DENSO
b-CAP 通信仕様書
RC7 用

Version 1.1.12

December 07, 2011

【備考】

【改版履歴】

日付	版数	内容
2006-08-02	1.0.0	初版
2007-10-28	1.0.1	第5章「b-CAP 通信関数」追加 for DENSO RC7M/J
2007-12-04	1.0.2	(1) 第3章 修正 “SOH = 0x01” (2) 第3章 リターンコード “0x80010004 E_ROBOTISBUSY 追加 (3) 第5章 コントローラオブジェクトに “Contoller_Execute” メソッド追加 (4) 第5章 ロボットオブジェクトに “Robot_Change” 追加 Tool=n Work=n (5) 第5章にシステム変数追加 @EXTSPEED @EXTACCEL @EXTDECEL (6) 第6章に予約変数 I[10] についての注記追加
2008-01-14	1.0.3	(1) 第5章 Task_start iMode の説明を修正 (2) 第5章 Task_Stop iMode の説明を修正 (3) 第5章 Controller_Execute パケットサンプル修正 (4) 第5章 パケットサンプルとその説明を追加
2008-01-31	1.0.4	(1) 第5章 関数 Controller_GetVariable のコントローラ変数の説明追加 “TOOL” – Tool 設定 “WORK” – Work 設定 “_ITP” – ITP CNF “_PAC” – PAC CNF “_DIO” – DIO CNF “_ARM” – ARM CNF “_SRV” – SRV CNF “_SPD” – SPD CNF “_VIS” – VIS CNF “_COM” – COM CNF
2008-02-25	1.0.5	(1) 第5章 関数 Controller_GetVariable のコントローラ変数の説明追加 “@AUTO_ENABLE” – 自動イネーブル信号 “@PROTECTIVE_STOP” – 保護停止

		“@DEADMAN_SW” – デッドマンスイッチ (2) 第5章 変数型修正 VT_BOOL to VT_I2 から “@EMERGENCY_STOP”
2008-05-05	1.0.6	ROM 版 : 2.622M 05/13/2008 b-CAP [VERTICAL] 2.623M 05/13/2008 b-CAP [SCARA] (1) コントローラオブジェクトのメソッド・プロパティ追加 - Controller_Execute(“SuspendAll”) - Controller_Execute(“ResetAll”) - Controller_GetTaskNames() - Robot_Execute(“MotionComp”) (2) コントローラオブジェクトの動作コマンド形式追加 - Move J(J1,J2,J3,J4,J5,J6) - Move T(X,Y,Z,Ox,Oy,Oz,Ax,Ay,Az,Fig) 注記:コントローラバージョン 2.801M以降で正式対応済み
2008-06-12	1.0.7	ROM 版 : 2.623M 06/11/2008 b-CAP [VERTICAL] 2.624M 06/11/2008 b-CAP [SCARA] (1) ファームウェアをアップデータしたことにより, ティーチチェック時のペンダントの問題を解決しました. 注記:コントローラバージョン 2.801M以降で正式対応済み
2009-03-02	1.1.0	コントローラバージョン 2.8
2009-04-01	1.1.1	P2J,J2P,J2T,P2T,T2J,T2P,TINV,NORMTRN のオプションパラメータのデータを VARIANT に変更しました.
2009-05-29	1.1.2	体裁修正
2009-06-03	1.1.3	(1) P2J,J2P,J2T,P2T,T2J,T2P,TINV,NORMTRN のオプションパラメータのデータ型が VT_VARIANT 以外にも対応できるように修正しました. (コントローラバージョン 3.000 以降) (2) UserExtension コマンドのサンプルを修正しました.
2009-07-31	1.1.4	(1) Controller_Execute に startLog,stpLog,clearLog 追加. (2) Robot_GetVariable に@TYPE 追加 (コントローラバージョン 3.000 以降)
2009-12-02	1.1.5	(1) UserExtension コマンドの引数に使用できる型を追加 (コントローラバージョン 3.000 以降) (2) Robot_Execute に slvChange,slvMove,slvGetMode を追加 (コントローラバージョン 3.000 以降)

		(3) slvMove のサンプルを追加
2010-02-19	1.1.6	(1) Robot_GetVariable に@HIGH_CURRENT_POSITION、@HIGH_CURRENT_ANGLE、@HIGH_CURRENT_TRANS を追加 (コントローラバージョン 3.000 以降) (2) Variable_GetValue に通信サンプルを追加
2010-05-24	1.1.7	対応 RC バージョン表記を追加
2010-11-17	1.1.8	Execute のコマンド文字列修正(大文字・小文字)
2010-11-24	1.1.9	Robot の Execute に GetSrvData, SetSrvData を追加 パケット構造に可変長システム予約領域を追加
2011-09-01	1.1.10	注記追加
2011-11-25	1.1.11	一部誤植修正
2011-12-07	1.1.12	第2章 専用 I/O ポートの処置の説明を修正

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す．

b-CAP は，ORiN で仕様が決められている 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す．そのため，b-CAP は CAP ファミリと同様な以下の特徴を持っています．

- 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

なお，CAP に関しての詳細な情報は，ORiN SDK に含まれる「CAP プロバイダユーザーズガイド」(CAP_ProvGuide_jp.pdf)に記載されていますので参照して下さい．

b-CAP は TCP ストリーム通信として実装されています．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す．

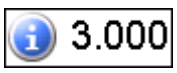
b-CAP を用いて RC7M/J コントローラにアクセスするための関数およびメッセージの仕様は”b-CAP 通信関数”に記載します．TCP ストリームを介して対応するメッセージの通信を行うことで，RC7M/J コントローラのさまざまな関数を使用することができます．

RC7M/J コントローラではバージョン 2.8 以降で b-CAP が使用可能です．

なお，バージョン 3.0 で追加された b-CAP スレーブ機能はロボットコントローラのオプション機能です。機能ライセンスを購入してください。b-CAP スレーブ機能の詳細は「b-CAP スレーブモードの使い方」を参照してください。

また，本仕様書で記載される機能はコントローラの機種およびバージョンにより依存するため，次表の様に記号として本書に表記しています．

表 1 コントローラ別対応記号例

コントローラ		記号	意味
機種	バージョン		
RC7J, RC7M	3.000 以上		コントローラバージョンが 3.000 以上の場合，使用可能です．
—	—		将来の実装に備えて予約されている機能です．

2. セットアップ

2.1. 非常停止スイッチの設置

ロボットコントローラを使用になる前に，非常の際にただちにロボットの運転を停止できるよう，作業者が容易に操作できる位置に非常停止スイッチを設置してください．

非常停止スイッチは，赤色にしてください．

非常停止の機能は，作動させたあと自動的に復帰せず，また他の作業者が不用意に復帰させることができないようにしてください．

非常停止スイッチは，電源スイッチとは別個に設けてください．

2.2. コントローラの初期設定

b-CAP を用いる前に，制御対象となるロボットコントローラの設定を行う必要があります．ロボットコントローラの初期設定にはティーチングペンダントもしくはミニペンダントのどちらかが必要となります．以降，それぞれを使った初期設定の方法について記述します．

2.2.1. ティーチングペンダントを用いた初期設定

ティーチングペンダントを用いたロボットコントローラの初期設定は，以下の手順で行います．

(1) ロボットコントローラを手動モードに設定します．

(2) ORiN オプションを有効にします．[オプション] => [機能拡張]で”1214”を入力し”ORiN”オプションを有効にします．[1]

[1]この”ORiN”オプションを設定することで，b-CAP クライアントや ORiN アプリケーションから変数・I/O へのアクセスが自由に可能となります．変数や I/O へのアクセスはロボット・コントローラ・プログラムなどの状態や内容をよく把握した上で行ってください．特に，変数・I/O への書き込み処理はロボットやプログラムの動作に重大な影響を与える場合があります

また，”ORiN”オプションが有効な状態では，内部自動モードにおいてはエラーレベル 3 以上のエラーが発生するとプログラムが停止しますが，外部自動モードにおいては，エラーレベル 2 以上のエラーが発生するとプログラムが停止します．外部自動モードでの誤操作や誤ったコマンド送信などにご注意ください．

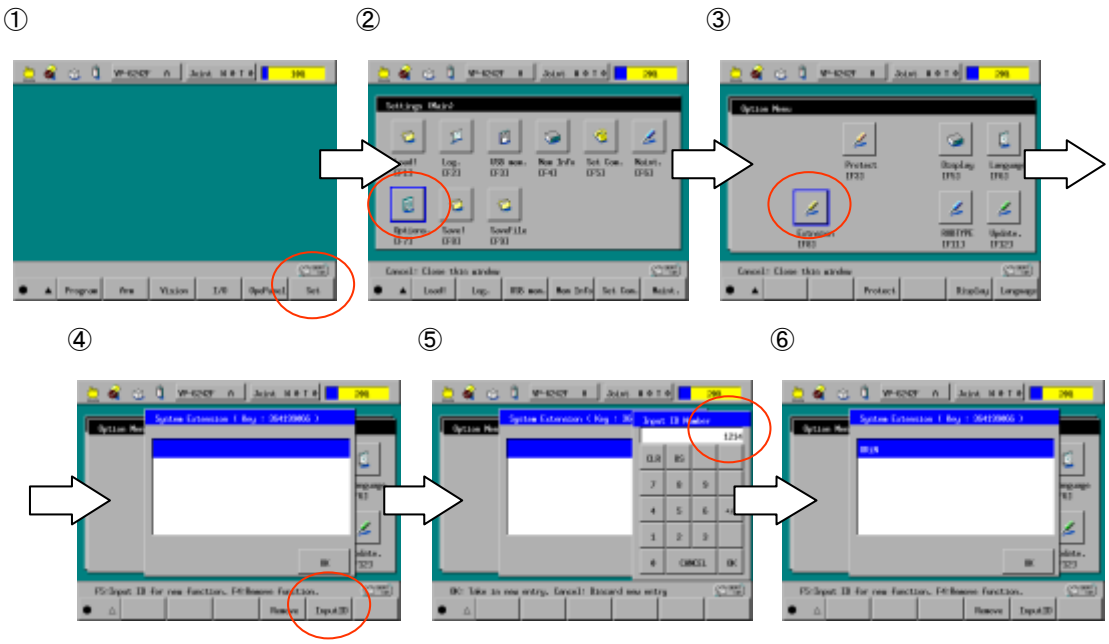


図 2-1 ORiN オプションの入力

(3) コントローラの通信権限を設定します．接続方法として Ethernet を用いる場合は，ティーチングペンダントの通信設定メニュー => 通信権で[Ethernet]に読み込み・書き込み権限を設定します．

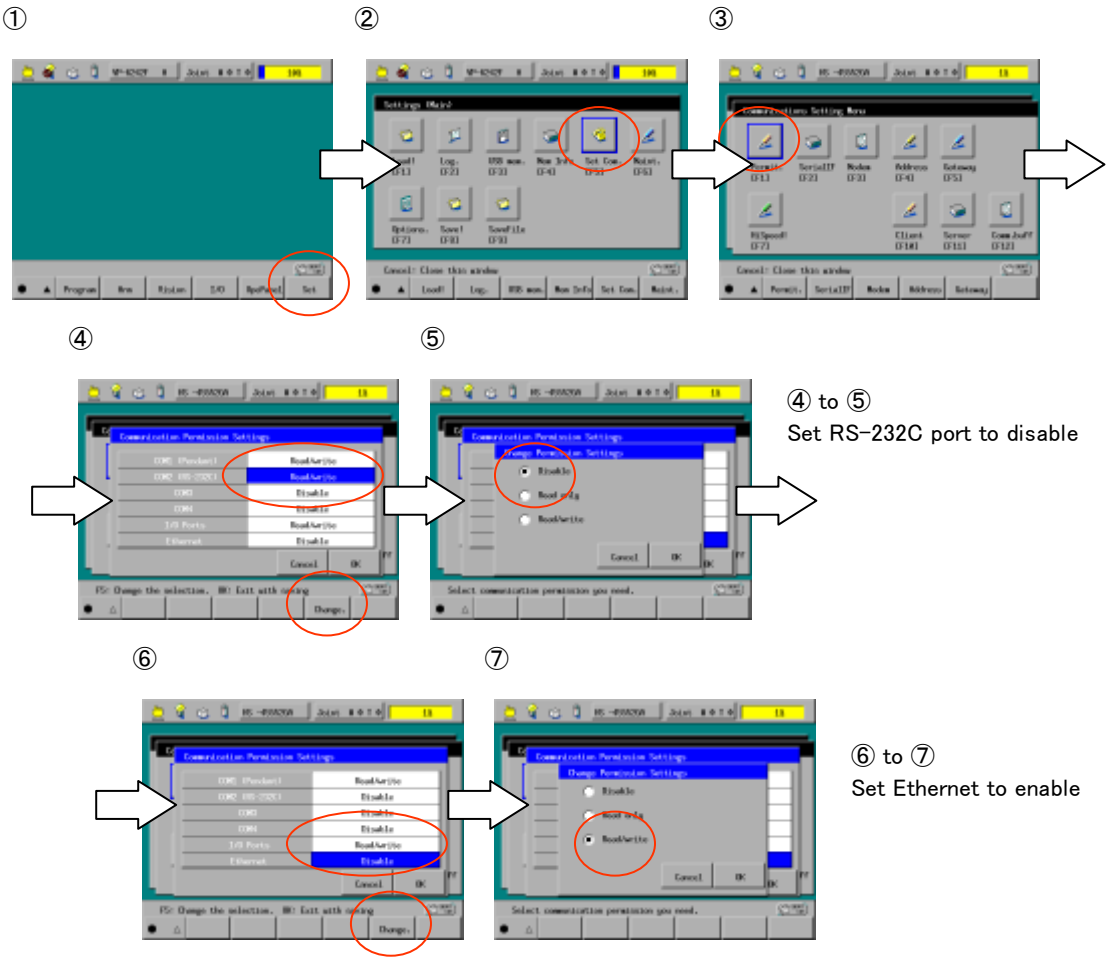


図 2-2 通信権限の設定

(4) b-CAP アプリケーションからモータの ON/OFF やプログラムの起動を行う場合には，コントローラの起動権を設定する必要があります．接続方法として Ethernet を用いる場合は，ティーチングペンダントの通信設定メニュー => 起動権で[Ethernet]に起動権を設定し，その後「F4:IP 設定」メニューでクライアントPCの IP アドレスを設定します．

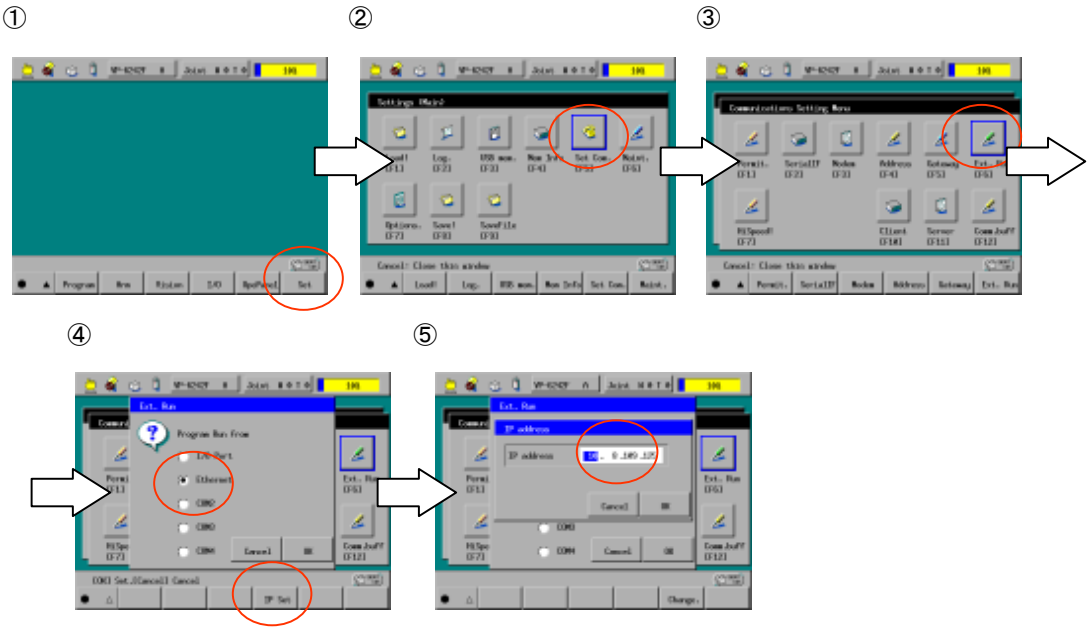


図 2-3 起動権の設定

2.2.2. ミニペンダントを用いた初期設定

ミニペンダントを用いたコントローラの初期設定は，以下の手順で行います．

(1) ロボットコントローラを手動モードに設定します．

(2) ORiN オプションを有効にします．[Aux Function] => [Extension] => [Extension] => [Add]で”1214”を入力し”ORiN”を有効にします．[1]

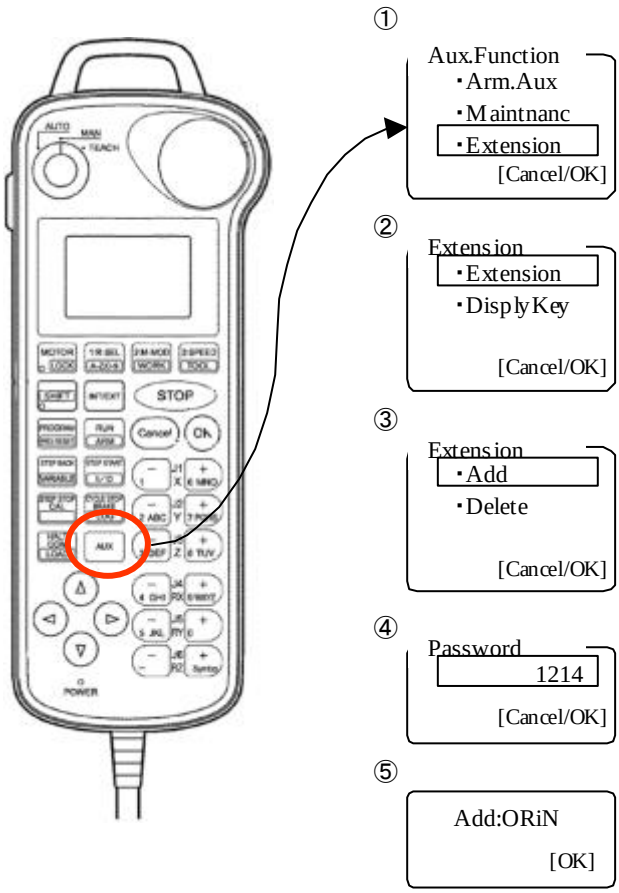


図 2-4 ORiN オプションの入力

(3) ロボットコントローラの通信権限を設定します. 接続方法として Ethernet を用いる場合は, ミニペンダントの[COM Setting] => [Permit]で[Ethernet]に[R/W]を設定します.

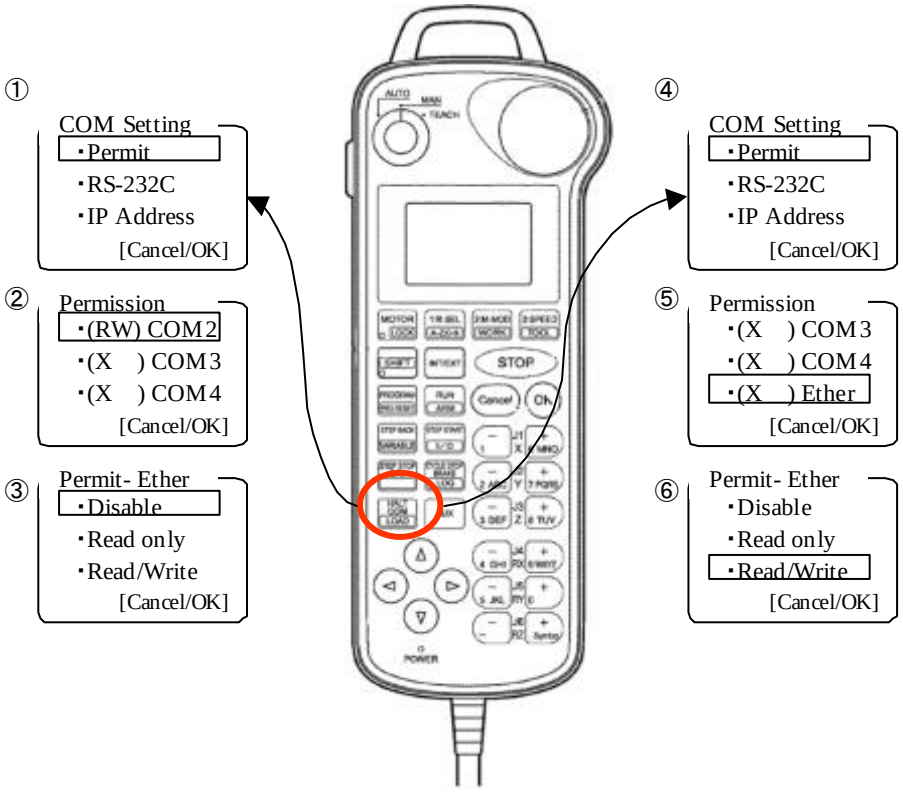


図 2-5 通信権限の設定

(4) b-CAP アプリケーションからモータの ON/OFF やプログラムの起動を行う場合には, コントローラの起動権を設定する必要があります. 接続方法として Ethernet を用いる場合は, ミニペンダントの [COM Setting] => [Ext Run]で[Ethernet]に起動権を設定し, [Client IP]でクライアントPCの IP アドレスを設定します.

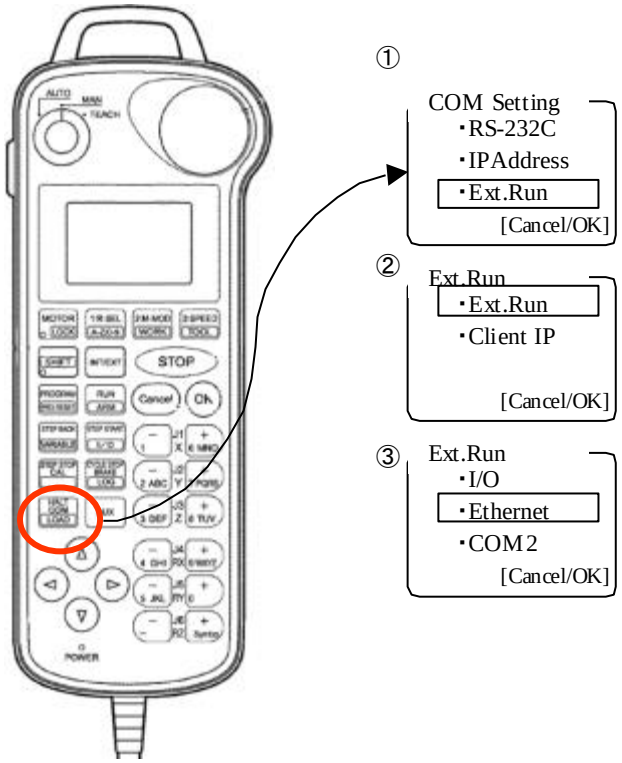


図 2-6 起動権の設定

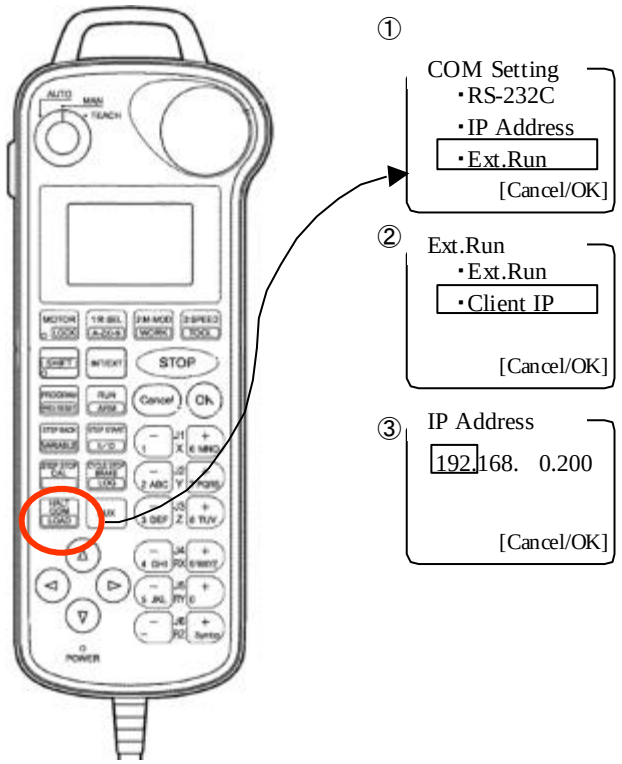


図 2-7 クライアント PC IP アドレスの設定

2.2.3. 通信設定のご注意

b-CAP を使用する際、コントローラは TCP のポート番号の”5007”を使用しています. そのため、RC7M/J のサーバ通信設定(ティーチングペンダントでの操作経路:[F6 設定]―[F5 通信設定.]―[F11 サーバ])でポート番号として”5007”を指定すると、b-CAP の通信が出来なくなりますのでご注意下さい.

2.3. 専用 I/O ポートの処置

ロボットコントローラは多くの専用入力信号を持っています.

外部のPCから b-CAP を用いてロボットやプログラム(PAC)を動作させるには,「ステップ停止(全タスク)信号」及び「瞬時停止信号」をプログラムが動作可能な状態にしておく必要があります.

「ステップ停止(全タスク)信号」及び「瞬時停止信号」はロボットコントローラの構成や I/O の割り付けにより配置されるコネクタや I/O 番号が異なりますのでご使用の構成にあわせて適切な処置を行ってください.

※RC5 ではロボット動作が使用出来ないためこの処置は不要です.

※ロボットやプログラム(PAC)の制御を行わない(変数やファイルアクセスのみの)場合, この処置は不要です.

2.3.1. 標準(I/O 増設ボードがない)構成の場合

I/O の標準構成は MiniI/O コネクタ CN5 のみを使用する場合の構成を指します。この場合, 出荷時の設定では入力点数16点の内8点が専用入力, 出力点数16点の内8点が専用出力として割り付けられています.

専用入力の内, Mini I/O:汎用・専用入出力コネクタ CN5 の端子 No.11 にステップ停止(全タスク)信号(ポート番号 0 番)が割り当てられていますので,

Mini I/O:汎用・専用入出力コネクタ CN5 の端子 No.11 をクローズしてください.

⇒ ステップ停止(全タスク)信号(ポート番号 0 番)が ON し,ロボット及びプログラムが動作に可能になります.

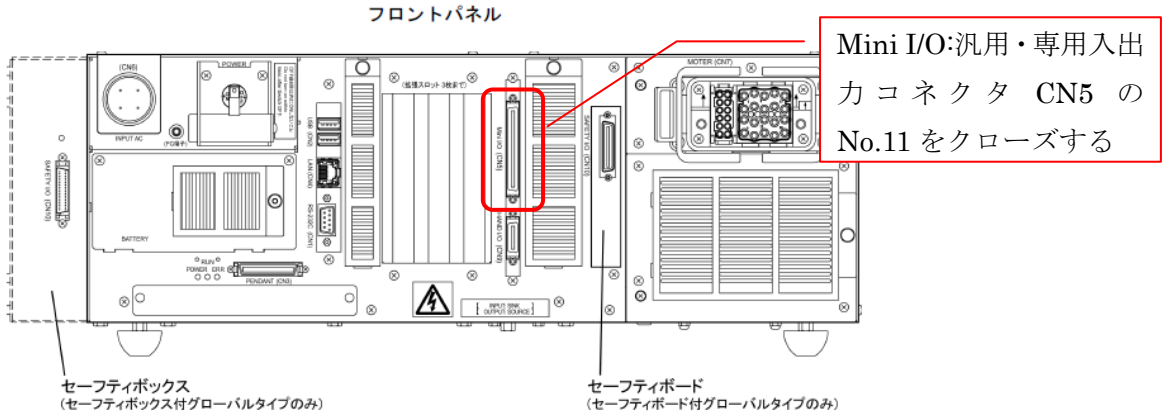


図 2-8 ステップ停止(全タスク)の処置例

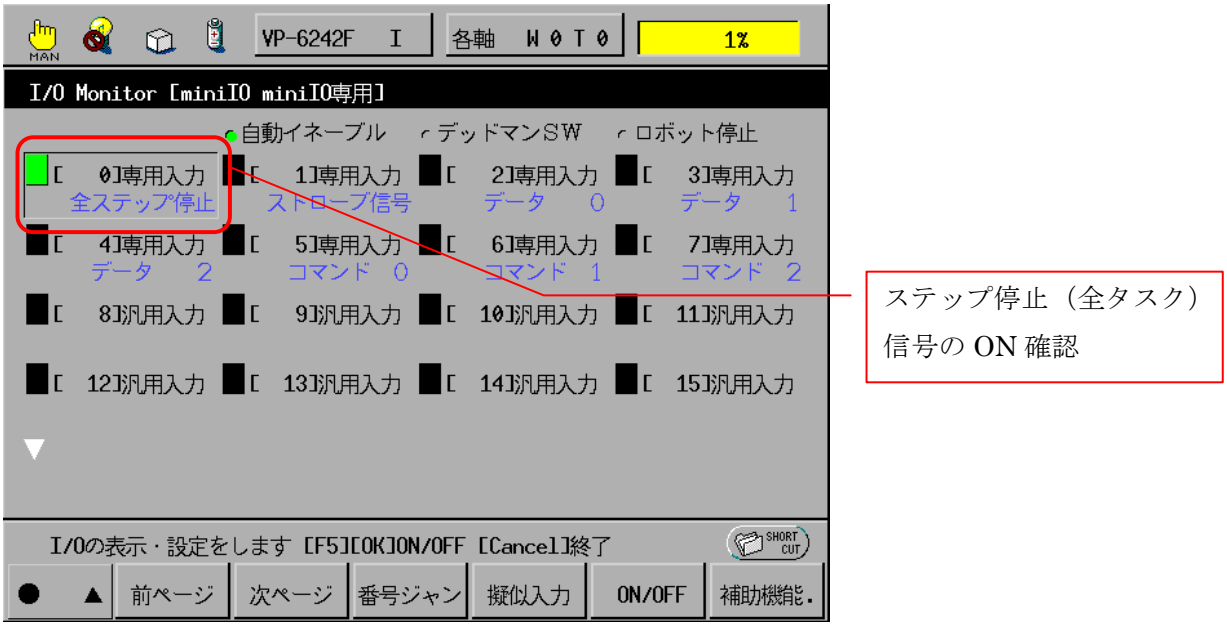


図 2-9 処置後の状態確認

瞬時停止信号はないため処置は不要です.

2.3.1.1. ミニ I/O 全汎用化オプション

ミニ IO に割り付けられている専用入力・出力を必要としない場合でかつロボットコントローラがバージョン 2.90 以上の場合，ミニ I/O の割付を全汎用化して，従来のステップ停止専用信号に対する処置を簡略することが出来ます.

ティーチングペンダントを用いたロボットコントローラの設定は，以下の手順で行います.

(1) ロボットコントローラを手動モードに設定します.
(2) ミニ IO 全汎用化オプションを有効にします. [オプション] => [機能拡張]で"**6319**"を入力します.
(3) ロボットコントローラの電源を切った後，再起動します.

※【注意事項1】 ミニ IO 全汎用化オプションを有効にした場合，専用信号であるステップ停止信号の割付は無くなるためステップ停止を行うことが出来なくなります.

※【注意事項 2】 ミニ IO 全汎用化オプションを有効にすることで，ステップ停止信号に対する処置は不要になりますが，自動イネーブル信号と非常停止信号に対する処置は必要で，省略することは出来ません.

2.3.2. オプション（I/O 増設ボードがある）構成の場合

I/O 増設ボード（パラレル I/O,DeviceNet,CC-Link,PROFIBUS 等）がある構成の場合はロボットコントローラの「設置・保守ガイド」および「オプション機器説明書」の「第2部 RC7M 用 I/O 増設ボード」を参照の上，「ステップ停止（全タスク）信号」及び「瞬時停止信号」を ON してください.

2.4. ロボットコントローラの起動権

2.4.1. ロボットコントローラの起動権に関する基礎知識

b-CAP アプリケーションからモータの ON/OFF やプログラムの起動を行うには起動権の設定が必要です. (2.2.1 と 2.2.2 を参照)また, 安全のため, ロボットコントローラは選択された一つの機器からのみ外部から制御可能となります(Single point of control). また, b-CAP アプリケーションからのモータの ON/OFF やタスクの起動はロボットコントローラが外部自動モードの状態でのみ実行可能となります.

コントローラの起動権は以下の図のように遷移します.

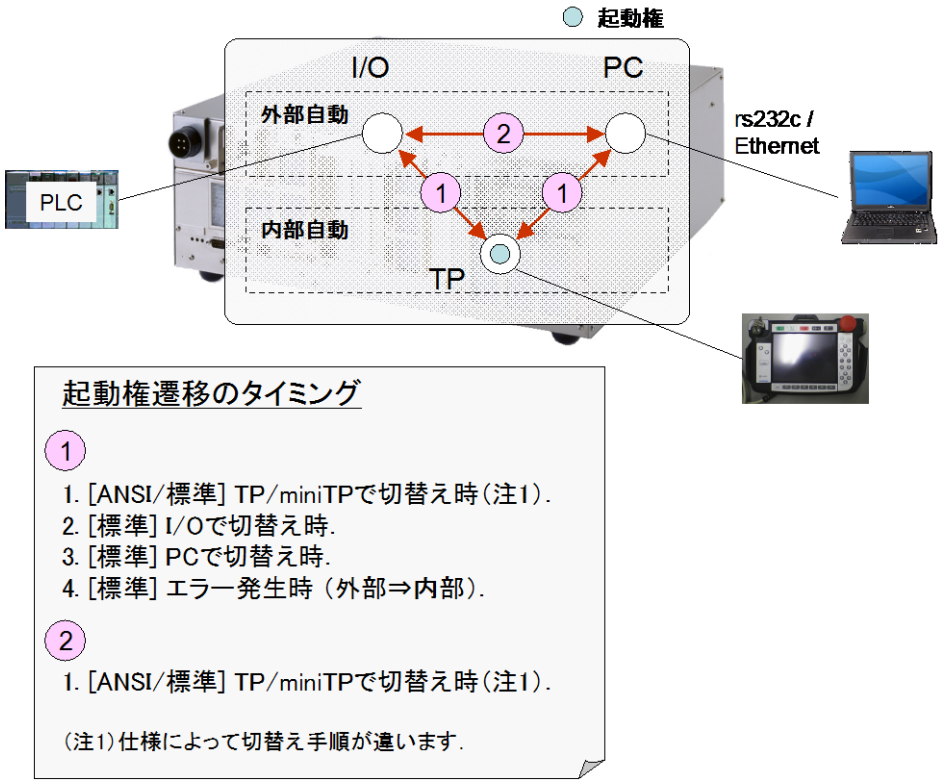


図 2-10 起動権の遷移

2.4.2. ANSI 仕様ロボットコントローラにおける注意点

前述のように, b-CAP アプリケーションからのモータの ON/OFF やタスクの起動はロボットコントローラが外部自動モードの状態でのみ実行可能となります.

ANSI 仕様のロボットコントローラでは, I/O 補助機能メニューの単一位置制御設定にて「外部自動」を選択しておかないと, 外部自動モードの状態にならないことに注意してください.

以下に ANSI 仕様ロボットコントローラでの外部自動モードの選択手順を示します.

(1) ティーチングペンダントを用いた選択手順

この選択処理が完了した後に自動モードへ切り替えを行うと，選択したモード（内部/外部）に切り替わります．

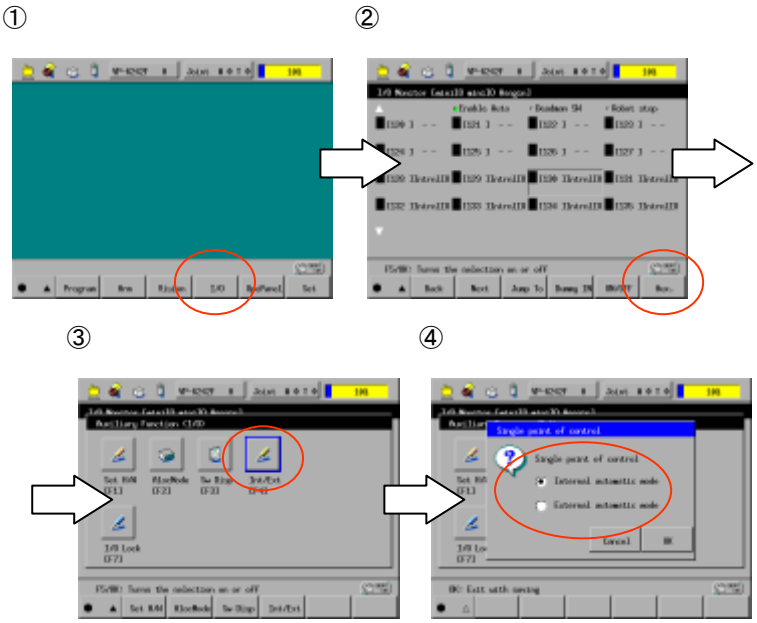


図 2-11 ペンダントを用いた内部/外部自動モード選択

(2) ミニペンダントを用いた選択手順

この選択処理が完了した後に自動モードへ切り替えを行うと，選択したモード（内部/外部）に切り替わります．

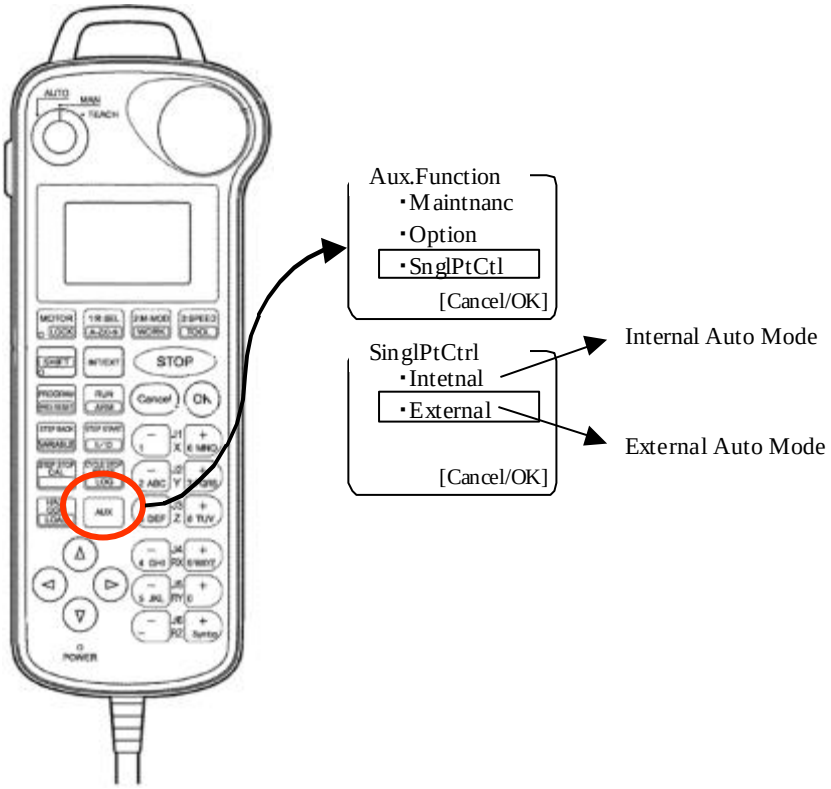


図 2-12 ミニペンダントを用いた内部/外部自動モード選択

2.5. PAC プログラムの転送

b-CAP でロボット動作命令などロボット関連命令や PAC プログラムを実行するには，予めロボットコントローラに必要な PAC プログラム（RobSlave.pac，RobSlave.h，UserExtention.pac）を転送して実行しなくてはなりません．

PAC プログラムをコントローラへ転送するには，付属ツールの RobMaster.exe を使用します．

これらの方法ではミニペンダント(miniTP)またはティーチングペンダント(TP)が必要です．

RobMaster.exe[2]はロボットコントローラの状態表示と RobSlave タスクの制御を PC から直接と行うためのツールです．RobMaster.exe は以下のフォルダにインストールされています．

<ORiN2 インストールフォルダ>¥CAO¥ProviderLib¥DENSO¥NetwoRC¥Bin

下記にセットアップ手順を示します．

1. コントローラを起動し，[手動]モードにする
2. RobMaster を起動する
   スタートメニュー→すべてのプログラム→ORiN2→CAO→ProviderLib→RobMaster
3. コントローラの IP アドレスを入力し，［接続］ボタンでコントローラと接続する
4. RobMaster の[セットアップ]ボタンを押して指示に従って必要な PAC プログラムを転送する

2.6. RobMaster の機能

付属ツール RobMaster はロボットコントローラと接続して，下記の機能を提供しています．

1. ORiN 用にロボットコントローラをセットアップする
2. ロボットコントローラの RobSlave タスクの起動および停止を行う
3. ロボットコントローラのモータ ON/OFF を行う
4. ロボットコントローラのエラー表示とエラークリアを行う
5. ロボットコントローラの状態表示を行う

以下に RobMaster の機能紹介を示します．

[2] <ORiN2 SDK インストールフォルダ>¥CAO¥ProviderLib¥DENSO¥NetwoRC¥Bin¥RobMaster.exe

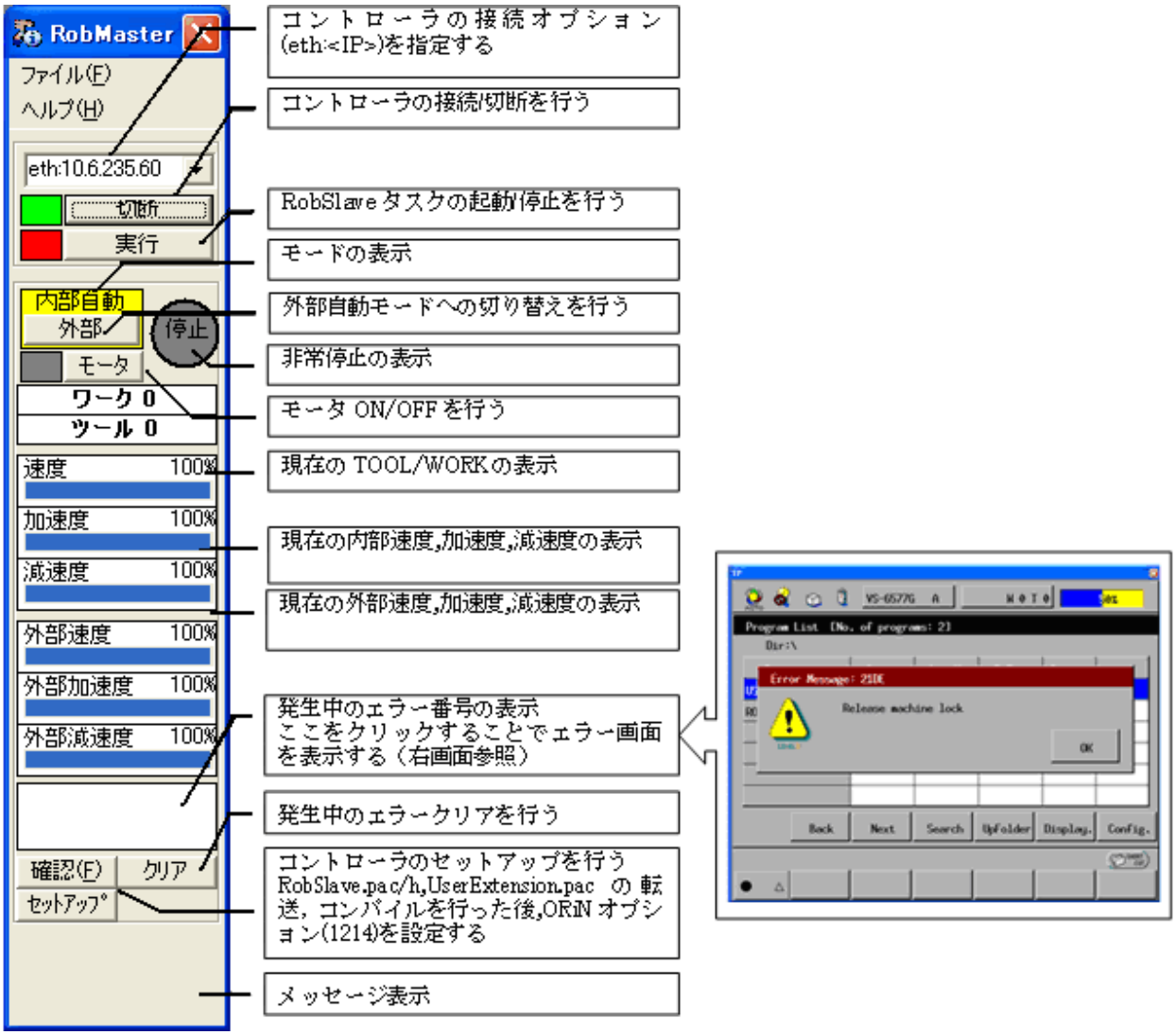


図 2-13 RobMaster の機能紹介

2.7. b-CAP Tester

ORiN2 SDK に付属しているツール b-CAP Tester を使用すると，コントローラと送受信されるパケットを確認できます．

b-CAP Tester(b-CAP Tester.exe)は以下のフォルダに格納されています．

ORiN2¥CAP¥b-CAP¥CapLib¥DENSO¥RC7¥Bin

図 2-12 に b-CAP Tester の機能紹介を示します．

図 2-14 b-CAP Tester の機能紹介

3. 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)に対応します．以下にその構造を図で示します．

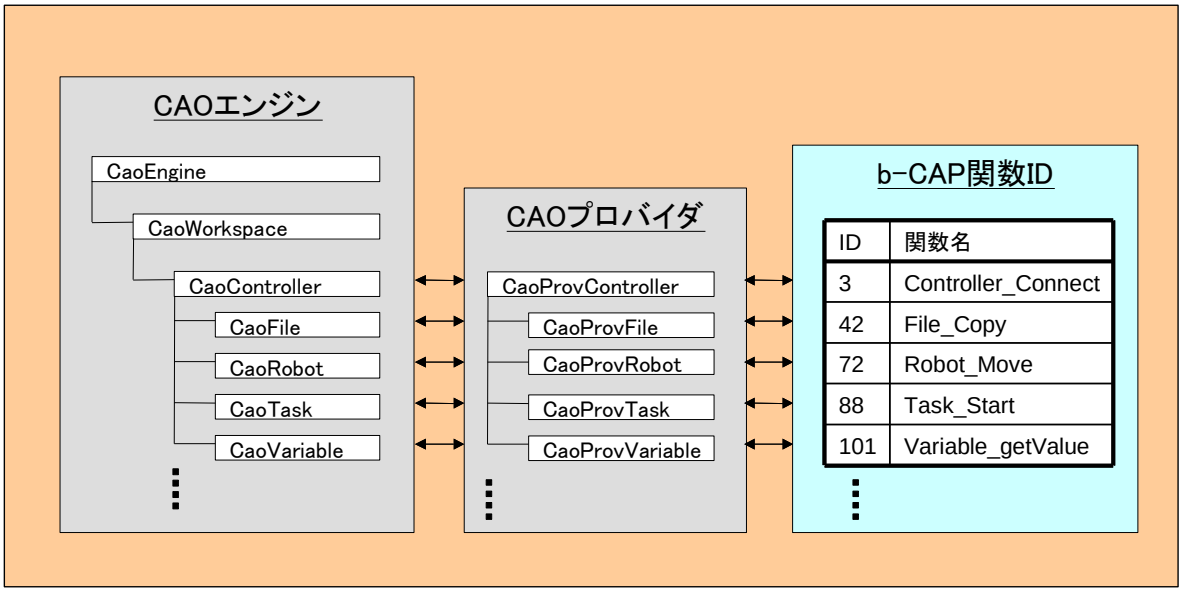


図 3-1 b-CAP の構造

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ます．

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たあとサーバ側に送信し，応答メッセージを受信して実行結果を確認します．

b-CAP サーバは，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します．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します．

クライアント，サーバの処理手順については，5 通信手順で詳細を説明します．

以下に b-CAP による接続例を示します．

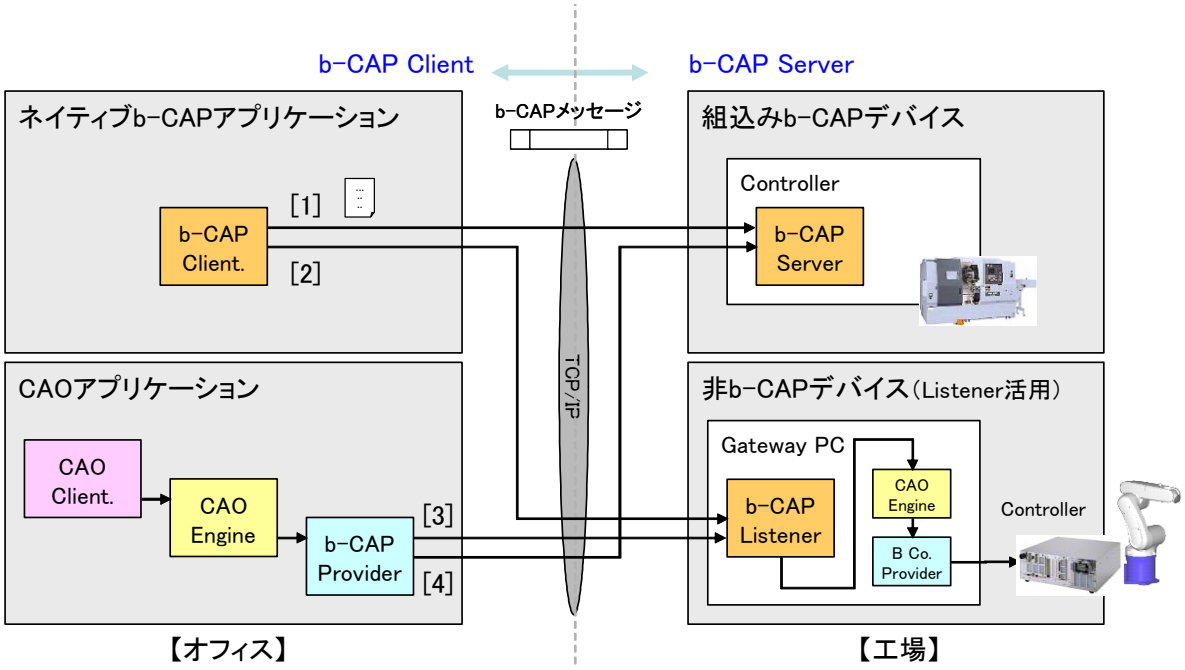

b-CAP Client
b-CAP Server
b-CAPメッセージ
ネイティブb-CAPアプリケーション
[1]
b-CAP Client.
[2]
組込みb-CAPデバイス
Controller
b-CAP Server
TCP/IP
CAOアプリケーション
CAO Client.
CAO Engine
b-CAP Provider
[3]
[4]
非b-CAPデバイス(Listener活用)
Gateway PC
b-CAP Listener
CAO Engine
B Co. Provider
Controller
【オフィス】
【工場】

図 3-2 b-CAP 接続例

4. パケット構造

4.1. 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ます

.

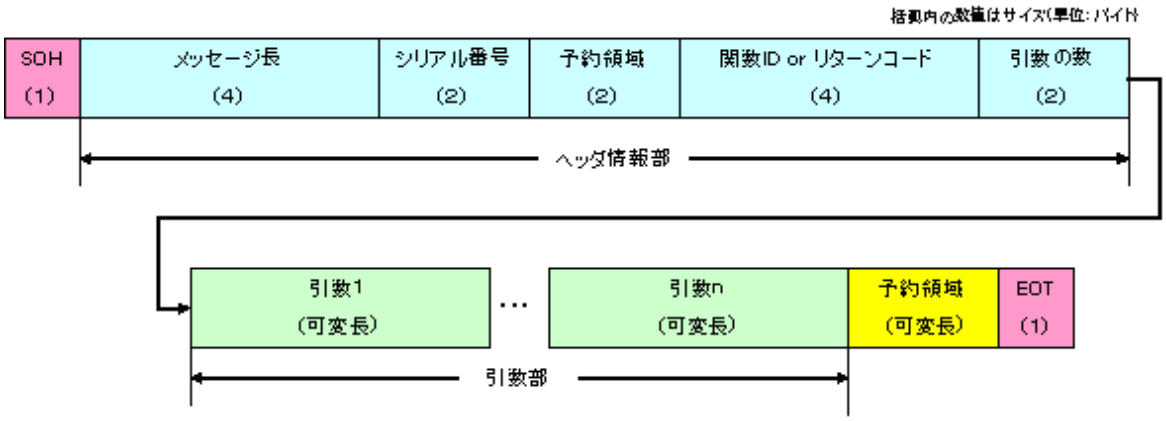


図 4-1 パケット構造

以下はパケットの各要素の説明です. 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)で格納されます.

- ヘッダ　パケットの先頭につけるスタートコードです. SOH(0x01)を使用します.
- メッセージ長　パケット全体のデータ長です. 符号なし長整数型(DWORD)を格納します. ヘッダからターミネータまでの長さを格納します.
- シリアル番号　メッセージのシリアル番号が格納されます. 符号なし短整数型(WORD)を格納します. 1 から WORD_MAX(0x0001～0xFFFF)までの任意の数値が入ります.
  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必要があります.
- 予約領域　システムで予約されている領域です. この領域は常に 0 を格納します.
- 関数 ID　呼び出し関数を識別する ID です. 符号なし長整数型(DWORD)を格納します. 呼び出しメッセージのときのみ使用します. (4.3 参照)
- リターンコード　呼び出し関数の実行結果コードで, 符号なし長整数型(DWORD)を格納します. 応答メッセージのときのみ使用します. (4.4 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です. 符号なし短整数型(WORD)を格納します.
- 引数 n　n 番目の引数部です. (4.2 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で, EOT(0x04)を使用します.

4.2. 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ます．
以下に引数部の構造を示します．

括弧内の数値はサイズ（単位：バイト）

引数データ長 （4）	データ型 （2）	要素数 （4）	データ （データ型ごとのサイズ）

図 4-2 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ます．データ部は共通部の情報を元に構造が決定されます．

引数部の各要素を以下に説明します．

- 引数データ長　　データ型，要素数，データの合計バイト数です．ただし，引数データ長は含まれません．符号なし長整数型（DWORD）を格納します．データ型ごとのサイズは表 4-1 を参照してください．
- データ型　　引数のデータ型です．符号なし短整数型（WORD）を格納します．使用できるデータ型は表 4-1 を参照してください．
- 要素数　　引数の要素数です．符号なし長整数型（DWORD）を格納します．データ型に VT_ARRAY が使用されていないときは，常に 1 になります．
- データ部　　引数のデータです．データ型によって使用するサイズが異なります．各データのサイズは表 4-1 を参照してください．また，データ部に格納される情報の構造については，図 4-3 を参照してください．

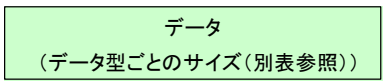
表 4-1 使用可能なデータ型

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	2	エラーコード
VT_UI1	17	1	バイト型
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 文字列長および文字列からなり，文字列長の後ろに文字列を Unicode（1 文字 2 バイト）で格納します．文字列長は文字列のバイト数を表しています．
VT_VARIANT	12	-	Variant 型 データに引数部と同じものが入ります． VT_ARRAY のときのみ使用が可能です．
VT_ARRAY	0x2000	-	配列 他のデータ型との論理和で指定します． 指定した型のデータを連続で格納します．

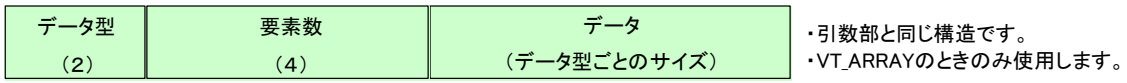
括弧内の数値はサイズ（単位：バイト）

・VT_BSTR，VT_VARIANT，VT_ARRAY以外のデータ



・VT_BSTR

・VT_VARIANT



・VT_ARRAY

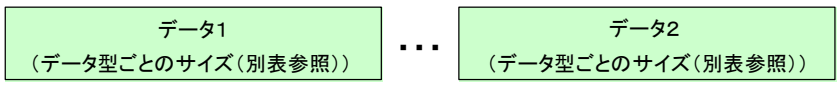


図 4-3 データの構造

4.3. 関数 ID

b-CAP では関数 ID は以下のように割り当てられています．

表 4-2 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可能です．

b-CAP の既定の関数の一覧を以下に示します．

表 4-3 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
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	コントローラ変数取得
17	Controller_Execute	コントローラの拡張関数の実行
62	Robot_GetVariable	ロボット変数取得
64	Robot_Execute	ロボットの拡張関数実行
66	Robot_Change	ロボットの CHANGE 文実行
72	Robot_Move	ロボットの MOVE 文実行
84	Robot_Release	ロボットの解放
85	Task_GetVariable	タスク変数取得
88	Task_Start	タスクの開始
89	Task_Stop	タスクの停止
99	Task_Release	タスクの解放
101	Variable_GetValue	変数の値取得
102	Variable_PutValue	変数の値設定
111	Variable_Release	変数の開放

4.4. 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当てられています．

表 4-4 リターンコードの割り当て

リターンコード	説明
0x00000000～0x8000FFFF 0x80010000～0x800101FF 0x80070000～0x8007FFFF	既定リターンコード，予約領域
0x80040200～0x8004FFFF	ユーザ定義エラー

以下に示す「既定リターンコード」以外のリターン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コードを割り当てることが可能です．

表 4-5 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	未実装の機能です．
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．
0x8000FFFF	E_UNEXPECTED	致命的エラーが発生しました．
0x80010001	E_INVALIDRCVPACKET	ロボットコントローラが受信した n パケットが不正です． このエラーが発生した際には、ロボットコントローラは自動的に接続を切断します． ロボットコントローラへ送信したパケットを確認してください．
0x80010002	E_INVALIDSNDPACKET	送信パケットが不正です．
0x80010003	E_INVALIDARGTYPE	受信パケット内の引数型が不正です．
0x80010004	E_ROBOTISBUSY	ロボットが動作中に新たな動作命令を受信しました．動作できません．
0x80010005	E_INVALIDCOMMAND	不正なコマンド文字列を受信しました．実行できません．
0x80010011	E_PACKETSIZEOVER	受信パケットサイズが不正です．(>16Mbytes)
0x80010012	E_ARGSIZEOVER	受信パケットの引数サイズが不正です．(>16Mbytes)
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.

5. 通信手順

5.1. 通信シーケンス

b-CAP のシーケンスは，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ります．サーバ側は，要求パケットの関数を実行し， 応答パケットをクライアントに返します．

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，常に同期をとる必要があります．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を推奨します．

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特に規定がありません．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りますので注意してください．

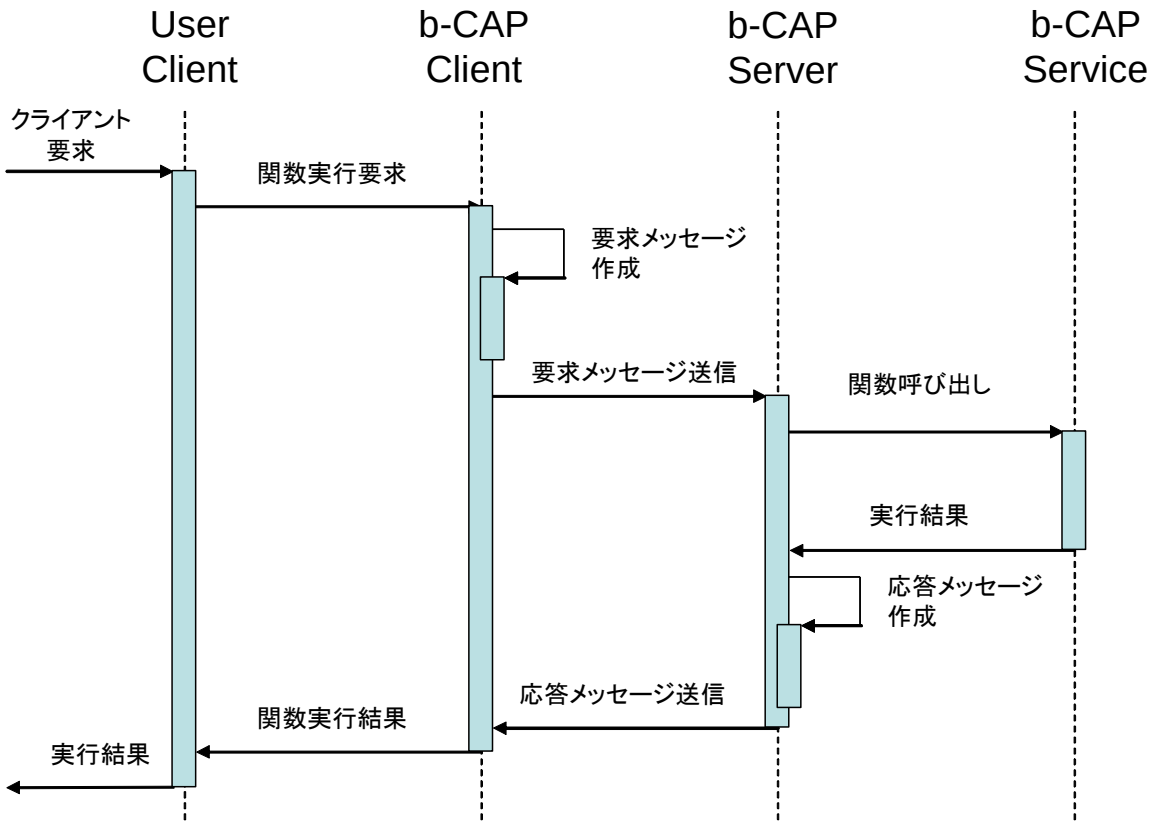


図 5-1 通信シーケンス

5.2. 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.

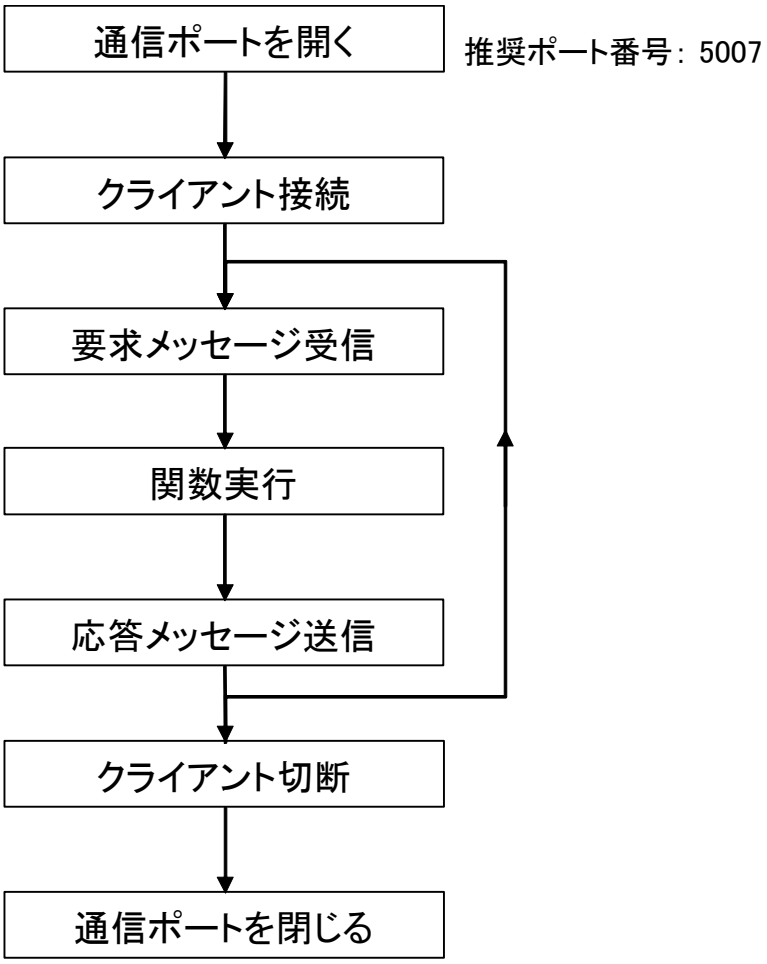


図 5-2 サーバの通信手順

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ちます. このとき開く TCP のポート番号は“5007”を使用します.

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します. 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します.

5.3. 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.

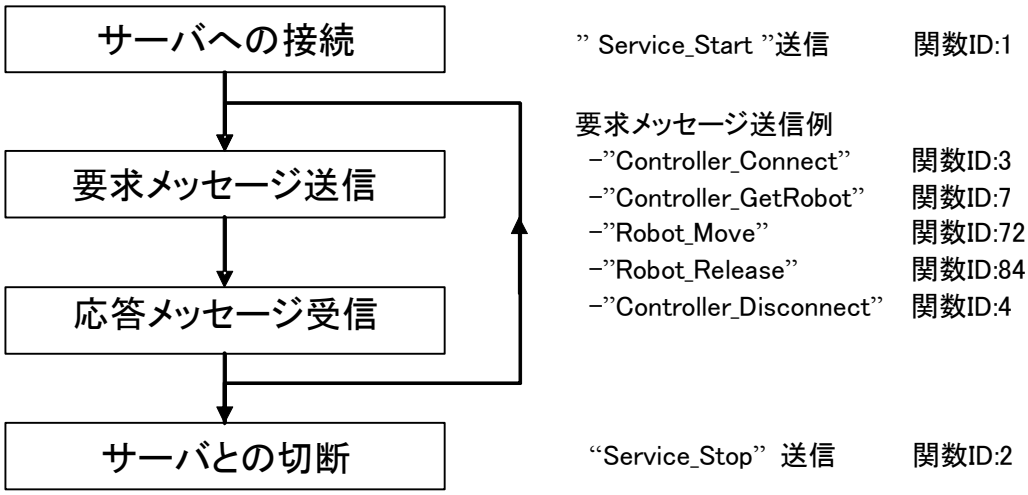


図 5-3 クライアントの通信手順

クライアントは最初にサーバへの接続し，セッションを確立します．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ります．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には注意が必要です．

5.4. 使用上のご注意

ロボットコントローラ内部では b-CAP 通信は PAC 言語よりも高い優先度で実行されています．そのため、b-CAP 通信を高速で行う事で、PAC 言語の実行速度にも影響を与えることがあり、結果としてロボットの動作精度に影響を与えることもあります．

また、b-CAP を高い頻度で使用している際に、WINCAPS にてﾌｧｲﾙ送受信をはじめとする通信を行うと、通信エラーが発生することがあります．

その際には、b-CAP の通信を停止するか、使用する頻度を落として使用下さい．

6. b-CAP 通信関数

6.1. サーバサービスの開始/停止

6.1.1. Service_Start

関数　HRESULT Service_Start()
関数 ID　1
引数　なし
戻り値　表 4-5 既定のリターンコード一覧　参照
説明　b-CAP サーバサービスを開始します.
参照項目　Service_Stop

通信サンプル 1				
サーバサービスを開始します.				
送信パケット	クライアント→サーバ: 01 10 00 00 00 01 00 00　00 01 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		
受信パケット	サーバ→クライアント: 01 10 00 00 00 01 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

6.1.2. Service_Stop

関数　HRESULT Service_Stop ()
関数 ID　2
引数　なし
戻り値　表 4-5 既定のリターンコード一覧　参照
説明　b-CAP サーバサービスを停止します.
参照項目　Service_Stop

通信サンプル 1				
サーバサービスを停止します.				
送信パケット	クライアント→サーバ: 01 10 00 00 00 08 00 00　00 02 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		
受信パケット	サーバ→クライアント: 01 10 00 00 00 08 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

6.2. コントローラオブジェクト

6.2.1. Controller_Connect

関数　HRESULT Controller_Connect ()

関数 ID　3

引数

[in]	BSTR	bstrCtrlName	コントローラ名（未使用）
[in]	BSTR	bstrProvName	プロバイダ名（未使用）
[in]	BSTR	bstrPcName	プロバイダの実行マシン名（未使用）
[in]	BSTR	bstrOption	オプション文字列 = "<option1>, <option2>, …" （未使用）
[out]	long	hController	コントローラハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　コントローラに接続し, コントローラオブジェクのハンドル（hController）を取得します.

参照項目　Controller_Disconnect

通信サンプル 1	
コントローラハンドル（hController）を返します.	
送信パケット	クライアント→サーバ: 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 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

	引数	説明	データ型	値
		バイナリ		
	bstrCtrlName	コントローラ名（未使用）	VT_BSTR	空文字列
		00 00 00 08 00 01 00 00 00 00 00 00 00 0A		
	bstrProvName	プロバイダ名（未使用）	VT_BSTR	空文字列
		00 08 00 01 00 00 00 00 00 00 00 0A 00 00		
	bstrPcName	クライアント PC 名（未使用）	VT_BSTR	空文字列
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		
	bstrOption	接続オプション（未使用）	VT_BSTR	空文字列
		00 00 00 00 00 00 00 00 0A 00 00 00 08 00 01		
受信 パケット	サーバ→クライアント: 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			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル	VT_I4	0x000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		

6.2.2. Controller_Disconnect

関数　　HRESULT Controller_Disconnect ()

関数 ID　　4

引数　　[in]　long　hController　コントローラハンドル

戻り値　　表 4-5 既定のリターンコード一覧 参照

説明　　クライアントを“hController”で指定したコントローラから切断します.

参照項目　　Controller_Connect

通信サンプル 1				
クライアントをコントローラハンドル（hController）で指定したコントローラから切断します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		

	hController	コントローラハンドル (Controller_Connect 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 04		0A
受信 パケット	サーバ→クライアント: 01 10 00 00 00 03 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

6.2.3. Controller_GetVariable

関数　HRESULT Controller_GetVariable ()

関数 ID　9

引数

[in]	long	hController	コントローラハンドル
[in]	BSTR	bstrName	変数名
[in]	BSTR	bstrOption	オプション文字列
[out]	long	hVariable	変数ハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　コントローラのシステム変数オブジェクトのハンドル(hVariable)を取得します.
このハンドル hVariable を使用して, 変数を読書きできます.

参照項目　Variable_GetValue
Variable_PutValue
Robot_GetVariable
Task_GetVariable

“bstrName”に下記の変数名を入れることにより, コントローラ変数を取得することができます.

表 6-1 使用可能な変数一覧

変数名	データ型	説明	属性	
			取得	設定
I 例) “I10”	VT_I4	I 型変数 変数名の後に変数番号(0～)をつけて下さい.	∨	∨
F 例) “F10”	VT_R4	F 型変数 変数名の後に変数番号(0～)をつけて下さい.	∨	∨
D	VT_R8	D 型変数	∨	∨

例）“D10”		変数名の後に変数番号(0～)をつけて下さい.		
V 例）“V10”	VT_ARRAY | VT_R4	V 型変数 変数名の後に変数番号(0～)をつけて下さい. この変数は 3 つの要素を含んでいます.	∨	∨
P 例）“P10”	VT_ARRAY | VT_R4	P 型変数 変数名の後に変数番号(0～)をつけて下さい. この変数は7つの要素を含んでいます.	∨	∨
J 例）“J10”	VT_ARRAY | VT_R4	J 型変数 変数名の後に変数番号(0～)をつけて下さい. この変数はロボットタイプによって 4 つもしくは 6 つの要素を含んでいます.	∨	∨
T 例）“T10”	VT_ARRAY | VT_R4	T 型変数 変数名の後に変数番号(0～)をつけて下さい. この変数は 10 の要素を含んでいます.	∨	∨
S 例）“S10”	VT_BSTR	S 型変数 変数名の後に変数番号(0～)をつけて下さい.	∨	∨
IO 例）“IO10”	VT_BOOL	IO 変数 変数名の後に変数番号(0～)をつけて下さい.	∨	∨
TOOL 例）“TOOL63”	VT_ARRAY | VT_R4	ツール設定: 変数名の後に変数番号(0～63)をつけて下さい. 注:ID No.0(TOOL0) は読み取り専用です.	∨	∨
WORK 例）“WORK7”	VT_ARRAY | VT_R4	ワーク設定: 変数名の後に変数番号(0～7)をつけて下さい. 注:ID No.0(WORK0)は読み取り専用です.	∨	∨
AREA 例）“AREA7”	VT_ARRAY | VT_R4	エリア設定: 変数名の後に変数番号(0～7)をつけて下さい.	∨	∨
_ITP 例）“_ITP10”	VT_I4	ITPCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨

_PAC 例）“_PAC10”	VT_I4	PACCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
_DIO 例）“_DIO10”	VT_I4	DIOCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
_ARM 例）“_ARM10”	VT_I4	ARMCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
_SRV 例）“_SRV10”	VT_I4	SRVCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
_SPD 例）“_SPD10”	VT_I4	SPDCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
_VIS 例）“_VIS10”	VT_I4	VISCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨

_COM 例）“_COM10”	VT_I4	COMCNF: 変数名の後に変数番号(0～)をつけて下さい. 注 1:変数番号 0 の値はパラメータの総数を表しています. ここには値を入れないで下さい. 注 2:間違った値を入力すると, コントローラに重大な障害を引き起こすおそれがあります.	∨	∨
@MODE	VT_I2	1: 手動 2: ティーチチェック 3: 内部自動 4: 外部自動	∨	-
@EMERGENCY_STOP	VT_BOOL	true = 非常停止スイッチ ON false =非常停止スイッチ OFF	∨	-
@ERROR_CODE	VT_I4	発生したエラーの番号 エラーがない場合は 0 を返します.	∨	-
@ERROR_DESCRIPTION	VT_BSTR (コントローラバージョンが V2.624 以前は VT_ARRAY|VT_UI1 で返します.)	発生したエラーの説明 注 1:エラーが発生していない場合は, “Undefined error”を返します. このエラーの言語はティーチングペンダントの言語設定に依存します.	∨	-
@AUTO_ENABLE	VT_I2	1 = 自動イネーブル信号 ON 0 = 自動イネーブル信号 OFF	∨	-
@PROTECTIVE_STOP	VT_I2	1 = 保護停止 ON 0 = 保護停止 OFF	∨	-
@DEADMAN_SW	VT_I2	1 = デッドマンスイッチ ON 0 = デッドマンスイッチ OFF	∨	-
@VERSION	VT_BSTR	ロボットコントローラのﾊﾞｰｼﾞｮﾝ文字列を返します	∨	-

通信サンプル 1	
変数ハンドル(hVariable)を返します. このハンドルを使用して変数を読書きすることができます.	
送信 パケット	クライアント→サーバ: 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 00 04

	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル (Controller_Connect 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrName	変数名	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	空文字列
		00 00 00 08 00 01 00 00 00 00 00 00 00 0A		
受信 パケット	サーバ→クライアント: 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			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数“I1”のハンドルを返します.	VT_I4	0x00020001
		00 00 00 03 00 01 00 00 00 01 00 02 00 04 0A		

6.2.4. Controller_Execute

関数　　HRESULT Controller_Execute()

関数 ID　　17

引数				
	[in]	long	hController	コントローラハンドル
	[in]	BSTR	bstrCommand	コマンド名
	[in]	VARIANT	vntParam	パラメータ
	[out]	long	pVal	結果値

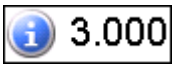
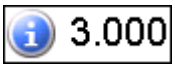
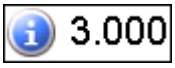
戻り値　　表 4-5 既定のリターンコード一覧 参照

説明　　コントローラ(hController)のコマンドを実行します.

“bstrCommand”に下記のコマンド名を入れることにより, コマンドを実行することができます.

表 6-2 コマンド実装一覧

コマンド	パラメータ	戻り値	動作
GetAutoMode	なし	VT_I2: Mode(	自動モードの取得

		0:Unkown 1:内部自動 2:外部自動)	
PutAutoMode	VT_I2: Mode(1:内部自動 2:外部自動)	なし	自動モードの設定
ClearError	VT_I4 ErrorCode	VT_I2: Result Code	エラークリア “ClearError” には時間がかかるので注意してください. 1 秒ほどかかることがあります.
SuspendAll	なし	なし	すべての PAC プログラムを中断します.
ResetAll	なし	なし	PAC プログラムを停止させリセットします.
StartLog 	なし	なし	スタートログ実行
StopLog 	なし	なし	ストップログ実行
ClearLog 	なし	なし	クリアログ実行

通信サンプル 1				
コントローラのコマンドを実行します. “ClearError” コマンドを実行する場合の例を以下に示します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル (Controller_Connect 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	コマンド文字列	VT_BSTR	“ClearError”

		1E 00 00 00 08 00 01 00 00 00 14 00 00 00 43 00 6C 00 65 00 61 00 72 00 45 00 72 00 72 00 6F 00 72 00		
	vntParam	クリアすべきエラーのコード (RC7 では 0x00000000 を入れてください.)	VT_I4	0x00000000
		0A 00 00 00 03 00 01 00 00 00 00 00 00 00 04		
受信 パケット	サーバ→クライアント: 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			
	引数	説明	データ型	値
		バイナリ		
	Result	エラークリアの結果(1 = 成功)	VT_I2	0x0001
		01 00 08 00 00 00 02 00 01 00 00 00 01 00		

6.2.5. Controller_GetTaskNames

関数　HRESULT Controller_GetTaskNames ()

関数 ID　14

引数

[in]	long	hController	コントローラハンドル
[in]	BSTR	bstrOption	オプション文字列
[out]	VARIANT	TaskNames	PAC プログラム一覧

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　PAC プログラムの一覧を取得します.

注 1:この関数では, 結果値(=PAC プログラム一覧)は **VT_VARIANT | VT_ARRAY** 型です.

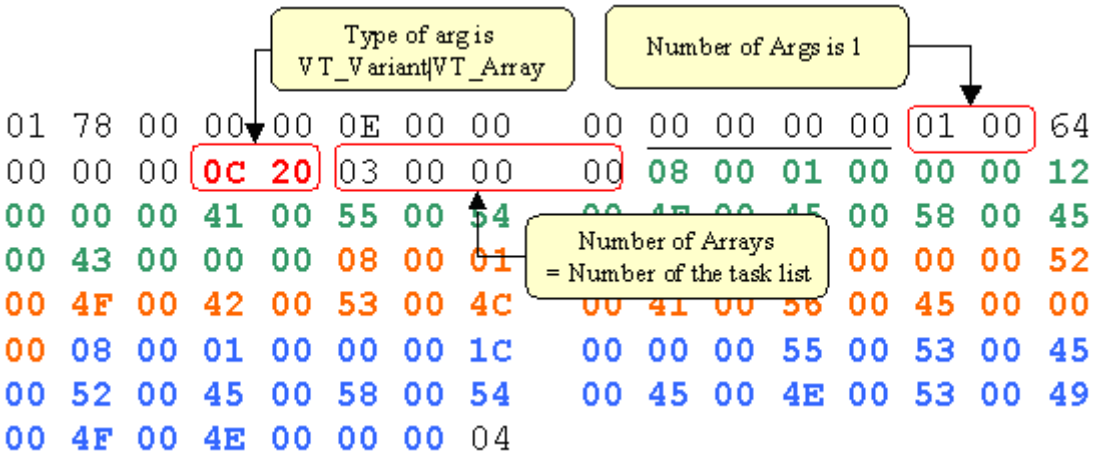
注 2:コントローラ内の PAC プログラムが多いと, 受信パケットの容量が大きくなります. クライアントプログラムの中に十分な受信バッファを用意してください.

通信サンプル 1				
PAC プログラムの一覧を返します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		

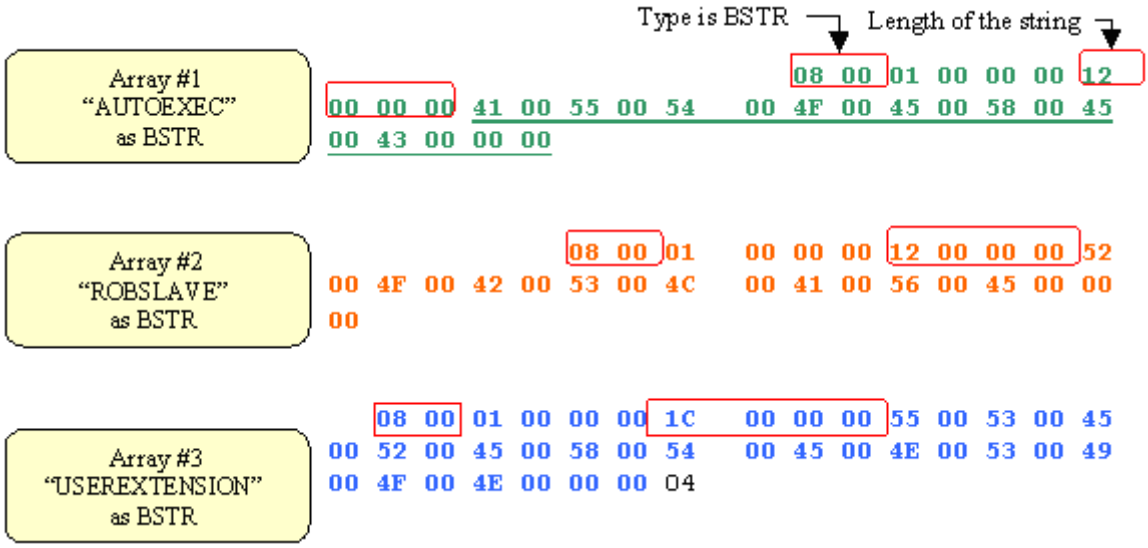
<table>
<tr><td rowspan="4"></td><td>hController</td><td>コントローラハンドル
(Controller_Connect 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00　00 01 00 00 00
0A</td></tr>
<tr><td>bstrOption</td><td>オプション文字列</td><td>VT_BSTR</td><td>空文字列</td></tr>
<tr><td colspan="3">00 08 00 01 00 00 00 00　00 00 00
0A 00 00</td></tr>
<tr><td>受信
パケット</td><td colspan="4">サーバ→クライアント:
01 78 00 00 00 0E 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>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>TaskNames</td><td>PAC プログラムの一覧</td><td>VT_ARRAY |
VT_VARIANT</td><td></td></tr>
<tr><td colspan="3">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>

PAC プログラムの一覧を VT_VARIANT|VT_ARRAY 型で返します.

受信パケットの構造は以下のとおり.



この Variant の引数の構造は以下の通り.



6.3. ロボットオブジェクト

6.3.1. Controller_GetRobot

関数	HRESULT Controller_GetRobot ()			
関数 ID	7			
引数	[in]	long	hController	コントローラハンドル
	[in]	BSTR	bstrName	変数名
	[in]	BSTR	bstrOption	オプション文字列
	[out]	long	hRobot	ロボットハンドル
戻り値	表 4-5 既定のリターンコード一覧 参照			
説明	ロボットオブジェクトのハンドル(hRobot)を取得します. このハンドルを使用して, アプリケーションソフトがロボットのさまざまな機能にアクセスします.			
参照項目	Controller_Connect			

通信サンプル 1				
ロボットハンドル(hRobot)を返します.				
送信パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		

	hController	コントローラハンドル (Controller_Connect 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrName	ロボット名（未使用）	VT_BSTR	空文字列
		00 08 00 01 00 00 00 00 00 00 00 0A 00 00		
	bstrOption	bstrOption	VT_BSTR	空文字列
		00 01 00 00 00 00 00 00 00 0A 00 00 00 08		
受信 パケット	サーバ→クライアント: 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			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル	VT_I4	0x000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		

6.3.2. Robot_Release

関数　HRESULT Robot_Release ()

関数 ID　84

引数　[in]　long　hRobot　ロボットハンドル

戻り値　表 4-5 既定のリターンコード一覧　参照

説明　“hRobot”で指定したロボットオブジェクトを解放します.

6.3.3. Robot_GetVariable

関数　HRESULT Robot_GetVariable ()

関数 ID　62

引数　[in]　long　hRobot　ロボットハンドル

[in]　BSTR　bstrName　変数名

[in]　BSTR　bstrOption　オプション文字列

[out]　long　hVariable　変数ハンドル

戻り値　表 4-5 既定のリターンコード一覧　参照

説明　ロボットのシステム変数オブジェクトのハンドル（hVariable）を取得します.

このハンドルを使用して，アプリケーションソフトが変数にアクセスします.

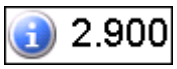
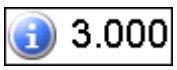
参照項目　Variable_GetValue

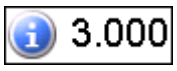
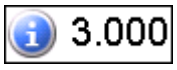
Variable_PutValue

Controller_GetVariable

Task_GetVariable

“bstrName”に下記の変数名を入れることにより，ロボット情報を取得することができます.

変数名	データ型	説明	属性	
			取得	設定
@CURRENT_POSITION	VT_ARRAY | VT_R4	ロボットの現在位置 (X, Y, Z, Rx, Ry, Rz, Fig)	∨	-
@CURRENT_ANGLE	VT_ARRAY | VT_R4	ロボットの現在角度 （各軸の角度）	∨	-
@CURRENT_TRANS	VT_ARRAY | VT_R4	ロボットの現在位置（T 型表記） （X,Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig）	∨	-
@CURRENT_TOOL	VT_I2	使用中のツール番号	∨	-
@CURRENT_WORK	VT_I2	使用中のワーク番号	∨	-
@SPEED	VT_R4	内部速度	∨	-
@ACCEL	VT_R4	内部加速度	∨	-
@DECEL	VT_R4	内部減速度	∨	-
@EXTSPEED	VT_R4	外部速度	∨	-
@EXTACCEL	VT_R4	外部加速度	∨	-
@EXTDECEL	VT_R4	外部減速度	∨	-
@SERVO_ON	VT_I2	サーボの状態 0=OFF,1= ON	∨	-
@TYPE 	VT_I4	ロボットタイプデータ	∨	-
@HIGH_CURRENT_POSITION 	VT_ARRAY | VT_R4	ロボットの現在位置(P 型) + タイムスタンプ (X, Y, Z, Rx, Ry, Rz, Fig, TimeStamp) 動作仕様: エンコーダ値を返します. (分解能:500 μsec)	∨	-

		ただし，マシンロック時はコントローラ内部の指令値を返します．(分解能:8msec) ※ マシンロック状態では現在位置が不定になります． ※ タイムスタンプは 23bit の精度を持ちます．したがって，0x7FFFFF を超えると 0 にリセットされます．		
@HIGH_CURRENT_ANGLE 	VT_ARRAY | VT_R4	ロボットの現在角度(J 型) + タイムスタンプ（各軸の角度, TimeStamp） 動作仕様に関しては@HIGH_CURRENT_POSITION を参照してください．	∨	-
@HIGH_CURRENT_TRANS 	VT_ARRAY | VT_R4	ロボットの現在位置（T 型）+ タイムスタンプ（X,Y, Z, Ox, Oy, Oz, Ax, Ay, Az, Fig, TimeStamp） 動作仕様に関しては@HIGH_CURRENT_POSITION を参照してください．	∨	-

通信サンプル 1				
変数ハンドル（hVariable）を返します．				
送信パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrName	変数名	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	オプション	VT_BSTR	空文字列

<table>
<tr><td></td><td></td><td colspan="3">0A 00 00
00 08 00 01 00 00 00 00 00 00 00</td></tr>
<tr><td rowspan="5">受信
パケット</td><td colspan="4">サーバ→クライアント:
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 30 00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>hVariable</td><td>変数ハンドル</td><td>VT_I4</td><td>0x300000</td></tr>
<tr><td colspan="3">0A
00 00 00 03 00 01 00 00 00 00 00 30 00</td></tr>
</table>

6.3.4. Robot_Move

関数　HRESULT Robot_Move ()

関数 ID　72

引数

[in]	long	hRobot	ロボットハンドル
[in]	long	lComp	補間 1:MOVE P 2:MOVE L
[in]	VARIANT	vntPose	ポーズデータ(BSTR のみ)
[in]	BSTR	bstrOpt	動作オプション

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　ロボットオブジェクトのハンドル(hRobot)で指定したロボットが，指定した座標まで動きます.

このメソッドは PAC 言語の MOVE 命令に対応します.

注 1:この関数を使用するためには，RobSlave.pac を実行させておく必要があります.

ポーズデータ型で文字列を指定した場合の書式と意味は以下のとおり.

VT_BSTR の場合

・ 補完指定 = 1, 2 の場合

"[<@パス開始変位>] <ポーズ>"

ex. "P1", "@P T100", "@E J520"

<ポーズ> : “<変数型><番号>” または “[<変数型>](<要素 1>,<要素 2>,…)”

<変数型> : ’P’,’T’,’J’のいずれか一文字

※要素(即値)指定で変数型が省略された場合は’P’指定扱いになります.

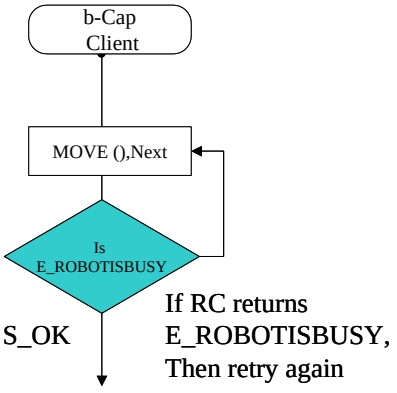
<番号> : 10 進数で表現される数値
<要素 n> : 各変数型(P,J,T)の要素
P 型＝P(<x>,<y>,<z>,<rx>,<ry>,<rz>,<fig>)
J 型＝J(<j1>,<j2>,<j3>,<j4>,<j5>,<j6>,<j7>,<j8>)
T 型＝T(<x>,<y>,<z>,<ox>,<oy>,<oz>,<ax>,<ay>,<az>,<fig>)
※4 軸ﾛﾎﾞｯﾄの場合 P 型の T 要素は<rz>に対応します. <rx>,<ry>は未使用

<@パス開始変位> : “@0” , “@P” , “@E” , “@<数値>”のいずれか

例 1. Move 1, “@P P1", "NEXT" ‘ MOVE P, @P P1, NEXT
例 2. Move 2, “P1” ‘ MOVE L, P1
例 3. Move 1, “@0 J1” ‘ MOVE P, @0 J1
例 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)
例 5. Move 1, “@P J(60,50,40,30,20,10)” ‘ MOVE L,@P J(60,50,40,30,20,10)

“NEXT”オプションをつけると, ロボットは, 動作が完了するのを待たずに次の非動作命令を実行します.

Move メソッドを二つ以上連続して実行した場合, 一つ前の動作命令が完了していない内は次の動作命令は“E_ROBOTISBUSY” (0x80010004)の状態になります. アプリケーションが Move メソッドを連続して実行する必要がある場合は, メソッドが“S_OK”を返すまで再試行して下さい.



Move メソッドで対応している PAC MOVE コマンド一覧を以下に示します.

表 6-3 Move コマンド一覧

区分	PAC コマンド	Move メソッド
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 >: 整数 0～65535（＝変数番号）

<table>
<tr><th colspan="4">通信サンプル 1 - MOVE 1, “P12”</th></tr>
<tr><td colspan="4">動作コマンド“MOVE 1, “P12””を実行します.</td></tr>
<tr><td rowspan="8">送信
パケット</td><td colspan="3">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</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>lComp</td><td>補間(MOVE P(= PTP))</td><td>VT_I4</td><td>1</td></tr>
<tr><td></td><td colspan="3">00 03 00 01 00 00 00 01 00 00 00 0A 00 00</td></tr>
<tr><td>vntPose</td><td>位置</td><td>VT_BSTR</td><td>“P12”</td></tr>
</table>

		10 00 00 00 08 00 01 00 00 00 06 00 00 00 50 00 31 00 32 00		
	bstrOpt	オプション	VT_BSTR	空文字列
		0A 00 00 00 08 00 01 00 00 00 00 00 00 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 39 0A 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

通信サンプル 2 - MOVE 2, “@P P12”, “NEXT”				
動作コマンド“MOVE 2, “@P P12”, “NEXT””を実行します				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		0A 00 00 00 03 00 01 00 00 00 01 00 00 00		
	lComp	補間(MOVE L(= CP))	VT_I4	2
		0A 00 00 00 03 00 01 00 00 00 02 00 00 00		
	vntPose	位置	VT_BSTR	“@P P12”
		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		
	bstrOpt	オプション	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		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 38 0A 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値

		バイナリ		
	なし	-	-	-
		-		

通信サンプル 3 - MOVE 2,“@P P(544.2,-79.2,136.6,0,0,3.9,0)”, “NEXT”			
動作コマンド“MOVE 2, “@P P(X,Y,Z,,,Fig)”, “NEXT””を実行する.			
送信パケット	クライアント→サーバ: 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 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		
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	lComp	補間（MOVE L(= CP))	VT_I4	2
		00 03 00 01 00 00 00 02 00 00 00 0A 00 00		
	vntPose	位置	VT_BSTR	“@P P(544.2,-79.2,136.6,0,0,3.9,0)”
		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 4C 00 00 00 08		
	bstrOpt	オプション	VT_BSTR	“NEXT”
		00 01 00 00 00 08 00 00 00 4E 00 45 00 58 00 54 00 12 00 00 00 08		
受信パケット	サーバ→クライアント: 01 10 00 00 00 48 0A 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

6.3.5. Robot_Execute

関数	HRESULT Robot_Execute()			
関数 ID	64			
引数	[in]	long	hRobot	ロボットハンドル
	[in]	BSTR	bstrCommand	コマンド名
	[in]	VARIANT	vntParam	パラメータ
	[out]	VARIANT	pVal	結果値
戻り値	表 4-5 既定のリターンコード一覧 参照			
説明	ロボット(hRobot)のコマンドを実行します.			

“bstrCommand”に下記のコマンド名を入れることにより, コマンドを実行することができます.

表 6-4 コマンド実装一覧

コマンド	パラメータ	戻り値	動作
Speed	VT_R4: Internal speed (0.1 to 100.0)	なし	内部速度を設定します. 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
ExtSpeed	VT_R4|ARRAY: (<ExtSpeed >,< ExtAccel >,< ExtDecel >) <ExtSpeed>=VT_R4:外部速度 (0.1 to 100.0) <ExtAccel>=VT_R4:外部加速度 (0.0001 to 100.0) <ExtDecel>=VT_R4:外部減速度 (0.0001 to 100.0)	なし	外部速度, 外部加速度, 外部減速度を設定します. 外部加速度, 外部減速度はオプションとして、それぞれ配列の２番目、３番目に設定します.
Motor	VT_I2: State (1:ON / 0:OFF)	なし	モータ ON/OFF “モータ ON/OFF” には時間がかかるので注意

			してください. 3 秒ほどかかることがあります.
DefTool	VT_R4|ARRAY: (<ToolNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ>)	なし	ツール座標系の設定を行います. 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
Defwork	VT_R4|ARRAY: (<WorkNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ>)	なし	ワーク座標系の設定を行います. 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
DefArea	VT_R4|ARRAY: (<AreaNo>,<X>,<Y>,<Z>, <RX>,<RY>,<RZ>, <VX>,<VY>,<VZ>, <IONo>,<P-Variable No>, <ErrorOut>,<Enable/Disable >)	なし	エリアの設定と有効無効を切り替えます. 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
MotionComp	なし	VT_I2: Mode(0:動作中/1:動作完了)	MotionComp 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
MotionSkip	なし	なし	MotionSkip 注:この関数を使用する

			ため には, RobSlave.pac を実行させておく必要があります.
Interrupt	VT_I2: 動作(0:OFF / 1:ON)	なし	INTERRUPT ON/OFF 注:この関数を使用するため には, RobSlave.pac を実行させておく必要があります.
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, [@pass] approach distance [,option =“NEXT”] 例.) “1,P1,@P 100,NEXT” 詳細は 6.3.5.1 を参照下さい.	なし	APPROACH <Interpolation method>, <pose variable type><pose variable number>, <pass> <approach distance>[, NEXT] 注:この関数を使用するため には, RobSlave.pac を実行させておく必要があります.

[3] コントローラバージョン 2.801 以前ではこのサンプルと同様にオプションパラメータのデータ型として VT_VARIANT(VT_R4|VT_ARRAY)もしくは VT_VARIANT(VT_BSTR)を指定してください. このデータ型での指定は, 旧バージョンとの互換性維持のために残されています. したがって, コントローラバージョン 2.802 以降のコントローラでは, オプションパラメータのデータ型として VT_BSTR もしくは VT_R4|VT_ARRAY を指定することをお勧めします.

Depart	VT_BSTR: Interpolation method(=1:P, 2:L), [@pass] depart distance , [,option =“NEXT”] 例) “1,@P 100,NEXT” 詳細は 6.3.5.2 を参照下さい.	なし	DEPART <Interpolation method >, <pass> <depart distance>[, NEXT] 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
DriveAEx	VT_BSTR: [@pass] (<Axis No.1>,<value>), [(<Axis No.1>,<value>)], … [(<Axis No.8>,<value>)] [,option :VT_BSTR “NEXT”] 例) “@P (1,20),(2,30)”	なし	DRIVEA <Pass start displacement> (<Axis1 number>, <Axis1 Coordinate>), (<Axis2 number>, <Axis2 Coordinate>),…(<Axis8 number>, <Axis8 Coordinate>) [,NEXT] 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
DriveEx	VT_BSTR [@pass] (<Axis No.1>,<value>), [(<Axis No.1>,<value>)], … [(<Axis No.8>,<value>)] [,option :VT_BSTR “NEXT”] 例) “@P (1,20),(2,30)”	なし	DRIVE <Pass start displacement> (<Axis1 number>, <Axis1 Coordinate>), (<Axis2 number>, <Axis2 Coordinate>),…(<Axis8 number>, <Axis8 Coordinate>) [,NEXT] 注:この関数を使用するためには, RobSlave.pac を実行さ

			せておく必要があります.
UserExt	VT_R4|ARRAY: CommandNumber, Parameter1,Parameter2,Parameter3, ・・・,Parameter9 Or VT_I4: CommandNumber Or VT_VARIANT|ARRAY: CommandNumber(VT_I4), Parameter1,Parameter2,Parameter3, …,Parameter9(VT_R4|ARRAY) 詳細は 6.3.5.3 を参照下さい	VT_R4 | VT_ARRAY	UserExtension 注:この関数を使用するためには，RobSlave.pac を実行させておく必要があります.
slvChangeMode 3.000	<SlaveMode: VT_I2 or VT_I4> = 0x0： スレーブモード停止 0x1： 同期型(P 型) 0x2： 同期型(J 型) 0x3： 同期型(T 型) 0x101 :非同期型(P 型) 0x102： 非同期型(J 型) 0x103： 非同期型(T 型)	なし	スレーブモードを変更します. スレーブモードに変更するとき，ロボットが完全停止するまで待ちます. (b-CAP スレーブ機能)
slvMove 3.000	<P/J/T 型:VT_R4|ARRAY> Or <P/J/T 型:VT_R8|ARRAY>	<現在位置(J 型): VT_R4|ARRAY>	SlaveMove を実行します. 引数に指定する P/J/T 型は、スレーブモードで設定した型になります. (b-CAP スレーブ機能)
slvGetMode 3.000	なし	<SlaveMode:VT_I2> = 0x0： スレーブモード停止 0x1： 同期型(P 型) 0x2： 同期型(J 型) 0x3： 同期型(T 型) 0x101 :非同期型(P 型) 0x102： 非同期型(J 型) 0x103： 非同期型(T 型)	現在のスレーブモードを取得します. (b-CAP スレーブ機能)
GetSrvData Reserved	<データ番号:VT_I4>	<サーボ内部データ: VT_R4|ARRAY>	データ番号で指定したロボット軸のサーボ内部データを取得します.
SetSrvData Reserved	<VT_VARIANT|ARRAY>: (<データ番号:VT_I4>, <サーボ内部データ :VT_R4|TV_ARRAY>)	なし	データ番号で指定したロボット軸のサーボ内部データを設定します.

<table>
<tr><th colspan="4">通信サンプル 1 – SPEED 50.0</th></tr>
<tr><td colspan="4">ロボットの内部速度を変更します.</td></tr>
<tr><td rowspan="8">送信
パケット</td><td colspan="3">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 01 00 00 00</td></tr>
<tr><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“SPEED”</td></tr>
<tr><td></td><td colspan="3">14 00 00
00 08 00 01 00 00 00 0A 00 00 00 53 00 50 00 45
00 45 00 44 00</td></tr>
<tr><td>vntParam</td><td>コマンドパラメータ</td><td>VT_R4</td><td>50.0</td></tr>
<tr><td></td><td colspan="3">0A 00 00 00 04 00 01 00 00 00 00
00 48 42</td></tr>
<tr><td rowspan="5">受信
パケット</td><td colspan="3">サーバ→クライアント:
01 10 00 00 00 5C 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="3">-</td></tr>
</table>

<table>
<tr><th colspan="4">通信サンプル 2 - EXTSPEED 50.0</th></tr>
<tr><td colspan="4">外部速度, 外部加速度, および外部減速度を変更します.</td></tr>
<tr><td rowspan="4">送信
パケット</td><td colspan="3">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
</table>

	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	コマンド文字列	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	コマンドパラメータ	VT_R4	50.0
		00 00 00 00 04 00 01 00 00 00 00 00 48 42 0A 00 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 8A 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

通信サンプル 3 - EXTSPEED 50.0,10.0, 3.0				
“EXTSPEED”コマンドを使用すると速度, 加速度, 減速度が個別に設定できます.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00 00 01 00 00 00 0A		
	bstrCommand	コマンド文字列	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	コマンドパラメータ	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		

受信 パケット	サーバ→クライアント: 01 10 00 00 00 8A 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

通信サンプル 4 – Motor ON/OFF				
モータ電源を ON/OFF します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001
		00 00 00 03 00 01 00 00　00 01 00 00 00		0A
	bstrCommand	コマンド文字列	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	コマンドパラメータ	VT_I2	0x0001
		08 00 00　00 02 00 01 00 00 00 01 00		
受信 パケット	サーバ→クライアント: 01 10 00 00 00 8B 00 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

通信サンプル 5 – MotionComp
MotionComp の値を取得する場合

<table>
<tr><td>送信
パケット</td><td colspan="4">クライアント→サーバ:
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</td></tr>
<tr><td rowspan="8"></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 01 00 00 00</td></tr>
<tr><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“MOTIONCOMP”</td></tr>
<tr><td></td><td colspan="3">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</td></tr>
<tr><td>vntParam</td><td>オプションパラメータ</td><td>VT_NULL</td><td></td></tr>
<tr><td></td><td colspan="3">06
00 00 00 01 00 01 00 00 00</td></tr>
<tr><td>受信
パケット</td><td colspan="4">サーバ→クライアント:
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</td></tr>
<tr><td rowspan="4"></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>pVal</td><td>Mode</td><td>VT_I2</td><td>1</td></tr>
<tr><td></td><td colspan="3">08
00 00 00 02 00 01 00 00 00 01 00</td></tr>
</table>

通信サンプル 6-a – P2J
このサンプルは P2J を実行します. J2P,J2T,P2J,P2T,T2J,T2P,TINV,NORMTRN は同様の方法で実行できます. バージョン 2.802 以降のコントローラではこのサンプルと同様に, オプションパラメータのデータ型として VT_BSTR もしくは VT_R4|VT_ARRAY を指定してください.

<table>
<tr><td>送信
パケット</td><td colspan="4">クライアント→サーバ:
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</td></tr>
<tr><td rowspan="3"></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
</table>

<table>
<tr><td rowspan="6"></td><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</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 0A</td></tr>
<tr><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“P2J”</td></tr>
<tr><td colspan="4">00 08 00 01 00 00 00 06 00 00 00 50 00 32 00 4A 00 10 00 00</td></tr>
<tr><td>vntParam</td><td>オプションパラメータ</td><td>VT_R4|VT_ARRAY
注記:
VT_BSTR を用いて”P1”のように変数指定も可能です.</td><td>673.3548
257.9997
300
0
0
83.35404
0</td></tr>
<tr><td colspan="4">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</td></tr>
<tr><td>受信
パケット</td><td colspan="4">サーバ→クライアント:
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 00 90 C1 38 29 82
42 00 00 96 43 1A 18 11 42 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="4">バイナリ</td></tr>
<tr><td rowspan="2">pVal</td><td>結果</td><td>VT_R4|ARRAY</td><td>-18
65.08051
300
36.27354</td></tr>
<tr><td colspan="4">00 00 00 04 20 04 00 00 00 00 00 00 90 C1 38 29 82 42 00 00 96 43 1A 18 11 42 16</td></tr>
</table>

<table>
<tr><td colspan="4">通信サンプル 6-b – P2J</td></tr>
<tr><td colspan="4">このサンプルは P2J を実行します.
J2P,J2T,P2J,P2T,T2J,T2P,TINV,NORMTRN は同様の方法で実行できます.
注記:
コントローラバージョン 2.801 以前ではこのサンプルと同様にオプションパラメータのデータ型として VT_VARIANT(VT_R4|VT_ARRAY)もしくは VT_VARIANT(VT_BSTR)を指定してください. このデータ型での指定は, 旧バージョンとの互換性維持のために残されています. したがって, コントローラバージョン 2.802 以降のコントローラでは, オプションパラメータのデータ型として VT_BSTR もしくは VT_R4|VT_ARRAY を指定することをお勧めします.</td></tr>
<tr><td>送信
パケット</td><td colspan="3">クライアント→サーバ:
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</td></tr>
<tr><td></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td></td><td colspan="3">バイナリ</td></tr>
<tr><td></td><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td></td><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 01 00 00 00 0A</td></tr>
<tr><td></td><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“P2J”</td></tr>
<tr><td></td><td></td><td colspan="3">00 08 00 01 00 00 00 06 00 00 00 50 00 32 00 4A
00 10 00 00</td></tr>
<tr><td></td><td>vntParam</td><td>オプションパラメータ</td><td>VT_VARIANT
(VT_R4|ARRAY)
注記:
VT_VARIANT
(VT_BSTR) を用い”P1”のように変数指定も可能です.</td><td>421.0982,
266.2033
798.9265
85.9726
34.23356
5</td></tr>
<tr><td></td><td></td><td colspan="3">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</td></tr>
</table>

受信パケット

サーバ→クライアント:

```
01 32 00 00 00 0E 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
```

引数	説明	データ型	値
	バイナリ		
pVal	結果	VT_R4|ARRAY	29.99896 29.99898 29.99889 29.99889 29.999 29.99669

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
```

Communication sample 7 – DriveEX

このサンプルは “DriveAEx” を実行します. :

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

“DriveEx “も同様の手法で実行できます.

送信パケット

クライアント→サーバ

```
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
```

引数	説明	データ型	値
	バイナリ		
hRobot	ロボットハンドル (Controller_GetRobot 参照)	VT_I4	0x0000001

0A

```
00 00 00 03 00 01 00 00   00 01 00 00 00
```

bstrCommand	コマンド文字列	VT_BSTR	“DriveAEx”

<table>
<tr><td rowspan="6"></td><td></td><td colspan="3">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</td></tr>
<tr><td rowspan="3">bstrPosition</td><td>動作位置</td><td>VT_BSTR</td><td>“@P (1,10),(2,10),(3,10),
(4,10),(5,10),(6,10)”</td></tr>
<tr><td colspan="3">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</td></tr>
<tr><td rowspan="2">bstrOption</td><td>オプション</td><td>VT_BSTR</td><td>“NEXT”</td></tr>
<tr><td colspan="3">12 00 00 00 08 00 01 00 00 00 08 00 00 00 4E
00 45 00 58 00 54 00</td></tr>
<tr><td></td><td colspan="3"></td></tr>
<tr><td rowspan="4">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 10 00 00 00 04 00 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td rowspan="2">なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">–</td></tr>
</table>

<table>
<tr><th colspan="4">Communication sample 8 – slvMove</th></tr>
<tr><td colspan="4">このサンプルは “slvMove” を実行します. :
SlvMove 12.6334, 4.920117, 233.132, 4.918625, 0, 0, 0, 0</td></tr>
<tr><td>送信
パケット</td><td colspan="3">クライアント→サーバ
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</td></tr>
<tr><td></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td></td><td>バイナリ</td><td colspan="2"></td></tr>
<tr><td></td><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td></td><td></td><td colspan="3">00 00 00 03 00 01 00 00 00 01 00 00 00 0A</td></tr>
<tr><td></td><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“slvMove”</td></tr>
<tr><td></td><td></td><td colspan="3">00 08 00 01 00 00 00 0E 00 00 00 73 00 6C 00 76 00 00 00 18
00 4D 00 6F 00 76 00 65 00</td></tr>
<tr><td></td><td>vntParam</td><td>オプションパラメータ</td><td>VT_R4 |
ARRAY</td><td>12.6334, 4.920117, 233.132,
4.918625, 0, 0, 0, 0</td></tr>
<tr><td></td><td></td><td colspan="3">00 00 00 66 22 4A 41 9A 71 9D 40 C8 21 69 43 61 00 00 00 26
65 9D 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00</td></tr>
<tr><td>受信
パケット</td><td colspan="3">サーバ→クライアント:
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</td></tr>
<tr><td></td><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td></td><td>バイナリ</td><td colspan="2"></td></tr>
<tr><td></td><td>pVal</td><td>現在位置</td><td>VT_R4 |
ARRAY</td><td>9.9993162, 4.9992185,
230.00119, 4.9984827,
0, 0, 0, 0</td></tr>
<tr><td></td><td></td><td colspan="3">00 00 00 04 20 08 00 00 00 33 FD 1F 41 99 F9 9F 00 00 00 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</td></tr>
</table>

6.3.5.1. Approach

Approach 命令は Robot_Execute()を利用する事で実行できます.
パラメータは以下のようになっています.

Argument	[in]	long	hRobot	ロボットハンドル番号
	[in]	BSTR	bstrCommand	コマンド文字列 “Approach”
	[in]	BSTR	vntParam	Approach パラメータ文字列(VT_BSTR)

書式と意味は以下のようになっています.

VntParam の型 : VT_BSTR

<補完指定 1 or 2>,<基準位置>,<[@パス開始変位] アプローチ距離>, [NEXT]

例). "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

<基準位置> : “<変数型><変数番号>” または “(X, Y, Z, RX, RY, RZ, Fig)”
: <変数型> : ’P’,’T’,’J’のいずれか一文字
※要素(即値)指定で変数型が省略された場合は’P’指定扱いになります.
<変数番号> : 10 進数で表現される数値
<要素 n> : 各変数型(P,J,T)の要素
P 型＝P(<x>,<y>,<z>,<rx>,<ry>,<rz>,<fig>)
J 型＝J(<j1>,<j2>,<j3>,<j4>,<j5>,<j6>,<j7>,<j8>)
T 型＝T(<x>,<y>,<z>,<ox>,<oy>,<oz>,<ax>,<ay>,<az>,<fig>)
※4 軸ﾛﾎﾞｯﾄの場合 P 型の T 要素は<rz>に対応します. <rx>,<ry>は未使用
<@パス開始変位> : “@0” , “@P” , “@E”

“NEXT”オプションをつけると, ロボットは, 動作が完了するのを待たずに次の非動作命令を実行します.
RobSlave.PAC を使用する命令(主に動作命令)を二つ以上連続して実行した場合, 一つ前の動作命令が完了していない内は次の動作命令は“E_ROBOTISBUSY” (0x80010004)の状態になります. アプリケーションが Move メソッドを連続して実行する必要がある場合は, メソッドが“S_OK”を返すまで再試行して下さい.

6.3.5.2. Depart

Depart 命令もまた Robot_Execute()を利用する事で実行できます.

そのパラメータは以下のようになっています.

Argument	[in]	long	hRobot	ロボットハンドル番号
	[in]	BSTR	bstrCommand	コマンド文字列 “Depart”
	[in]	BSTR	vntParam	Depart パラメータ文字列（VT_BSTR）

書式と意味は以下のようになっています.

vntParam の型 : VT_BSTR

<補完指定 1 or 2>,<[@パス開始変位] デパート距離>, [NEXT]

例) "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

<@パス開始変位>= "@0", "@P", "@E"

“NEXT”オプションをつけると, ロボットは, 動作が完了するのを待たずに次の非動作命令を実行します.

RobSlave.PAC を使用する命令（主に動作命令）を二つ以上連続して実行した場合, 一つ前の動作命令が完了していない内は次の動作命令は“E_ROBOTISBUSY” (0x80010004)の状態になります. アプリケーションが Move メソッドを連続して実行する必要がある場合は, メソッドが“S_OK”を返すまで再試行して下さい.

6.3.5.3. UserExtension

UserExtention.pac にユーザ拡張プログラムを追加しそれをユーザ定義のコマンドとして実行することができます.

<table>
<tr><td colspan="5">通信サンプル – UserExtension を用いた SYSSTATE 命令の実行.</td></tr>
<tr><td colspan="5">このサンプルは UserExt コマンド（UserExtension.PAC でのコマンド番号 10055 (= 0x2747)）の例を示しています.</td></tr>
<tr><td rowspan="3">送信
パケット</td><td colspan="4">クライアント→サーバ
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="4">バイナリ</td></tr>
</table>

<table>
<tr><td rowspan="6"></td><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot を参照してください)</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 0A</td></tr>
<tr><td>bstrCommand</td><td>コマンド文字列</td><td>VT_BSTR</td><td>“UserExt”</td></tr>
<tr><td colspan="4">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 18 00 00</td></tr>
<tr><td>VntParam</td><td>コマンド番号</td><td>VT_I4</td><td>10055 (= 0x2747)</td></tr>
<tr><td colspan="4">00 00 00 47 27 00 00　0A 00 00 00 03 00 01</td></tr>
<tr><td>受信
パケット</td><td colspan="4">サーバ→クライアント:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="4">バイナリ</td></tr>
<tr><td>pVal</td><td>結果</td><td>ARRAY | R4</td><td>2073
0
0
0
0
0
0
0
-1</td></tr>
<tr><td colspan="4">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 2E</td></tr>
</table>

6.3.6. Robot_Change

関数　HRESULT Robot_Change()

関数 ID　66

引数				
	[in]	long	hRobot	ロボットハンドル
	[in]	BSTR	bstrName	ツールまたはユーザ座標系

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　　ロボット(hRobot)のツール座標系またはユーザ座標系を変更します.
注 1:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.

“bstrName”に下記のコマンド名を入れることにより, コマンドを実行することができます.

表 6-5　コマンド実装一覧

名前	動作
Tooln	ツール座標系選択 このコマンドではツールの数量が bstrName に含まれています.(この場合, n=0～63) 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.
Workn	ユーザ座標系選択 このコマンドではツールの数量が bstrName に含まれています.(この場合,n=0 to 7) 注:この関数を使用するためには, RobSlave.pac を実行させておく必要があります.

<table>
<tr><td colspan="5">通信サンプル 1 –Tool 7 を選択</td></tr>
<tr><td colspan="5">ツール番号 7 を選択します.</td></tr>
<tr><td rowspan="6">送信
パケット</td><td colspan="4">クライアント→サーバ:
01 38 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 0C　00 00 00 54 00 4F 00 4F
00 4C 00 3D 00 37 00 04</td></tr>
<tr><td rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>hRobot</td><td>ロボットハンドル
(Controller_GetRobot 参照)</td><td>VT_I4</td><td>0x0000001</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00　00 01 00 00 00 0A</td></tr>
<tr><td rowspan="2">bstrName</td><td>名前文字列</td><td>VT_BSTR</td><td>“TOOL7”</td></tr>
<tr><td colspan="3">00 08 00 01 00 00 00 0C　00 00 00 54 00 4F 00 4F 16 00 00
00 4C 00 3D 00 37 00</td></tr>
<tr><td rowspan="3">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 10 00 00 98 04 00 00　00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="4"></td></tr>
</table>

		バイナリ		
	なし	-	-	-
		-		

6.4. タスクオブジェクト

ロボット内の PAC プログラムを開始/停止させることができます.

6.4.1. Controller_GetTask

タスク名で指定したタスクオブジェクトハンドルを取得します.

関数　HRESULT Controller_GetTask ()

関数 ID　8

引数				
	[in]	long	hController	コントローラハンドル
	[in]	BSTR	bstrName	タスク名（PAC プログラム名）
	[in]	BSTR	bstrOption	オプション文字列（未使用）
	[out]	long	hTask	タスクハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　タスクオブジェクトのハンドル（hTask）を取得します.

参照項目　Controller_connect

通信サンプル 1 – Controller_GetTask(“RobSlave”)				
タスクハンドルを返します.				
送信パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hController	コントローラハンドル (Controller_Connect 参照)	VT_I4	0x0000001
		0A 00 00 00 03 00 01 00 00　00 01 00 00 00		
	bstrName	タスク名	VT_BSTR	“ROBSLAVE”
		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		
	bstrOption	bstrOption	VT_BSTR	空文字列
		0A 00 00 00 08 00 01 00 00 00 00 00 00　00		

<table>
<tr><td rowspan="5">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 1E 00 00 00 99 04 00　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 rowspan="2">引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td colspan="3">バイナリ</td></tr>
<tr><td>hTask</td><td>タスクハンドル</td><td>VT_I4</td><td>0x0000113b</td></tr>
<tr><td colspan="4">00 00 00 03 00 01 00 00　00 3B 11 00 00　0A</td></tr>
</table>

6.4.2. Task_Release

関数　HRESULT Task_Release ()

関数 ID　99

引数　[in]　long　hTask　タスクハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　“hTask”で指定したタスクオブジェクトを解放します.

6.4.3. Task_GetVariable

関数　HRESULT Task_GetVariable ()

関数 ID　85

引数　[in]　long　hTask　コントローラハンドル

[in]　BSTR　bstrName　変数名

[in]　BSTR　bstrOption　オプション文字列

[out]　long　hVariable　変数ハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　タスクのシステム変数オブジェクトのハンドル(hVariable)を取得します.

参照項目　Variable_GetValue

Variable_PutValue

Controller_GetVariable

Robot_GetVariable

以下の変数名を使用して, タスクに関する情報を取得します.

変数名	データ型	説明	属性	
			取得	設定
@STATUS	VT_I2	タスクの状態 1: DORMANT 2: READY	∨	-

		3: RUN 4: WAIT 6: SUSPEND 0: NON_EXISTENT		
@PRIORITY	VT_I2	タスクの優先度	∨	-
@LINE_NO	VT_I2	現在実行中のメインプログラムの行番号	∨	-
@CYCLE_TIME	VT_I4	タスクの 1 サイクルの実行時間	∨	-

通信サンプル 1 – Task_GetVariable (“@STATUS”)

変数ハンドルを返します.

送信パケット

クライアント→サーバ:

```
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
```

引数	説明	データ型	値
	バイナリ		
hTask	タスクハンドル (Controller_Connect 参照)	VT_I4	0x0000113b
	00 00 00 03 00 01 00 00 00 3B 11 00 00 0A		
bstrName	変数名	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	空文字列
	0A 00 00 00 08 00 01 00 00 00 00 00 00 00		

受信パケット

サーバ→クライアント:

```
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
```

引数	説明	データ型	値
	バイナリ		
hVariable	タスクハンドル	VT_I4	0x0040113b
	00 00 00 03 00 01 00 00 00 3B 11 40 00 01 00 0A		

6.4.4. Task_Start

関数	HRESULT Task_Start ()			
関数 ID	88			
引数	[in]	long	hTask	タスクハンドル
	[in]	long	lMode	開始モード 1: 1 Cycle execution 2: Cyclic execution 3: 1 Step forward ~~4: 1 Step backword,~~ 5: Resume all
	[in]	BSTR	bstrOpt	オプション文字列（未使用）
戻り値	表 4-5 既定のリターンコード一覧 参照			
説明	タスク（hTask）に対応する PAC プログラムを開始します. If this method is called with mode 5 (Resume), then all suspended programs in the robot controller are resumed.			
参照項目	Controller_GetTask Task_Stop			

<table>
<tr><th colspan="4">通信サンプル 1 – Task_Start - Cyclic execution</th></tr>
<tr><td colspan="4">タスクを開始します.</td></tr>
<tr><td rowspan="7">送信
パケット</td><td colspan="3">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td rowspan="2">hTask</td><td>タスクハンドル
(Controller_GetTask 参照)</td><td>VT_I4</td><td>0x0000113b</td></tr>
<tr><td colspan="3">00 00 00 03 00 01 00 00 00 3B 11 00 00 0A</td></tr>
<tr><td rowspan="2">lMode</td><td>lMode(2: Cyclic execution)</td><td>VT_I4</td><td>2</td></tr>
<tr><td colspan="3">00 03 00 01 00 00 00 02 00 00 00 0A 00 00</td></tr>
<tr><td rowspan="2">bstrOpt</td><td>bstrOption</td><td>VT_BSTR</td><td>Null string</td></tr>
<tr><td colspan="3">00 01 00 00 00 00 00 00 00 0A 00 00 00 08</td></tr>
</table>

受信 パケット	サーバ→クライアント: 01 10 00 00 00 A1 04 00　00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

6.4.5. Task_Stop

関数　HRESULT Task_stop

関数 ID　89

引数

[in]	long	hTask	タスクハンドル
[in]	long	lMode	停止モード 1: 一時停止(Suspend) 2: ステップ停止 3: サイクル停止 4: リセット 5: Suspend all (Continuous stop)
[in]	BSTR	bstrOpt	オプション文字列(未使用)

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　タスク(hTask)に対応するプログラムを停止します.

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

参照項目　Controller_GetTask

Task_Start

通信サンプル 1 – Task_Stop - Cycle stop				
タスクの停止				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hTask	タスクハンドル (Controller_GetTask 参照)	VT_I4	0x0000113b

<table>
<tr><td rowspan="6"></td><td></td><td colspan="3">0A
00 00 00 03 00 01 00 00 00 01 00 00 00</td></tr>
<tr><td rowspan="2">lMode</td><td>lMode(3: Cycle stop)</td><td>VT_I4</td><td>3</td></tr>
<tr><td colspan="3">0A 00 00
00 03 00 01 00 00 00 03 00 00 00</td></tr>
<tr><td rowspan="2">bstrOpt</td><td>bstrOption</td><td>VT_BSTR</td><td>空文字列</td></tr>
<tr><td colspan="3">0A 00 00 00 08
00 01 00 00 00 00 00 00 00</td></tr>
<tr><td></td><td colspan="4"></td></tr>
<tr><td rowspan="5">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 10 00 00 00 A5 04 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td rowspan="2">なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">-</td></tr>
</table>

6.5. 変数オブジェクト

変数の読み書きができます.

6.5.1. Variable_Release

関数　HRESULT Variable_Release ()

関数 ID　111

引数　[in]　Long　hVar　変数ハンドル

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　“hVar”で指定した変数オブジェクトを解放します.

6.5.2. Variable_GetValue

変数ハンドルで指定した変数の値を取得します.

関数　HRESULT Variable_GetValue ()

関数 ID　101

引数　[in]　long　hVariable　変数ハンドル

[out]　VARIANT　pVal　値

戻り値　表 4-5 既定のリターンコード一覧 参照

説明　“hVariable”で指定した変数オブジェクトの値を取得します.

参照項目　Controller_GetVariable

Robot_GetVariable

Task_GetVariable

<table>
<tr><th colspan="5">通信サンプル 1 – Variable_GetValue (“I40”)</th></tr>
<tr><td colspan="5">この関数は, 変数“I40”の値を返します.</td></tr>
<tr><td rowspan="4">送信
パケット</td><td colspan="4">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>hVariable</td><td>変数ハンドル</td><td>VT_I4</td><td>0x0020028
0A
00 00 00 03 00 01 00 00 00 28 00 02 00</td></tr>
<tr><td rowspan="4">受信
パケット</td><td colspan="4">サーバ→クライアント:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>pVal</td><td>変数“I40”の値</td><td>VT_I4</td><td>0x12345678
0A
00 00 00 03 00 01 00 00 00 78 56 34 12</td></tr>
</table>

<table>
<tr><th colspan="5">通信サンプル 2 – Variable_GetValue (“D40”)</th></tr>
<tr><td colspan="5">この関数は, 変数“D40”の値を返します.</td></tr>
<tr><td rowspan="4">送信
パケット</td><td colspan="4">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>hVariable</td><td>変数ハンドル</td><td>VT_I4</td><td>0x40028
0A
00 00 00 03 00 01 00 00 00 28 00 04 00</td></tr>
<tr><td rowspan="4">受信
パケット</td><td colspan="4">サーバ→クライアント:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td>バイナリ</td><td></td><td></td></tr>
<tr><td>pVal</td><td>変数“D40”の値</td><td>VT_R8</td><td>3.1415
0E
00 00 00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09
40</td></tr>
</table>

通信サンプル 3 – Variable_GetValue (“P40”)				
この関数は, 変数“P40”の値を返します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x60028
		0A 00 00 00 03 00 01 00 00　00 28 00 06 00		
受信 パケット	サーバ→クライアント: 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			
	引数	説明	データ型	値
		バイナリ		
	pVal	変数“P40”の値	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		

通信サンプル 4– Variable_GetValue (“@ ERROR_DESCRIPTION”)				
コントローラ変数“@ERROR_DESCRIPTION”の値を取得します.				
送信 パケット	クライアント→サーバ: 01 1E 00 00 00 04 00 00　00 65 00 00 00 01 00 0A 00 00 00 03 00 01 00 00　00 04 00 01 00 04			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x10004
		0A 00 00 00 03 00 01 00 00　00 04 00 01 00		
受信 パケット	サーバ→クライアント: 01 46 00 00 00 04 00 00　00 00 00 00 00 01 00 32 00 00 00 08 00 01 00 00　00 28 00 00 00 52 00 65 00 6C 00 65 00 61 00 73　00 65 00 20 00 6D 00 61 00 63 00 68 00 69 00 6E　00 65 00 20 00 6C 00 6F 00 63 00 6B 00 04			
	引数	説明	データ型	値

		バイナリ		
	pVal	エラーのメッセージ文字列 注 : ASCII コードの文字列です.（BSTR ではありません. ）	VT_BSTR	“Release machine lock”
		32 00 00 00 08 00 01 00 00 00 28 00 00 00 52 00 65 00 6C 00 65 00 61 00 73 00 65 00 20 00 6D 00 61 00 63 00 68 00 69 00 6E 00 65 00 20 00 6C 00 6F 00 63 00 6B 00		

通信サンプル 5– Variable_GetValue (“@HIGH_CURRENT_POSITION”)				
ロボット変数“@HIGH_CURRENT_POSITION”の値を取得します.				
送信 パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x30020B
		0A 00 00 00 03 00 01 00 00 00 0B 02 03 00		
受信 パケット	サーバ→クライアント: 01 3A 00 00 00 06 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 9C 23 4F 43 00 00 00 00 00 00 D0 4A 04			
	引数	説明	データ型	値
		バイナリ		
	pVal	高分解能現在位置(P 型) + タイムスタンプ	VT_R4|ARRAY	-215.1631, 21.47029, 243.6481, 0, 0, 207.1391, 0, 6815744
		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 9C 23 4F 43 00 00 00 00 00 00 D0 4A		

6.5.3. Variable_PutValue

関数　　HRESULT Variable_PutValue ()

関数 ID　　102

引数　　[in]　long　hVariable　変数ハンドル

　　[in]　VARIANT　newVal　値

戻り値　　表 4-5 既定のリターンコード一覧 参照

説明 “hVariable”で指定された変数オブジェクトの値を書き込みます.

参照項目 Controller_GetVariable
Robot_GetVariable
Task_GetVariable

通信サンプル 1 – Variable_PutValue (“I40”)				
ハンドルで指定した変数“I40”に新しい値を書き込みます.				
送信パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x0020028
		00 00 00 03 00 01 00 00 00 28 00 02 00 0A		
	newVal	新しく書き込む値	VT_I4	0x12345678
		00 03 00 01 00 00 00 78 56 34 12 0A 00 00		
受信パケット	サーバ→クライアント: 01 10 00 00 00 34 03 00 00 00 00 00 00 00 00 04			
	引数	説明	データ型	値
		バイナリ		
	なし	-	-	-
		-		

通信サンプル 2 – Variable_PutValue (“D40”)				
ハンドルで指定した変数“D40”に新しい値を書き込みます.				
送信パケット	クライアント→サーバ: 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			
	引数	説明	データ型	値
		バイナリ		
	hVariable	変数ハンドル	VT_I4	0x0040028
		00 00 00 03 00 01 00 00 00 28 00 04 00 0A		
	newVal	新しく書き込む値	VT_R8	3.1415

<table>
<tr><td></td><td></td><td colspan="3">0E 00 00
00 05 00 01 00 00 00 6F 12 83 C0 CA 21 09 40</td></tr>
<tr><td rowspan="4">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 10 00 00 00 39 03 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="3">-</td></tr>
</table>

<table>
<tr><td colspan="5">通信サンプル 3 – Variable_PutValue (“P40”)</td></tr>
<tr><td colspan="5">ハンドルで指定した変数“P40”に新しい値を書き込みます.</td></tr>
<tr><td rowspan="8">送信
パケット</td><td colspan="4">クライアント→サーバ:
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</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>hVariable</td><td>変数ハンドル</td><td>VT_I4</td><td>0x0060028</td></tr>
<tr><td></td><td colspan="3">00 0A
00 00 00 03 00 01 00 00 00 28 00 06 00</td></tr>
<tr><td>newVal</td><td>新しく書き込む値</td><td>VT_R4|ARRAY</td><td>(10,20,30,40,50,60,5)</td></tr>
<tr><td></td><td colspan="3">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</td></tr>
<tr><td rowspan="4">受信
パケット</td><td colspan="4">サーバ→クライアント:
01 10 00 00 00 39 03 00 00 00 00 00 00 00 00 04</td></tr>
<tr><td>引数</td><td>説明</td><td>データ型</td><td>値</td></tr>
<tr><td></td><td colspan="3">バイナリ</td></tr>
<tr><td>なし</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td colspan="3">-</td></tr>
</table>

7. ロボット動作命令の実行概要

ロボットオブジェクトの動作命令(Move, Speed, Tool, Work)の実行は, RC7M/J コントローラ内部の b-CAP サーバがコントローラ内で動作する PAC プログラムである RobSlave(RobSlave.pac)と通信を行って, PAC プログラムが指示された PAC コマンドを実行することで実現されています. b-CAP サーバと RobSlave 間の通信には, コントローラのグローバル変数 I[0],T[0],T[1]の変数が使用されます. I[0]は実行中のコマンドの状態(実行中, 完了, エラー等)を表し, T[0]は b-CAP サーバから RobSlave へのコマンドとパラメータを渡すために使用されます. T[1]は RobSlave から b-CAP サーバを介してクライアントプログラムへ値を返すために使用されます.

ロボット動作コマンドの実行手順を以下に図で示します.

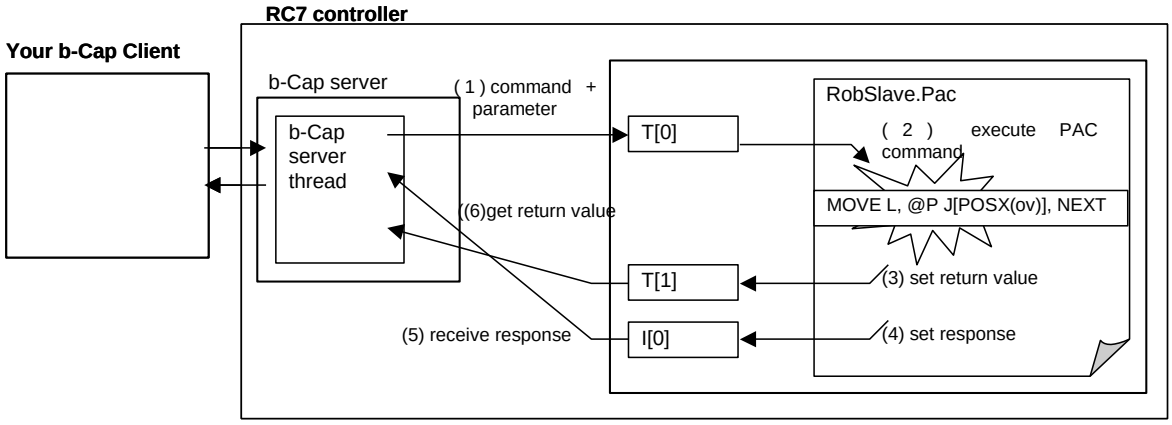


図 7-1 ロボット動作コマンドの実行手順

【 注記 】

全グローバル変数(I,F,D,V,P,J,T,S)のインデックス[0]から[9]まで変数はシステム予約しています.

これらの変数へのアクセスはユーザープログラムでは行わないでください.