデータ型

データ型	値	サイズ(Byte)	説明
VT_EMPTY	0	0	空データ
VT_NULL	1	0	NULL 値
VT_ERROR	10	4	エラーコード
VT_UI1	17	1	バイナリ
VT_I2	2	2	短整数
VT_UI2	18	2	符号なし短整数
VT_I4	3	4	長整数
VT_UI4	19	4	符号なし長整数
VT_R4	4	4	単精度浮動小数点

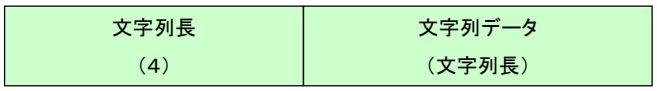
VT_R8	5	8	倍精度浮動小数点
VT_CY	6	8	通貨型
VT_DATE	7	8	日付型
VT_BOOL	11	2	ブール型
VT_BSTR	8	文字数×2＋4	文字列型 先頭 4 バイトに文字列長が入る. 文字列長の後ろに文字列を Unicode で格納する.
VT_VARIANT	12	-	Variant 型 データに引数部と同じものが入る. VT_ARRAY のときのみ使用することができる.
VT_ARRAY	0x2000	-	配列 他のデータ型との論理和で指定する. 指定した型のデータを連続で格納する. 1次元配列のみを格納することができます.

括弧内の数値はサイズ（単位：バイト）

・VT_BSTR、VT_VARIANT、VT_ARRAY以外のデータ

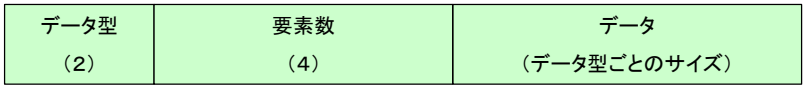


・VT_BSTR



・文字列はUnicode文字列で格納
・文字列の最後にNULL文字は含めない

・VT_VARIANT



・引数部と同じ構造
・VT_ARRAYのときのみ使用

・VT_ARRAY

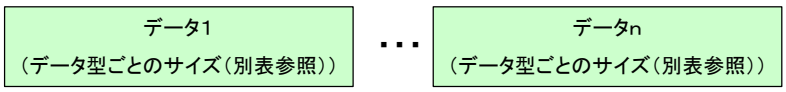


図 1-9 データの構造

3.3. 関数 ID

b-CAP では関数 ID は以下のように割り振られている.

表 2 関数 ID の割り当て

関数 ID	説明
1～137	既定の関数
138～255	予約領域
255～	ユーザ関数

既定の関数以外の関数を使用したいときは, ユーザ関数に任意の関数を割り当てて使用することができる.

b-CAP の既定の関数の一覧を以下に示す.

表 3 既定の関数一覧

関数 ID	関数名	説明
1	Service_Start	サーバサービスの開始
2	Service_Stop	サーバサービスの停止
3	Controller_Connect	コントローラとの接続
4	Controller_Disconnect	コントローラとの切断
5	Controller_GetExtension	コントローラの拡張ボード取得
6	Controller_GetFile	コントローラのファイル取得
7	Controller_GetRobot	コントローラのロボット取得
8	Controller_GetTask	コントローラのタスク取得
9	Controller_GetVariable	コントローラの変数取得
10	Controller_GetCommand	コントローラのコマンド取得
11	Controller_GetExtensionNames	コントローラの拡張ボード名一覧取得
12	Controller_GetFileNames	コントローラのファイル名一覧取得
13	Controller_GetRobotNames	コントローラのロボット名一覧取得
14	Controller_GetTaskNames	コントローラのタスク名一覧取得
15	Controller_GetVariableNames	コントローラの変数名一覧取得
16	Controller_GetCommandNames	コントローラのコマンド名一覧取得
17	Controller_Execute	コントローラの拡張関数の実行
18	Controller_GetMessage	コントローラのイベントメッセージ取得
19	Controller_GetAttribute	コントローラの属性値取得
20	Controller_GetHelp	コントローラのヘルプ文字列取得

21	Controller_GetName	コントローラの名前取得
22	Controller_GetTag	コントローラのタグ情報取得
23	Controller_PutTag	コントローラのタグ情報設定
24	Controller_GetID	コントローラの ID 取得
25	Controller_PutID	コントローラの ID 設定
26	Extension_GetVariable	拡張ボードの変数取得
27	Extension_GetVariableNames	拡張ボードの変数名一覧取得
28	Extension_Execute	拡張ボードの拡張関数実行
29	Extension_GetAttribute	拡張ボードの属性値取得
30	Extension_GetHelp	拡張ボードのヘルプ文字列取得
31	Extension_GetName	拡張ボードの名前取得
32	Extension_GetTag	拡張ボードのタグ情報取得
33	Extension_PutTag	拡張ボードのタグ情報設定
34	Extension_GetID	拡張ボードの ID 取得
35	Extension_PutID	拡張ボードの ID 設定
36	Extension_Release	拡張ボードの解放
37	File_GetFile	ファイルの別ファイル取得
38	File_GetVariable	ファイルの変数取得
39	File_GetFileNames	ファイルの別ファイル名一覧取得
40	File_GetVariableNames	ファイルの変数名一覧取得
41	File_Execute	ファイルの拡張関数実行
42	File_Copy	ファイルのコピー
43	File_Delete	ファイルの削除
44	File_Move	ファイルの移動
45	File_Run	ファイルの実行
46	File_GetDateCreated	ファイルの作成日時取得
47	File_GetDateLastAccessed	ファイルの最終アクセス日時取得
48	File_GetDateLastModified	ファイルの最終更新日時取得
49	File_GetPath	ファイルのパス取得
50	File_GetSize	ファイルのサイズ取得
51	File_GetType	ファイルのファイルタイプ取得
52	File_GetValue	ファイルの内容取得
53	File_PutValue	ファイルの内容設定
54	File_GetAttribute	ファイルの属性取得
55	File_GetHelp	ファイルのヘルプ文字列取得

56	File_GetName	ファイルの名前取得
57	File_GetTag	ファイルのタグ情報取得
58	File_PutTag	ファイルのタグ情報設定
59	File_GetID	ファイルの ID 取得
60	File_PutID	ファイルの ID 設定
61	File_Release	ファイルの解放
62	Robot_GetVariable	ロボットの変数取得
63	Robot_GetVariableNames	ロボットの変数名一覧取得
64	Robot_Execute	ロボットの拡張関数実行
65	Robot_Accelerate	ロボットの ACCEL 文実行
66	Robot_Change	ロボットの CHANGE 文実行
67	Robot_Chuck	ロボットの GRASP 文実行
68	Robot_Drive	ロボットの DRIVE 文実行
69	Robot_GoHome	ロボットの GOHOME 文実行
70	Robot_Halt	ロボットの HALT 文実行
71	Robot_Hold	ロボットの HOLD 文実行
72	Robot_Move	ロボットの MOVE 文実行
73	Robot_Rotate	ロボットの ROTATE 文実行
74	Robot_Speed	ロボットの SPEED/JSPEED 文実行
75	Robot_Unchuck	ロボットの REELASE 文実行
76	Robot_Unhold	ロボットの HOLD 文解除
77	Robot_GetAttribute	ロボットの属性値取得
78	Robot_GetHelp	ロボットのヘルプ文字列取得
79	Robot_GetName	ロボットの名前取得
80	Robot_GetTag	ロボットのタグ情報取得
81	Robot_PutTag	ロボットのタグ情報設定
82	Robot_GetID	ロボットの ID 取得
83	Robot_PutID	ロボットの ID 設定
84	Robot_Release	ロボットの解放
85	Task_GetVariable	タスクの変数取得
86	Task_GetVariableNames	タスクの変数名一覧取得
87	Task_Execute	タスクの拡張関数実行
88	Task_Start	タスクの開始
89	Task_Stop	タスクの停止
90	Task_Delete	タスクの削除

91	Task_GetFileName	タスクの元ファイル名
92	Task_GetAttribute	タスクの属性取得
93	Task_GetHelp	タスクのヘルプ文字列取得
94	Task_GetName	タスクの名前取得
95	Task_GetTag	タスクのタグ情報取得
96	Task_PutTag	タスクのタグ情報設定
97	Task_GetID	タスクの ID 取得
98	Task_PutID	タスクの ID 設定
99	Task_Release	タスクの解放
100	Variable_GetDateTime	変数のタイムスタンプ取得
101	Variable_GetValue	変数の値取得
102	Variable_PutValue	変数の値設定
103	Variable_GetAttribute	変数の属性値取得
104	Variable_GetHelp	変数のヘルプ文字列取得
105	Variable_GetName	変数の名前取得
106	Variable_GetTag	変数のタグ情報取得
107	Variable_PutTag	変数のタグ情報設定
108	Variable_GetID	変数の ID 取得
109	Variable_PutID	変数の ID 設定
110	Variable_GetMicrosecond	変数のタイムスタンプ(ミリ秒)取得
111	Variable_Release	変数の解放
112	Command_Execute	コマンドの実行
113	Command_Cancel	コマンドのキャンセル
114	Command_GetTimeout	コマンドのタイムアウト時間取得
115	Command_PutTimeout	コマンドのタイムアウト時間設定
116	Command_GetState	コマンドの状態取得
117	Command_GetParameters	コマンドのパラメータ取得
118	Command_PutParameters	コマンドのパラメータ設定
119	Command_GetResult	コマンドの実行結果取得
120	Command_GetAttribute	コマンドの属性値取得
121	Command_GetHelp	コマンドのヘルプ文字列取得
122	Command_GetName	コマンドの名前取得
123	Command_GetTag	コマンドのタグ情報取得
124	Command_PutTag	コマンドのタグ情報設定
125	Command_GetID	コマンドの ID 取得

126	Command_PutID	コマンドの ID 設定
127	Command_Release	コマンドの解放
128	Message_Reply	イベントメッセージの応答
129	Message_Clear	イベントメッセージのクリア
130	Message_GetDateTime	イベントメッセージのタイムスタンプ取得
131	Message_GetDescription	イベントメッセージの説明文取得
132	Message_GetDestination	イベントメッセージの送信先取得
133	Message_GetNumber	イベントメッセージのメッセージ番号取得
134	Message_GetSerialNumber	イベントメッセージのシリアル番号取得
135	Message_GetSource	イベントメッセージの送信元取得
136	Message_GetValue	イベントメッセージの値取得
137	Message_Release	イベントメッセージの解放

3.4. リターンコード

b-CAP ではリターンコードは以下のように割り振られている.

表 4 リターンコードの割り当て

リターンコード	説明
0x00000000～0x8000FFFF	既定リターンコード, 予約領域
0x80010000～0x8001FFFF	ユーザ定義エラー

以下に示す“既定エラーコード”以外のエラーコードを作成するときは, “ユーザ定義エラー”の値の範囲内で任意のエラーコードを割り当てることができる.

表 5 既定のリターンコード一覧

リターンコード	エラー	説明
0x00000000	S_OK	正常終了.
0x80004001	E_NOTIMPL	未実装.
0x80004004	E_ABORT	関数が中断されました.
0x80004005	E_FAIL	関数が失敗しました.
0x80070005	E_ACCESSDENIED	アクセスできません.
0x80070006	E_HANDLE	ハンドルが不正です.
0x8007000E	E_OUTOFMEMORY	メモリが不足しています.
0x80070057	E_INVALIDARG	引数が不正です.
0x8000FFFF	E_UNEXPECTED	致命的エラーが発生しました.

4. 通信手順

4.1. 通信シーケンス

b-CAP は TCP と UDP の2つの通信仕様があり，どちらの通信シーケンスでも，必ずクライアントからの要求パケットの送信によって始まる．サーバ側は，要求パケットに対応するコマンド（関数）を実行し， 実行結果を応答パケットに載せてクライアントへ返す．

1つのセッションでは，要求メッセージ送信後は応答メッセージの受信を待ち，常に同期をとるべきである．もし，複数の要求メッセージを使用する場合には，複数セッションで行うことが望ましい．

サーバ側では要求パケットを受信してから回答を返信するまでの時間に関しては何の規定もない．このため，サーバ側の処理時間が長いときにクライアントのタイムアウト検出時間が短い場合，クライアント側でタイムアウトが発生することになるので注意が必要である．

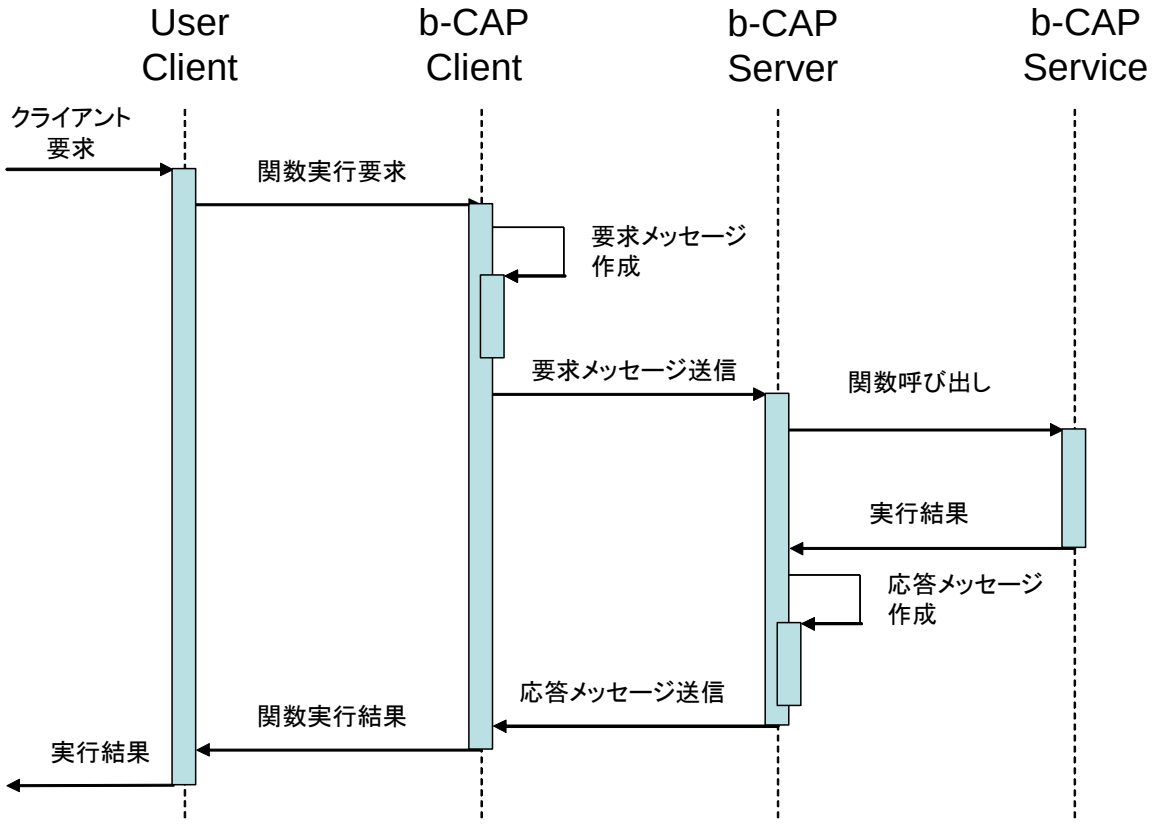


図 1-10　通信シーケンス

4.2. b-CAP/TCP リトライシーケンス

b-CAP/TCP の場合，リトライ処理は TCP プロトコルスタックで処理されるためクライアントプログラムが明示的にリトライ処理を書く必要はない．しかしながら，OS によってはそのデフォルトのリトライ時間が長く（数百ミリ秒～数秒）設定されているので，アプリケーションによっては不都合な場合がある．その場合には，TCP であ

ってもタイムアウトを明示的に短く設定して，わざとタイムアウトを発生させることでリトライ処理（またはエラー処理）をするようなプログラミングが必要となる．

そもそも，そのようなアプリケーションの場合は次節の UDP を使うのも一つの方法である．

4.3. b-CAP/UDP リトライシーケンス

b-CAP/UDP の場合はリトライ処理をクライアント/サーバアプリ側で実装しなくてはならない．その概要を下記に示す．

【クライアント側】

(1) b-CAP/UDP は再送のとき，シリアル番号をカウントアップして送る．
→ 応答がないと思って再送したけど，その直後に前回のリクエストの結果が返ってきたときにそれを捨てるため（シリアル番号不一致）．

(2) 同時に b-CAP パケットの予約領域に最初のシリアル番号を格納する．

【サーバ側】

(1) 予約領域にシリアル番号が入っているとき，それがリトライパケットであると認識する．複数接続を許可する場合は，相手（IP＋ポート）毎に最後のシリアル番号と回答を保存しておく必要がある．

(2) リトライパケットの場合，そのシリアル番号が直前に実行したシリアル番号と一致するなら，コマンドの二重実行を防ぐために再実行しないで直前に実行したときの回答のコピーを返します．このとき，回答のシリアル番号はリトライパケットのシリアル番号に更新します．

4.4. サーバの通信手順

以下にサーバの通信手順の概要を示します.

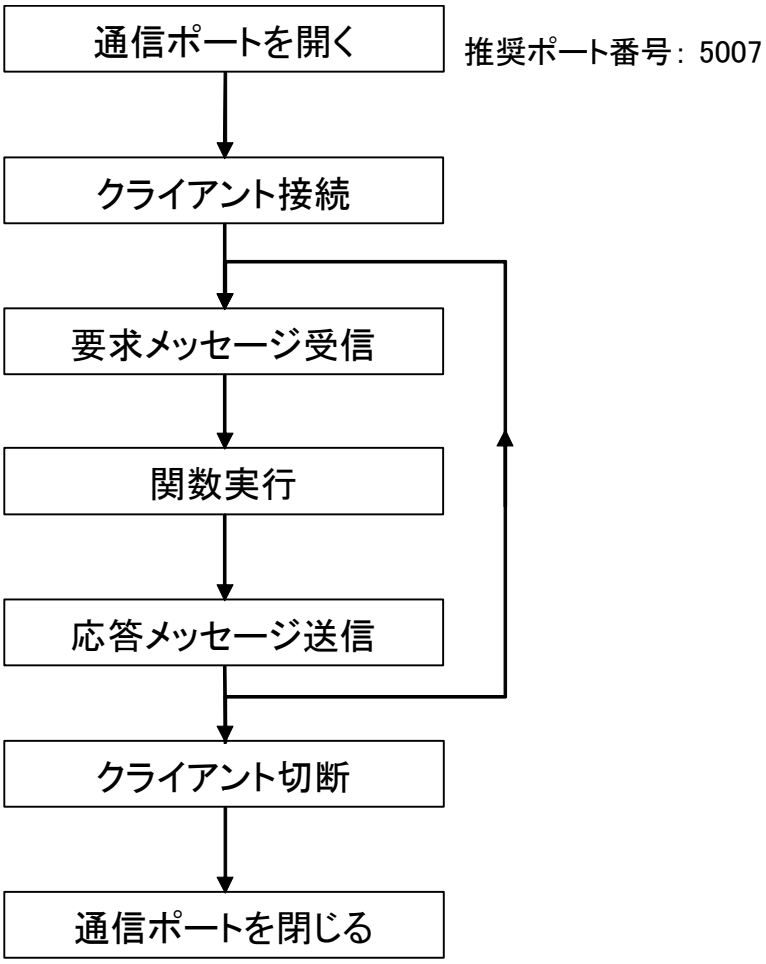


図 1-11 サーバの通信手順

サーバは, 最初に通信ポートを開いた後, クライアントからの接続を待つ. このとき開く TCP のポート番号は“5007”を推奨する.

クライアントからの接続後は, 要求メッセージに対応する関数を実行する. 実行結果は応答メッセージに格納し, クライアントにそのメッセージを返す.

4.5. クライアントの通信手順

以下にクライアントの通信手順の概要を示します.

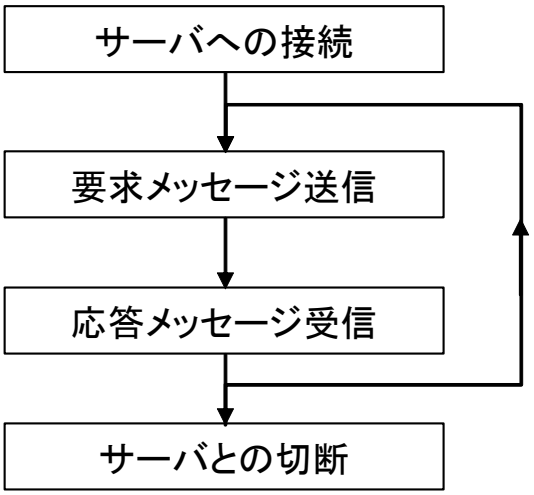


図 1-12 クライアントの通信手順

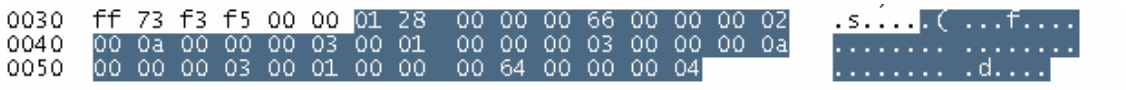
クライアントは最初にサーバへの接続し, セッションを確立する. この後, 実行した関数の要求メッセージを送信し, サーバからの実行結果を待ちます.

クライアントは, サーバからの応答がないときにタイムアウト処理を行う必要がある. しかし, サーバの応答時間は処理内容によって異なるため, タイムアウトの設定時間は注意が必要である.

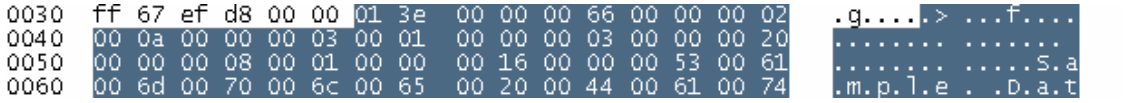
5. メッセージサンプル

以下に通信メッセージのサンプルを示す.

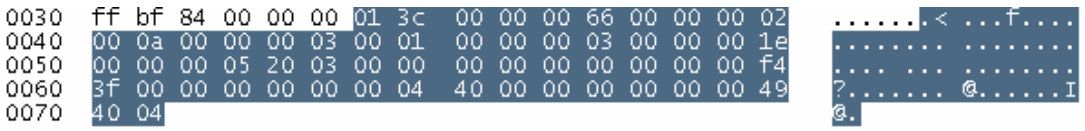
Variable_putValue（VT_I4　100）



Variable_putValue（VT_BSTR　“Sample Data”）



Variable_putValue（VT_ARRAY|VT_R8　1.25,2.50,50）



6. 便利なツール

6.1. Microsoft Network Monitor

このツールは，ネットワーク上に流れるパケット情報をリアルタイムに解析するネットワーク監視ツールです.

以下に設定手順を示します. この設定を行うことで b-CAP パケットの解析が可能になります.

(1) Microsoft Network Monitor 3.2 をインストール

下記からインストールするか、インターネットからダウンロードする

¥¥stream¥pub¥Microsoft¥NetworkMonitor32_x86.exe

日本語化したい人は以下を適用する

¥¥stream¥pub¥Microsoft¥NetworkMonitor32_x86_Japanese_patch.zip

(2) Microsoft Network Monitor 3.2 を 1 度起動する

（次に編集するファイルが作成されるので必要）

(3) “<マイドキュメント>¥Network Monitor 3¥Parsers¥my_sparser.npl”を編集する

赤字の部分を追加して保存

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

(4) オプションを変更する

Microsoft Network Monitor を起動するとエラーが出るので設定を変更する

- 「Tool」→「Option」を開く
- 「Paser」タブ「New」ボタンから”b-CAP.npl”のフォルダを追加する
- 「Save and Reload Pasers」を押す

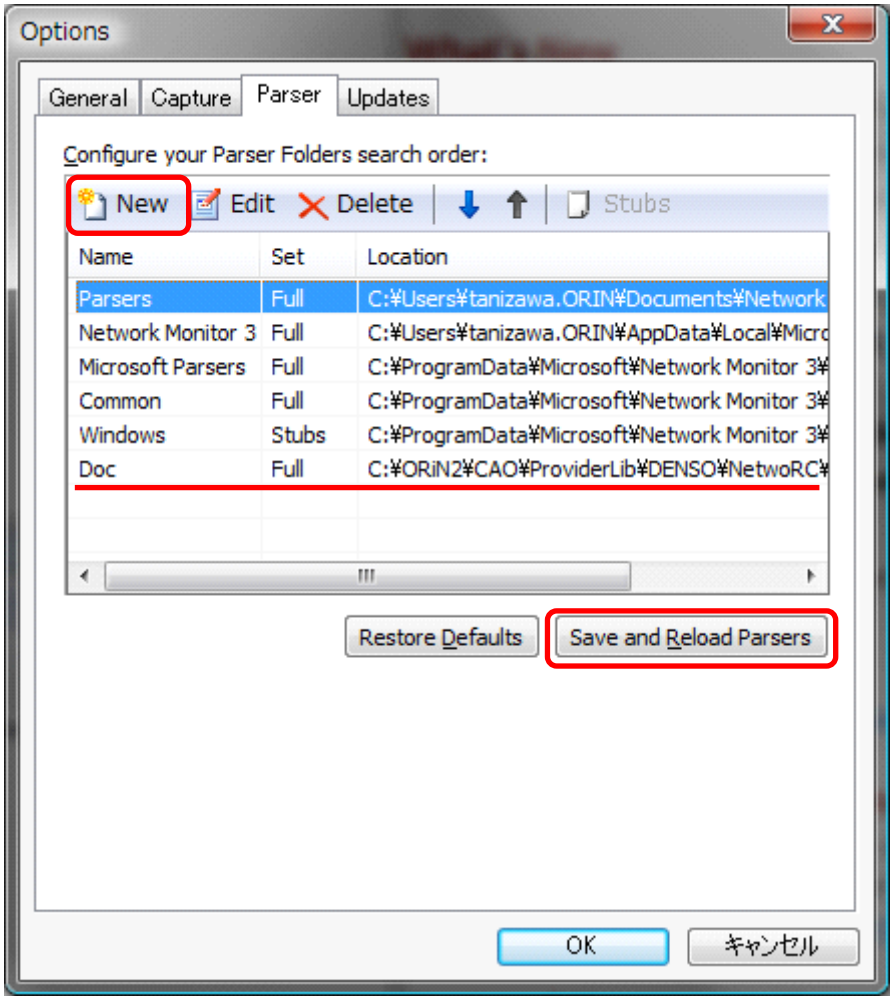


図 6-1 Microsoft Network Monitor の Option 設定画面