RC8 プロバイダ

デンソー ロボット RC8 対応

Version 1.1.9

ユーザーズ ガイド

May 16, 2022

【備考】

【改版履歴】

バージョン	日付	内容
1.1.0	2012-09-10	初版
1.1.1	2012-11-19	VRC 接続追加, Hand オブジェクト追加, スプライン関連コマンド追加, ExtSpeed コマンド修正
	2013-02-06	Move, Rotate, Draw, Approach, Depart, DriveEx, DriveAEx, RotateH 動作オプション修正
	2013-06-27	@IfNotMember オプションの扱い追記
	2013-07-09	ログ取得 I/F 追加
	2013-07-24	KillAll, SuspendAll, StepStopAll, に同期フラグ追加
		CurJntEx,HighCurJntEx,DestJntEx,CurPosEx,HighCurPosEx,DestPosEx, CurTrnEx,HighCurTrnEx,DestTrnEx 追加
	2013-08-01	KillAllTsr, RunAllTsr
		ForceParam に制御割合, 最大並進速度, 最大回転速度の引数追加
		ForceValue 追加
		ForceWaitCondition 追加
		ForceChangeTable 追加
	2013-12-06	DPS コマンド追加
1.1.2	2014-02-24	付録 C 追記
	2014-03-05	SysInfo,RobInfo 追加
	2014-03-06	無停止教示点補正機能追加
	2014-04-02	GetPublicValue, SetPublicValue 追加
	2014-09-08	SyncTimeStart,SyncTimeEnd,SyncMoveStart,SyncMoveEnd,SetBaseDef, GetBaseDef,SetHandIO,GetHandIO,StartSrvLog,ClearSrvLog,StopSrvLog 追加
	2014-11-06	GetCtrlLogMaxTime,SetCtrlLogMaxTime,GetCtrlLogInterval,SetCtrlLogInterval を追加
1.1.3	2015-01-08	イベント一覧追記
		ロボットクラスのシステム変数に Base*を追加
		DetectOn,DetectOff,PluralSarvoData 追加
1.1.3	2015-05-26	GenerateNonStopPath コマンドの内容を追記
1.1.3	2015-07-30	AngularTrigger,GetCurErrorCount,GetCurErrorinfo を追加
1.1.3	2017-04-27	VirtualFence, ExclusiveArea, SetExclusiveArea, ResetExclusiveArea, ExclusiveControlStatus を追加
1.1.3	2017-04-27	付録 D.2. エラーコード にエラーの復帰処置を追加, 誤記修正

1.1.3	2017-05-22	GetAllSrvData,CurForceParam を追加
1.1.3	2018-03-01	GenerateNonStopPath コマンドの内容を修正 付録 D.1. パラメータの内容を修正
1.1.4	2018-09-18	GetLogCount,GetLogRecord を追加
1.1.5	2019-02-26	Ver2.8.0 以降 ・Cobotta 専用コマンドを追加 ・変数と IO の配列アクセス用変数を追加
1.1.5	2019-07-04	・GetSrvData, GetSrvJntData の戻り値の内容を追加
1.1.6	2019-08-27	Ver2.8.0 以降 ・COBOTTA 電動バキュームと K3 ハンドのコマンドを追加
1.1.6	2019-08-30	・Cobotta 専用コマンドのコマンド一覧を追加 ・コマンド一覧にバージョン情報を追加
1.1.6	2020-03-06	3.1.1. RC8 プロバイダの提供する機能の内容を修正
1.1.7	2020-05-18	Ver2.11.0 以降 GetForceLogRecord, CurExtSpd, CurExtAcc, CurExtDec を追加
1.1.7	2020-06-24	対応機種に RC8A を追加
1.1.7	2021-01-11	Cobotta 専用コマンドに ManualResetPreparation を追加
1.1.7	2021-03-19	GetAllSrvData コマンドの内容を修正
1.1.8	2021-06-02	・システム変数を追加（@IO_ALLOC_MODE, @STATE, @ASP_MODE, @ERROR_CODE, @ERROR_DESCRIPTION, @LOCK） ・Cobotta 専用コマンドに GetDirectMode, GetMechaButtonState, ClearMechaButtonState を追加
1.1.9	2022-01-24 2022-05-16	AddController メソッドを修正 「5.3.3. タスククラス」の誤記修正

【対応機器】

機種	バージョン	注意事項
RC8		
RC8A		

【動作確認機器】

機種	バージョン	注意事項
RC8	1.2.4	

【対応コマンド】

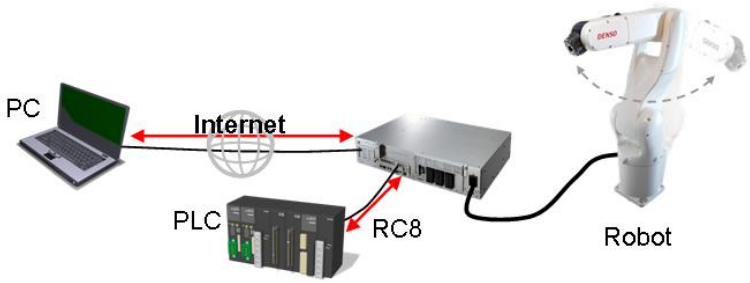
機種	バージョン	注意事項
RC8	1.3.6	スプライン関連コマンドを追加

目次

1. はじめに

本書は，デンソーロボット RC8 コントローラ用（**VRC**[1]を含む）の ORiN Ver2 仕様に準拠する CAO プロバイダの外部仕様を規定するものです．この RC8 コントローラ用（VRC を含む）CAO プロバイダを，以下，RC8 プロバイダと呼びます．

本書では，RC8 プロバイダの接続方法，変数や I/O アクセス，ファイル操作，タスク制御，ロボット制御，ハンド制御および独自拡張に関する仕様を記載しています．



1.1. 本書が想定している環境とバージョン

クライアント PC が Windows 上で動作し，対象とするロボットコントローラが RC8 以上である環境を想定しています． PC の開発環境は，Component Object Model（COM，コンポーネント・オブジェクト・モデル）をサポートするプログラミング環境であれば開発が可能です．

1.2. 参考となる情報源

本書のプログラミング事例は，すべて Visual Basic 6.0 で記載していますが，C++，Java，.NET，LabVIEW，Delphi などさまざまなプログラム言語で開発が可能です． 使用方法に関しては，「ORiN 2 プログラミングガイド」を参照ください．

「ORiN 2 プログラミングガイド」は ORiN2 SDK インストールフォルダの以下のファイルに該当します．

・ ORiN2¥CAO¥Doc¥ORiN2_ProgrammersGuide_<*lang*>.pdf

※<lang>の部分は環境毎の言語文字列に置き換えてお読みください．

プロバイダを使ったアプリケーションを開発する上で必要となる ORiN2，COM/DCOM の基礎知識や技術に関して例を交えながら解説されています．

その他,必要に応じて以下のドキュメントを参照ください．

b-CAP プロバイダユーザーガイド

・ ORiN2¥CAO¥ProviderLib¥b-CAP¥Doc¥b-CAP_ProvGuide_<*lang*>.pdf

NetwoRC プロバイダユーザーガイド (RC7 コントローラ用プロバイダ)

・ ORiN2¥CAO¥ProviderLib¥DENSO¥NetwoRC¥Doc¥NetwoRC_ProvGuide_<*lang*>.pdf

※RC8 コントローラの機能に関しては RC8 取扱説明書を参照ください．

[1] VRC（バーチャルロボットコントローラ）はデンソーの製品です．ご使用になる場合は，別途“VRC”ライセンスが必要です．

2. アプリケーション開発のための環境セットアップ

2.1. PC 開発環境のセットアップ

2.1.1. RC8 プロバイダの自動インストール

ORiN2 SDK Ver 2.1.9 以上から RC8 プロバイダがインストーラによりセットアップされます.

ORiN2 SDK Ver2.1.9 以上がインストールされている環境であれば, RC8 ロボットコントローラ(以下, ロボットコントローラ)に接続するための動作環境(ラインタイム)の準備は完了です.

開発環境のセットアップは別途, Microsoft Visual Studio 6.0, 2003/2005/2008/2010, LabVIEW など Component Object Model(COM, コンポーネント・オブジェクト・モデル)をサポートする, プログラミング環境をご準備してください.

2.1.2. RC8 プロバイダの手動インストール

RC8 プロバイダを使用するにあたって, インストーラを使用せずにセットアップを行う場合は手作業で下記レジストリ登録を行う必要があります.

表 2-1 RC8 プロバイダ

ファイル名	CaoProvRC8.dll
ProgID	CaoProv.DENSO.RC8
レジストリ登録	Regsvr32 CaoProvRC8.dll
レジストリ登録の抹消	Regsvr32 /u CaoProvRC8.dll

CAO エンジンが動作するには予め, PC 毎に正規の ORiN2 SDK ライセンスが 1 つ登録されていなくてはなりません. ORiN2 SDK ユーザーズガイド内にある「ライセンスの追加と削除」の節を参照してください.

2.2. RC8 コントローラのセットアップ

2.2.1. 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.

(1) 非常停止スイッチは, 赤色にしてください.
(2) 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.
(3) 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.

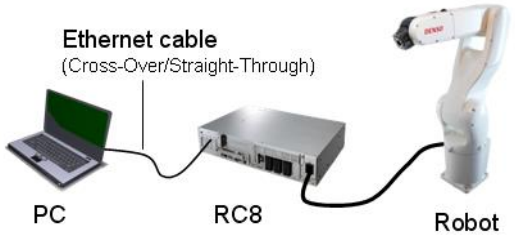
2.2.2. ハードウェアの準備

ロボットコントローラのクライアントとして構成できる基本的なハードウェアのシステムは以下の通りです. 設備を設計する際には, お客様が望むソフトウェアのシステム構成を考慮して, ハードウェアの準備を行ってく

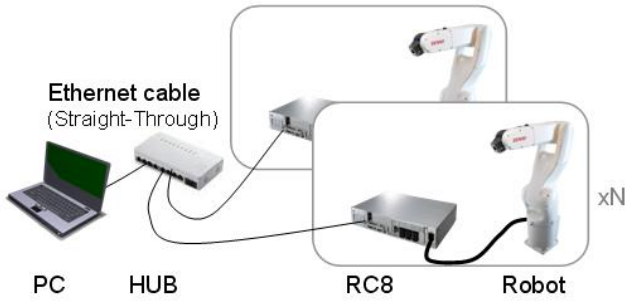
ださい.

(1) PC ベースのロボットシステム

・ RC8 一台の構成の場合



・ RC8 複数台の構成の場合



(2) RC8 ベースのロボットシステム

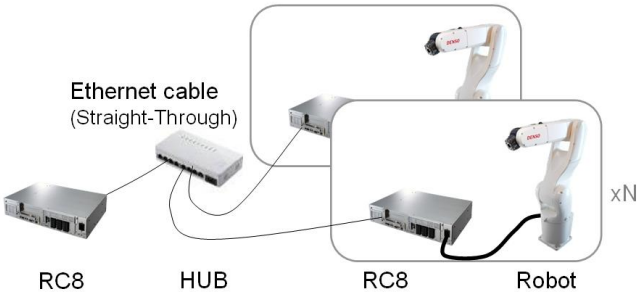


表 2-2　ロボットシステム別構成

ハードウェア		ソフトウェア			
クライアント	接続形態	OS	開発言語	依存モジュール	備考
(1) PC ベース	Ethernet (TCP/IP)	Windows	C,C++,C#,VB,VBA, Java,LabVIEW Delphi, Python, Ruby,... (DCOM 技術サポート環境)	ORiN2 SDK (CAO, RC8/b-CAP プロバイダ)	・ORiN2 技術を用いて RC8 がサポートする全 API が利用可能 ・クライアント PC 用に別途 ORiN2 SDK が必要
		Linux (その他)	C,C++ (Socket 通信サポート環境)	Socket ライブラリ	・Socket 通信技術を用いて b-CAP プロトコルに対応することで RC8 がサポートする全 API が利用可能
(2) RC8 ベース	Ethernet, I/O	RC8 依存 Windows	PacScript (VBA ベース)	ORiN2 SDK 標準搭載 (CAO, RC8/b-CAP プロバイダ)	・標準搭載の機能で RC8 がサポートする全 API が利用可能

2.2.3. システムパラメータの設定

RC8 プロバイダを使用する前に，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システムパラメータ設定には，ティーチングペンダント（TP）もしくはミニペンダント（MiniTP）のどちらかが必要となります．システムパラメータとして設定が必要になるのは，①通信権と②起動権です．

通信権は，ロボットコントローラに対してデータの読み込みと書き込みの権限を，通信デバイスに与えるための設定です．変数データの書き込みやロボット制御を行う場合は，書き込みの権限を与えてください．

起動権は，ロボットコントローラに対してプログラムタスクの起動（実行），モータ ON 及びロボット制御（動作指令）の権限を，通信デバイスに与えるための設定です．設定可能な値は，①TP，②I/O，③Ethernet，④Any のいずれかです．Any の場合は通信経路を問わずいつでも起動権が与えられますので，Any に設定する場合はクライアント PC や PLC 側で衝突が起こらないよう，通信デバイス間で排他処理を施してください．

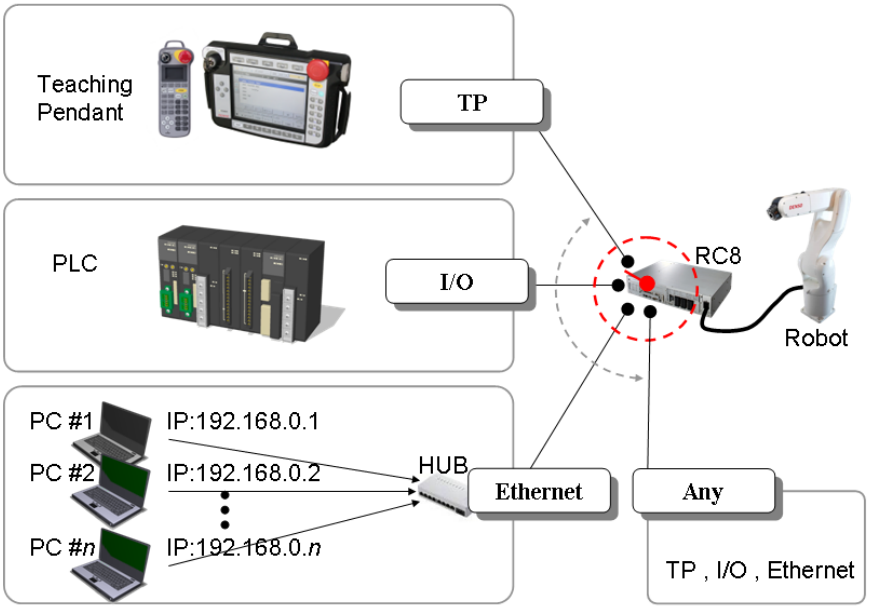


図 2-1 起動権を持つデバイスの設定

接続方法として Ethernet を用いる場合は，クライアント PC の IP アドレスを設定する必要があります．この設定により，ロボットコントローラは特定のクライアント PC からのみプログラムタスクの起動やロボット制御を可能にします．

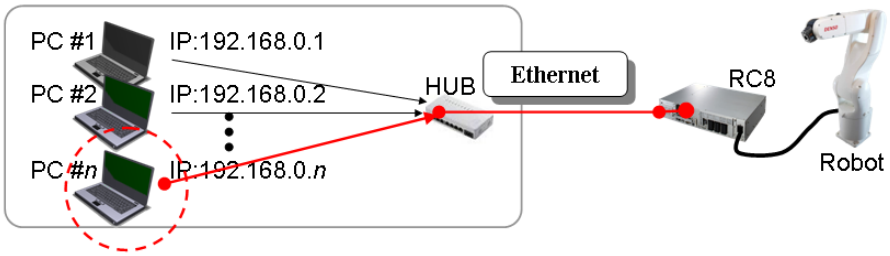


図 2-2 起動権を持つクライアントの設定

以降，それぞれを使った設定の方法について記述します．

2.2.3.1. ティーチングペンダントを用いた設定

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 IP アドレス設定は，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．

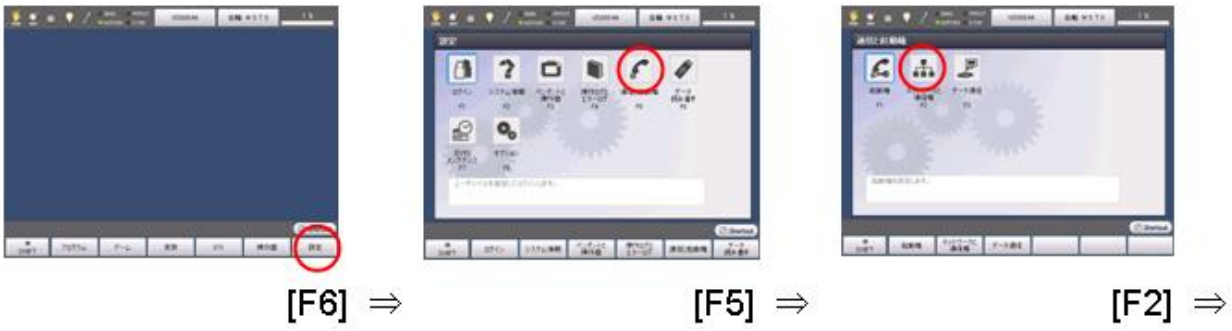
(2) ロボットコントローラの**起動権を設定**します.
接続方法として Ethernet を用いる場合は, ティーチングペンダントの[設定(F6)]メニュー ⇒ [通信と起動権(F5)] ⇒ [起動権(F1)]で起動権を Ethernet に設定します.

図 2-3 起動権設定

さらに，[編集(F5)]を押し，ロボットコントローラに起動権を与えるクライアントの IP アドレスを設定します．

図 2-4 クライアントの IP アドレス設定

(3) ロボットコントローラの<u>ネットワークと通信権を設定</u>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[設定(F6)]メニュー ⇒ [通信と起動権(F5)] ⇒ [ネットワークと通信権(F2)]でイーサネット(Ethernet)に読み込み・書き込み権限を設定します．



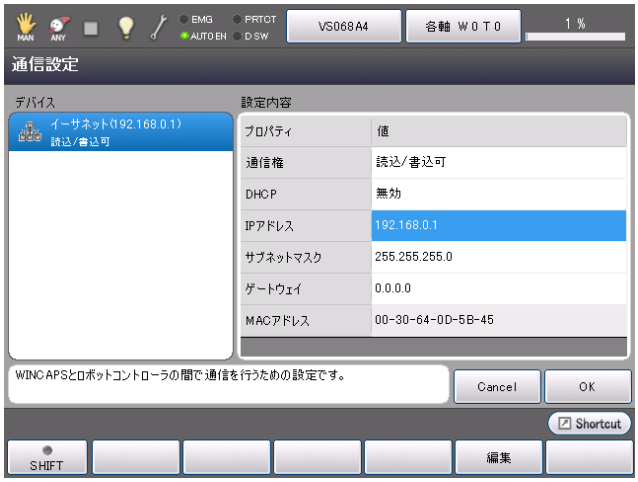


図 2-5 通信設定

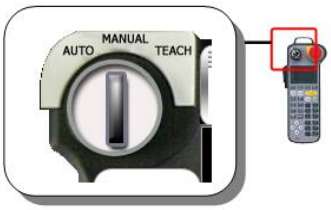
さらに，[編集(F5)]を押し，ロボットコントローラの IP アドレスとサブネットマスクを設定します．また，必要に応じてゲートウェイのアドレスを設定します．

図 2-6 ロボットコントローラの IP アドレス設定

2.2.3.2. ミニペンダントを用いた設定

ミニペンダントを用いたロボットコントローラの IP アドレス設定は，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) ロボットコントローラの**起動権を設定**します.
[通信]を押して, 次のような「通信設定一覧」を表示します.

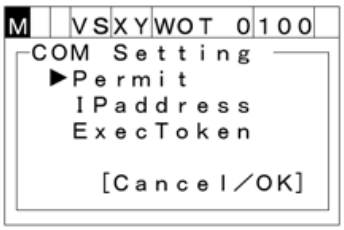


図 2-7 通信設定一覧

上下カーソルキーで, "ExecToken"を選択し[OK]を押して,「起動権設定一覧」を表示します.

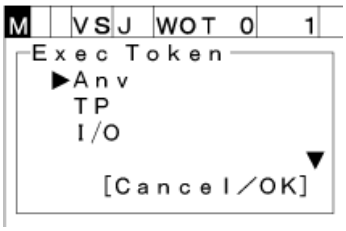


図 2-8 起動権設定一覧

上下カーソルキーで, "Ether"を選択し[OK]を押します. 次のような「クライアントIPアドレス」が表示されます. ロボットコントローラに起動権を与えるクライアントの IP アドレスを設定します.

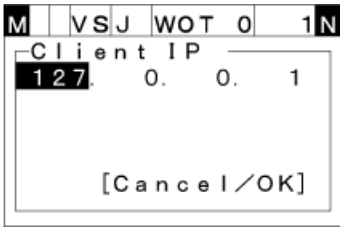


図 2-9 クライアントIPアドレス設定

[OK]を押すと, 変更が確定します.
[Cancel]を押すと, 変更が無効となります.

(3) ロボットコントローラの<u>通信権を設定</u>します.
[通信]を押して, 次のような「通信設定一覧」を表示します.

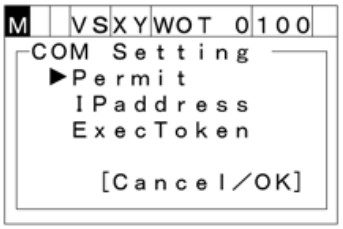


図 2-10 通信設定一覧

上下カーソルキーで, "Permit"を選択し[OK]を押して, 次のような「ポート選択一覧」を表示します.
(Off):使用不可, (R):読込みのみ可, (RW):読込み／書込み可
[Cancel]を押すと, 通信設定は終了します.

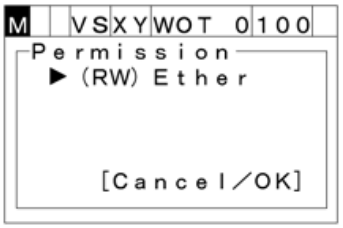


図 2-11 ポート選択一覧

“Ether"を選択し, [OK]を押します. 次のような「通信選択一覧」が表示されます.
[Cancel]を押すと, 通信設定は終了します.

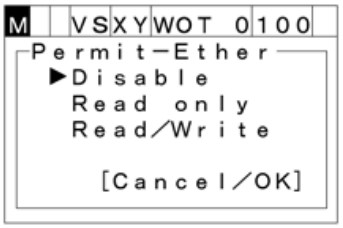


図 2-12 通信選択一覧

上下カーソルキーで, 通信権を"Disable"(使用不可), "Read only"(読込みのみ可),
“Read/Write"(読込み／書込み可)から選択し, [OK]を押して, 通信権を変更します.
[Cancel]を押すと, 通信権の変更は無効になります.

(4) ロボットコントローラのネットワークを設定します.
[通信]を押して, 次のような「通信設定一覧」を表示します.

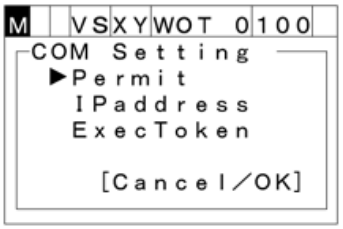


図 2-13 通信設定一覧

上下カーソルキーで, "IP address"を選択し[OK]を押すと, 次のような「IP アドレスの設定画面」が表示されます.
[Cancel]を押すと, 通信設定は終了します.

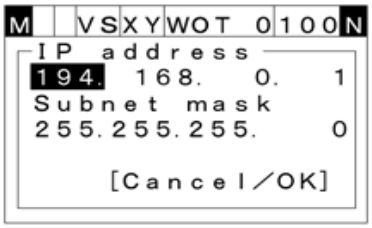


図 2-14 IPアドレス設定画面

項目を上下左右カーソルキーで選択し, 数値入力キーで値を変更することができます.

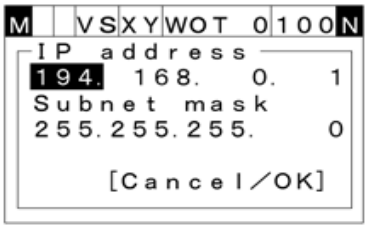


図 2-15 IPアドレスの値変更

[OK]を押すと, 変更が確定します.
[Cancel]を押すと, 変更が無効となります.

2.3. CaoTester を使った動作確認

開発クライアントアプリケーションを動作させる前に，ORiN2 SDK 標準ツールである CaoTester を使用して制御対象となる RC8 ロボットコントローラのセットアップが完了しているかを確認します．

2.3.1. 変数アクセスの確認

CaoTester を使った変数アクセスの動作を実施して，クライアント PC が対象のロボットコントローラと基本的な接続ができているかの確認を以下の手順で行います．この操作が正しく行えない場合はクライアント PC のインストール環境あるいは対象ロボットコントローラのネットワーク環境と設定を疑って，再度セットアップを行ってください．

(1) CaoTester を起動します．
CaoTester は ORiN2 SDK のインストールフォルダ以下の
[ORiN2¥CAO¥Tools¥CaoTester¥Bin¥CaoTester.exe]を選択して起動してください．

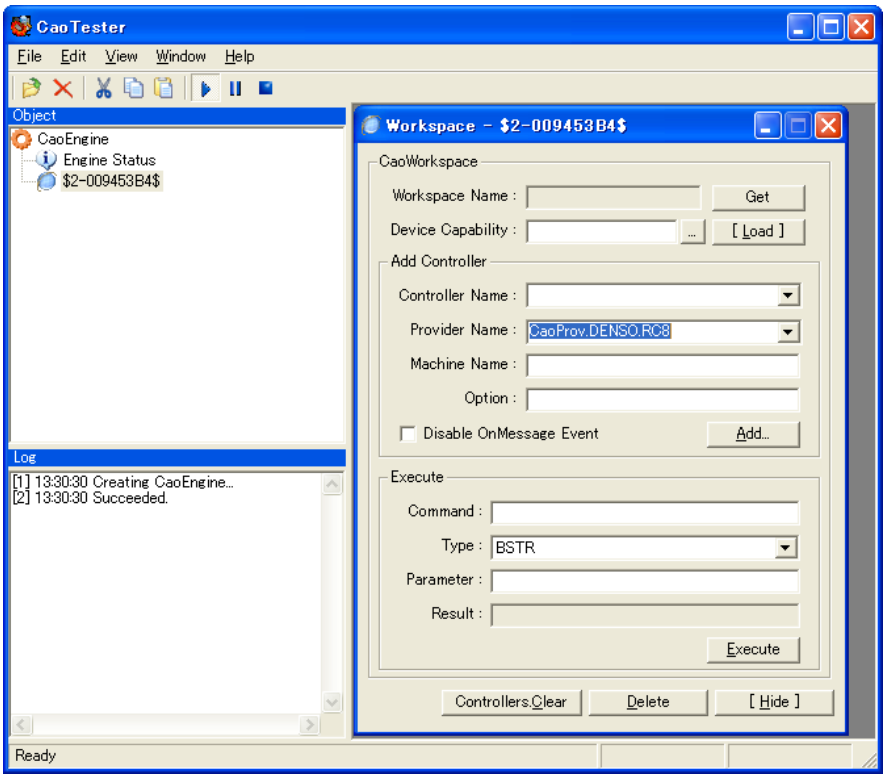


図 2-16 CaoTester の初期画面

(2) Workspace ウィンドウを選択し，Add Controller のパラメータを設定します.
説明の都合上，ここでは対象コントローラの IP アドレスが 192.168.0.1 である前提で説明しますのでお客様の実際の環境に置き換えて設定を行ってください.

Controller Name	:	RC8
Provider Name	:	CaoProv.DENSO.RC8
Machine Name	:	<空白>
Option	:	Server=*192.168.0.1* ※対象コントローラの IP アドレス

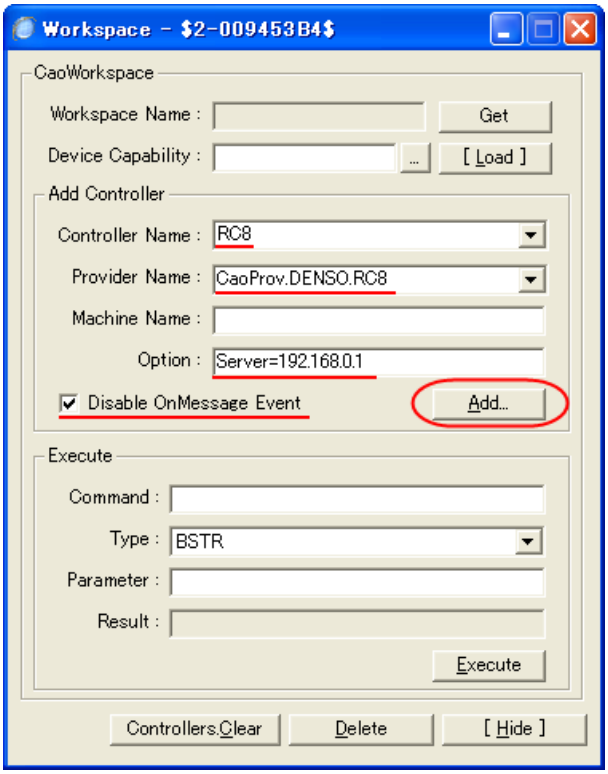


図 2-17 Workspace ウィンドウ

(3) Worksapace ウィンドウの Add ボタンを押し，CaoController ウィンドウを表示します．

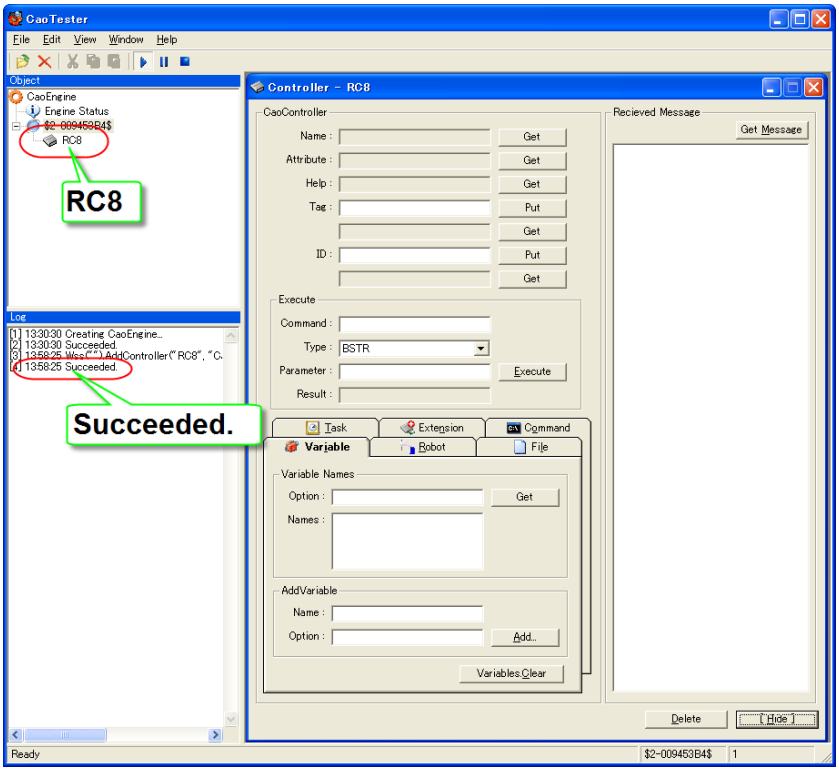


図 2-18 Controller ウィンドウ作成時の CaoTester 画面

(4) Controller ウィンドウの Variable の AddVariable で I1 変数用 Variable ウィンドウを作成します．

Name : I1

Option : <空白>

AddVariable で上記パラメータを設定し，[Add..]ボタンを押してください．

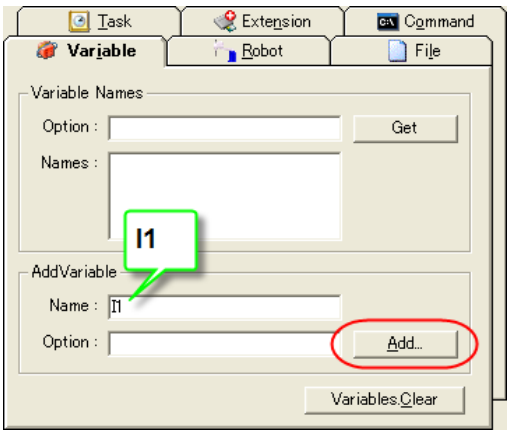


図 2-19 Variable タブの設定

(5) Variable ウィンドウの Value で変数にアクセスします.
[Get]および[Put]ボタンを押して対象コントローラの値にアクセスしてください.

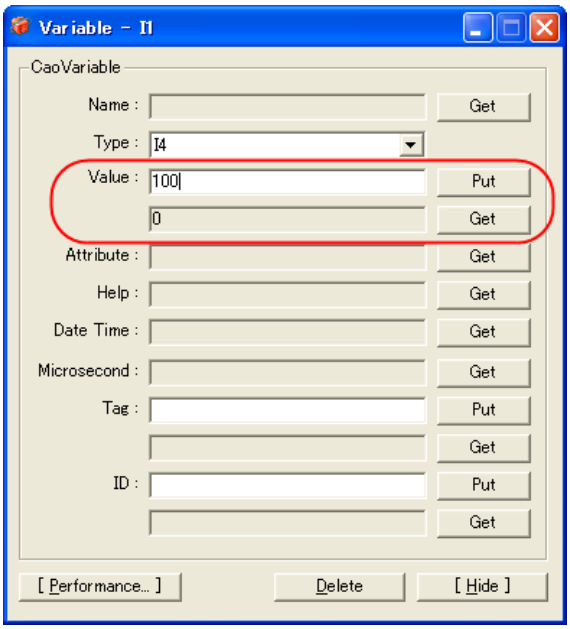


図 2-20 Variable ウィンドウの Value 設定

2.3.2. モータ ON の確認

CaoTester を使ったモータ ON/OFF の動作を実施して, クライアント PC が対象のロボットコントローラに対してモータ制御ができているかの確認を以下の手順で行います. この操作が正しく行えない場合はクライアント PC に対して対象ロボットコントローラの起動権の設定が間違っている可能性がありますので, 再度起動権の設定を確認してください.

(1) ロボットコントローラを**自動モードに設定**します.

(2) CaoTester の Controller ウィンドウを選択し，Robot タブで Robot ウィンドウを作成します．

Name : Arm0

Option : <空白>

AddRobot で上記パラメータを設定し，[Add..]ボタンを押してください．

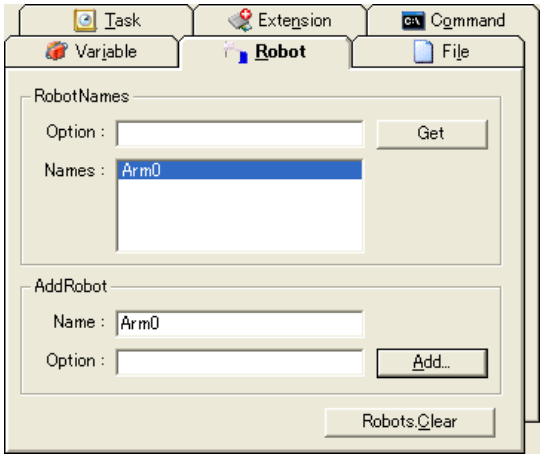


図 2-21 Robot タブの設定

(3) Robot ウィンドウの Variable の AddVariable で@SERVO_ON 用 Variable ウィンドウを作成します．

Name : @SERVO_ON

Option : <空白>

AddVariable で上記パラメータを設定し，[Add..]ボタンを押してください．

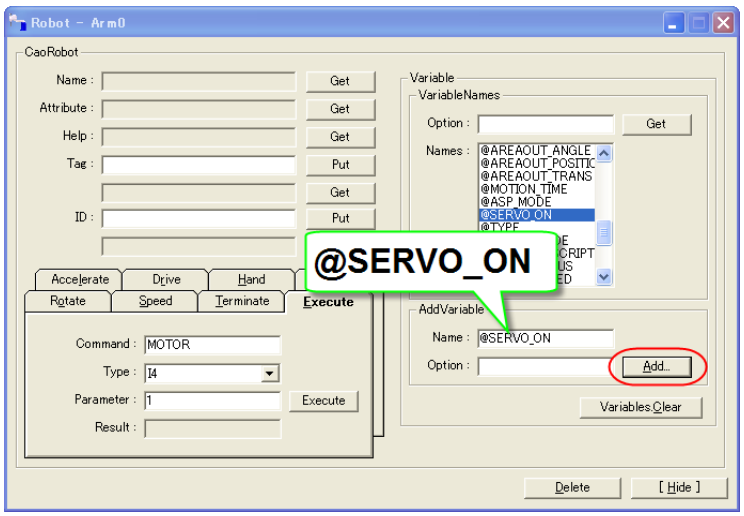


図 2-22 Robot ウィンドウの設定

(4) Variable ウィンドウの Value でモータの ON/OFF を実行します.

[Get]および[Put]ボタンを押して対象コントローラのモータの ON(1)/OFF(0)を実行してください.

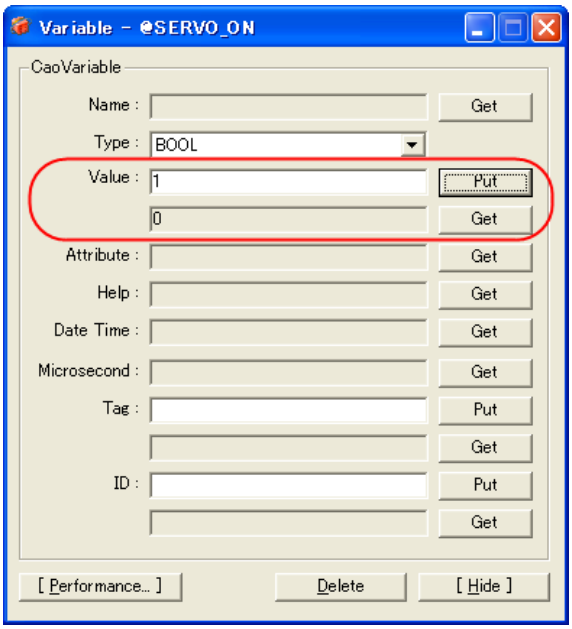


図 2-23 Variable ウィンドウの Value 設定

3. RC8 プログラミングのための基礎知識

3.1. RC8 プロバイダの概要

3.1.1. RC8 プロバイダの提供する機能

RC8 プロバイダは，ロボットコントローラが外部に対して提供する全機能を呼び出せるように，ORiN2 に準拠した様々な API を提供しています．

RC8 プロバイダが提供する機能概要は，以下の内容になります．詳細に関してはは「5.コマンドリファレンス」を参照してください．

表 3-1 RC8 プロバイダの機能概要

機能名	カテゴリ	備考
イベント通知	CaoController	コントローラのエラー通知や状態の変化を OnMessage イベントとして非同期で受け取ることが可能
変数アクセス	CaoVariable	I/O，グローバル変数，システムパラメータの読み書きが可能 また，様々なコントローラリソースの情報や状態をも取得が可能
ファイル操作	CaoFile	ファイルやフォルダに対する情報取得や操作が可能
タスク制御	CaoTask	実行タスクの状態取得，起動，停止が制御可能 また，タスクのメッセージキューを利用したタスク間通信も可能
ロボット制御	CaoRobot	モータの ON/OFF，ロボットの動作速度/動作指令，TOOL/WORK/AREA の設定などロボットの制御が可能
拡張ボード	CaoExtension	標準ハンドのパラメータ設定/取得，状態取得，動作指令の制御が可能

3.1.2. RC8 プロバイダのシステム構成

RC8 プロバイダは，ロボットコントローラのハードウェアに依存しないコアなモジュールです．実機およびシミュレーションでプロバイダを共有化することで高い互換性を提供し，高精度なシミュレーションを実現しています．

PC とロボットコントローラとの接続，および VRC との接続におけるシステム構成を以下に示します．

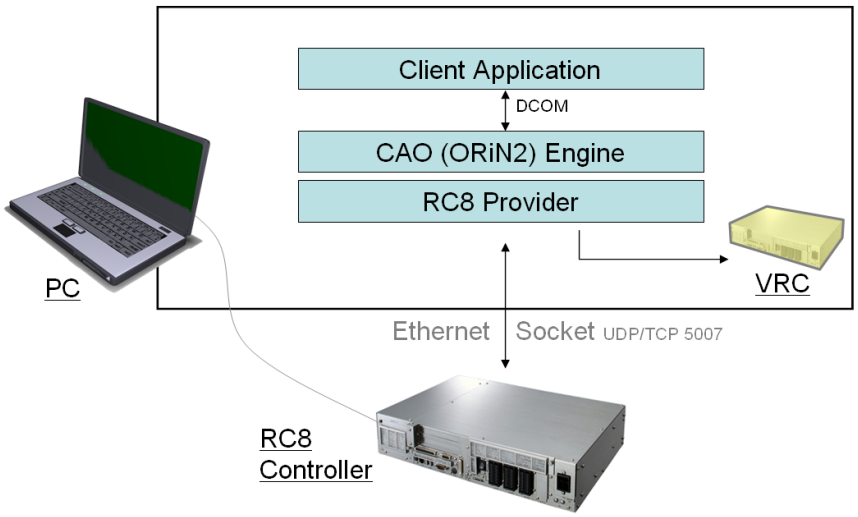


図 3-1 PC と RC8 のシステム構成

この接続形態は，クライアントアプリケーションが RC8 プロバイダを接続（AddController）する際に指定する接続パラメータによって判別されます．RC8 プロバイダに対して，ロボットコントローラの IP アドレス（”Server=...”）を指定した場合はロボットコントローラへの接続となり，WINCAPS3 のプロジェクトファイル（”wpj=...”）を指定した場合は VRC への接続になります．

3.1.2.1. CAO エンジンと CAO プロバイダの構成

RC8 プロバイダなどの CAO プロバイダは ORiN2 の CAO エンジンのプラグインであるため，クライアントアプリケーションを作成するには，まず，CAO エンジンのクラス構成を理解する必要があります．

以下に CAO エンジンと CAO プロバイダのクラス構成を示します．

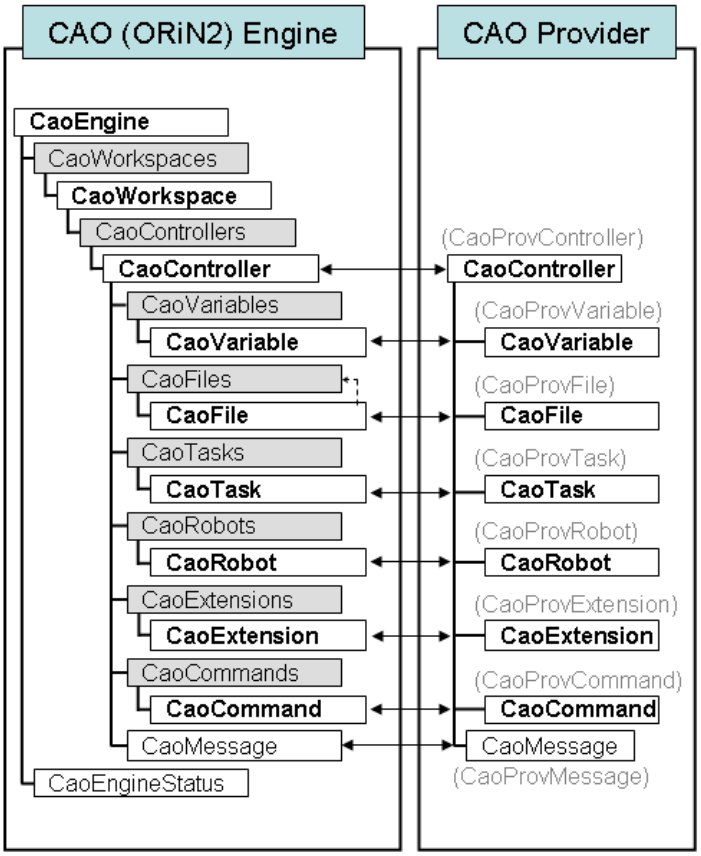


図 3-2 CAO エンジンとプロバイダの構成

CAO エンジンのクラスはロボットコントローラを含め，一般的なデバイスが持つリソースをモデル化したものです．この CAO エンジンが提供するクラスにクライアントアプリケーションがアクセスすることで，接続されるデバイスに対して間接的にアクセスすることが可能になります．

3.1.3. HRESULT とエラーの扱いについて

Cao プロバイダの各クラスのメソッドやプロパティの応答を示す HRESULT の値は 0 以上の場合は処理が正常に行えたことを意味します. 逆に負の場合は呼び出しが失敗したことを意味します.

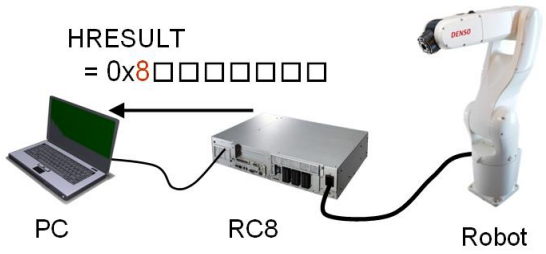


図 3-3 PC からの呼び出しエラー

HRESULT のエラーが 0x8□□□□□□□の場合は, ロボットコントローラ付属マニュアルのエラーコード表からエラーを参照してください.

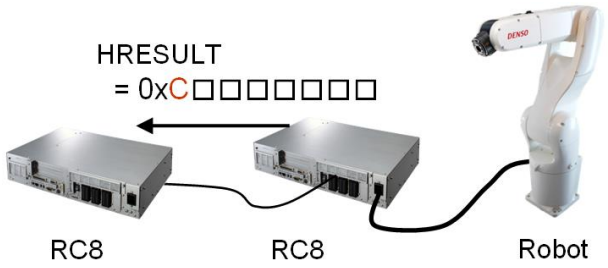


図 3-4 RC8 からの呼び出しエラー

HRESULT のエラーが 0xC□□□□□□□の場合は, 0xC⇒0x8 に置き換えて, ロボットコントローラ付属マニュアルのエラーコード表からエラーを参照してください.

3.1.4. プロパティ定義の扱いについて

Cao プロバイダの各クラスのプロパティ仕様を説明にするために, 本書では以下の表記で扱っています.

プロパティ取得 <代入変数> = Obj.PropertyName

<代入変数> = Obj.get_Property() として扱われます.

get_ PropertyName <PropertyName>プロパティの値取得

プロパティ設定 Obj.PropertyName = <設定値>

Obj.put_Property(<設定値>) として扱われます.

put_ PropertyName <PropertyName>プロパティの値設定

3.1.5. Execute メソッドと実行時バインディングについて

実行時バインディングの機能を使って対象クラスに定義していないメソッドを呼び出した場合，次の仕様で Execute メソッドが自動的に呼ばれます.

```
vntRet = Obj.CommandName( Param1, Param2, ...)
      ↓
vntRet = Obj.Execute("CommandName", Array(Param1, Param2, ... ) )
```

1. 第一引数にコマンド名が BSTR 文字列で渡る
2. 第二引数に全パラメータが VARIANT 配列で渡る

この仕様を実現するために Cao プロバイダの各クラスの Execute メソッドは以下の通りに定義されています.

書式 [<vntRet:VARIANT> =] Execute(<bstrCmd:BSTR > [,<vntParam:VARIANT>])

bstrCmd	:	[in]	コマンド名，BSTR 型文字列
vntParam	:	[in]	パラメータ，VARIANT 型配列（または単数）
vntRet		[out]	返り値，VARIANT 型

Execute メソッドの引数は，コマンドを BSTR，パラメータを VARIANT 配列で指定します.

4. プロバイダによる RC8 プログラミング

RC8 プロバイダを使用してロボットを制御するには，ORiN がインストールされた PC とロボットコントローラを Ethernet で接続する必要があります．また一部コマンドはロボットコントローラの設定が必要なものもあります．設定の方法は「2 アプリケーション開発のための環境セットアップ」，各種コマンドについては「5 コマンドリファレンス」を参照してください．

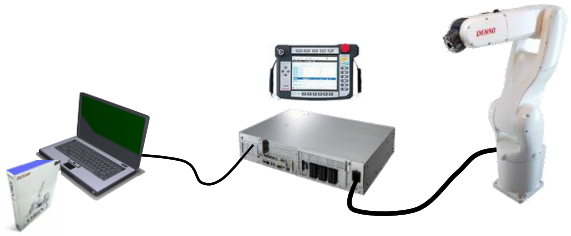


図 4-1 ロボットとの接続

作成するアプリケーションがロボットコントローラと通信するためには，RC8 プロバイダを使用し，ソケット（UDP/TCP）を生成して，ロボットコントローラと通信を行います．

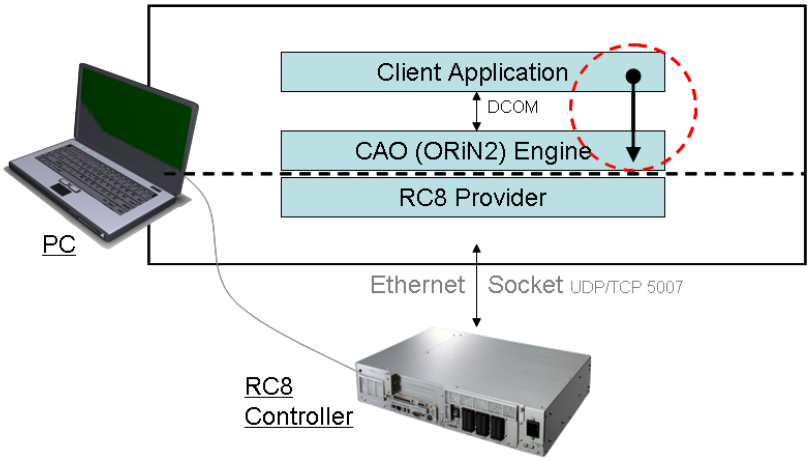


図 4-2 プログラミング概要

RC8 プロバイダでは，以下の手順で PC とロボットコントローラを接続する事ができます．

- CaoEngine の作成
- CaoWorkspace の作成
- CaoController の作成

ロボットコントローラに接続した後は，CaoVariable オブジェクトを生成することで，コントローラ内の変数にアクセスすることや，CaoRobot オブジェクトを生成することでロボットを動作させることができます．以下に具体例を挙げながら，ロボットプログラムの手順を示します．

4.1. RC8 コントローラの変数アクセス

変数アクセスを行うには， 図 4-3 に示す処理を行います.

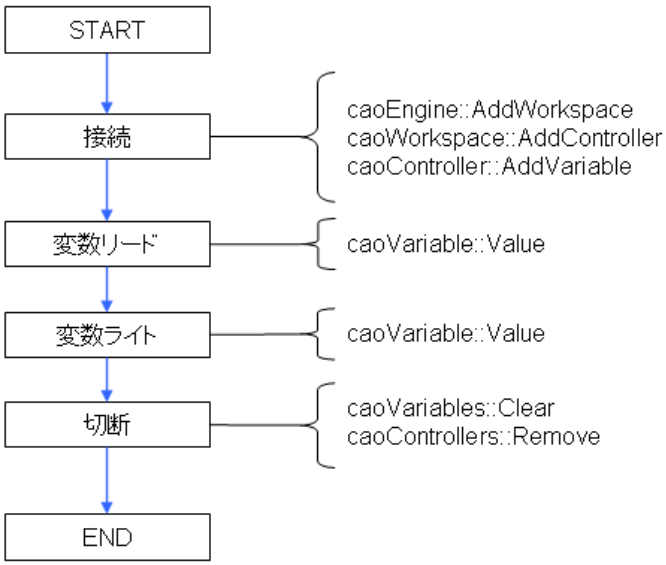


図 4-3 変数アクセスの流れ

4.1.1. 接続

コントローラとの接続を行うには，以下の手順を取ります.

(1) オブジェクトを保持するための変数を用意します. コントローラ接続に必要なオブジェクトは，CaoEngine オブジェクトと CaoWorkspace オブジェクトと CaoController オブジェクトです. CaoWorkpace オブジェクトは，CaoController オブジェクトを CaoWorkspaces から取得する場合には変数を用意する必要はありません. また変数にアクセスするための CaoVariable オブジェクトも必要になります. 以下に VB6 でのコード例を示します.

```
Dim g_eng as CaoEngine                  ‘ CaoEngine オブジェクト用の変数
Dim g_wrks as caoWorkspace              ‘ CaoWorkspace オブジェクト用の変数
Dim g_ctrl as CaoController             ‘ CaoController オブジェクト用の変数
Dim g_val as CaoVariable                ‘ CaoVariable オブジェクト用の変数
```

(2) CaoEngine オブジェクトを生成します. CaoEngine オブジェクトは New キーワードを使って生成します.

```
Set g_eng = New CaoEngine               ‘ CaoEngine オブジェクトの生成
```

(3) CaoWorkspaceオブジェクトを取得もしくは生成します．CaoEngineオブジェクトを生成すると，デフォルトで Caoworkspaces オブジェクトと Caoworkspace オブジェクトを 1 つずつ生成しています．サンプルプログラムでは，デフォルトワークスペースを利用しています．以下に CaoWorkspace オブジェクトを新しく生成するコード例を示します．

```
Set g_wrks = g_eng.Addworkspace("NewWrks", "")
```

(4) CaoControllerオブジェクトを生成します．CaoControllerオブジェクトを生成するには，使用するプロバイダ名と使用するためのパラメータを設定します．RC8 プロバイダでは，接続先のコントローラの IP アドレスをオプションで指定します．以下にコード例を示します．

```
Set g_ctrl = g_wrks.AddController("RC8", "CaoProv.DENSO.RC8", "", "Server=192.168.0.1")
```

(5) CaoVariable オブジェクトを生成します．接続したい変数の CaoVariable オブジェクトを生成します．以下に P 型の 10 番目の変数オブジェクトを生成するコード例を示します．

```
Set g_val = g_ctrl.AddVariable("P10", "")
```

4.1.2. 変数リード・ライト

接続した変数の値を取得・設定するには，CaoVariable オブジェクトの Value プロパティを参照します．変数に格納，変数からの設定を行う場合は，接続した変数型に合わせた変数を用意する必要があります．以下にコード例を示します．

```
Dim vntPose as Variant
vntPose = g_val.Value                          ' 値の取得
g_val.Value = Array(50, 50, 50, 0, 0, 0, -1)   ' 値の設定
```

4.1.3. 切断

コントローラと切断する場合には，生成したオブジェクトを消去すると共に，オブジェクトを管理するコレクションクラスから消去するオブジェクトを削除します．以下にコード例を示します．

```
g_ctrl.Variables.Clear                      ' CaoVariables から全てのオブジェクトの削除
Set g_val = Nothing                         ' CaoVariable の消去
g_wrks.Controllers.Remove g_ctrl.Index      ' CaoControllers から CaoController の削除
Set g_ctrl = Nothing                        ' CaoCtonroller の消去
g_eng.Workspaces.Remove g_wrks.Index        ' CaoWorkspaces からの CaoWorkspace の削除
Set g_wrks = Nothing                        ' CaoWorkspace の消去
Set g_eng = Nothing                         ' CaoEngine の消去
```

4.1.4. サンプルプログラム

以下に VB6 で作成したサンプルプログラムを示します. サンプルプログラムは, デフォルトワークスペースを利用し, 変数 IO150(I/O の 150 番目)の読み書きを行います. IP は各コントローラで設定した値を用いてください. サンプルプログラムでは以下の設定値を用いて接続しています.

IP:192.168.0.1

List 4-1 Variable.frm

```
Dim g_eng As CaoEngine
Dim g_ctrl As CaoController
Dim g_val As CaoVariable

Private Sub Command1_Click()
    ' 変数の読み込み
    Text1.Text = g_val.Value
End Sub

Private Sub Command2_Click()
    ' 変数の書き込み
    g_val.Value = CBool(Text2.Text)
End Sub

Private Sub Form_Load()
    Set g_eng = New CaoEngine

    ' 接続処理  IP は各コントローラの設定にしてください
    Set g_ctrl = g_eng.Workspaces(0).AddController("RC8", "CaoProv.DENSO.RC8", "",
"Server=192.168.0.1")

    ' 変数名"IO150"
    Set g_val = g_ctrl.AddVariable("IO150", "")

End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' 変数オブジェクトの破棄
    g_ctrl.Variables.Clear
    Set g_val = Nothing

    ' コントローラオブジェクトの破棄
    g_eng.Workspaces(0).Controllers.Remove g_ctrl.Index
    Set g_ctrl = Nothing

    ' CaoEngine の破棄
    Set g_eng = Nothing
End Sub
```

4.2. RC8 コントローラのタスク制御

タスク制御を行うには，図 4-4 に示す処理を行います．タスクを起動するには，コントローラが自動モードになっている必要があります．またコントローラの起動権の設定を ORiN がインストールされている PC の IP に設定してください．詳しくは「2.2.3 システムパラメータの設定」を参照してください．

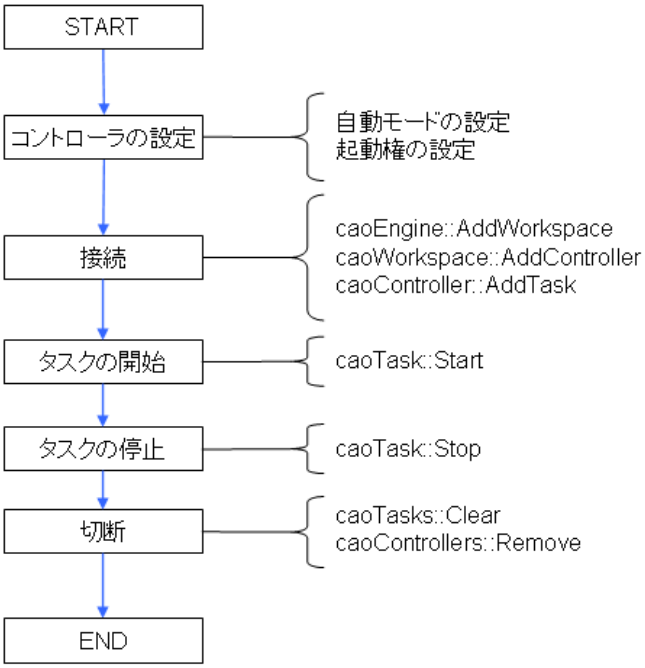


図 4-4 タスク制御の流れ

4.2.1. 接続

CaoController オブジェクトを生成するまでの手順は「4.1.1 接続」を参照してください．タスクを制御するには CaoTask オブジェクトを生成します．以下にタスクオブジェクトを生成するコード例を示します．

```
Dim g_task as CaoTask          ' CaoTask オブジェクトを格納する変数
Set g_task = g_ctrl.AddTask("PRO01", "")
```

4.2.2. タスクの開始・停止

タスクの開始と停止を行うには，CaoTask オブジェクトの Start メソッドと Stop メソッドを使用します．以下にタスクの連続実行とサイクル停止の例を示します．

```
g_task.Start 2                 ' 連続実行
g_task.Stop 3                  ' サイクル停止
```

4.2.3. サンプルプログラム

サンプルプログラムは，デフォルトワークスペースを利用し，タスク"PRO01"の制御（連続実行とサイクル停止）を行います．

List 4-2 Task.frm

```
Dim g_eng As CaoEngine
Dim g_ctrl As CaoController
Dim g_task As CaoTask

Private Sub Command1_Click()
    ' タスクの開始
    g_task.Start 2
End Sub

Private Sub Command2_Click()
    ' タスクの停止
    g_task.Stop 3
End Sub

Private Sub Form_Load()
    Set g_eng = New CaoEngine

    ' 接続処理　IP は各コントローラの設定にしてください
    Set g_ctrl = g_eng.Workspaces(0).AddController("RC8", "caoProv.DENSO.RC8", "",
"Server=192.168.0.1")

    ' タスク名"PRO1"
    Set g_task = g_ctrl.AddTask("PRO1", "")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    g_ctrl.Tasks.Clear
    Set g_task = Nothing

    g_eng.Workspaces(0).Controllers.Remove g_ctrl.Index
    Set g_ctrl = Nothing

    Set g_eng = Nothing
End Sub
```

4.3. RC8 コントローラのロボット制御

ロボット制御を行うには，コントローラで自動モードにしておく必要があります．

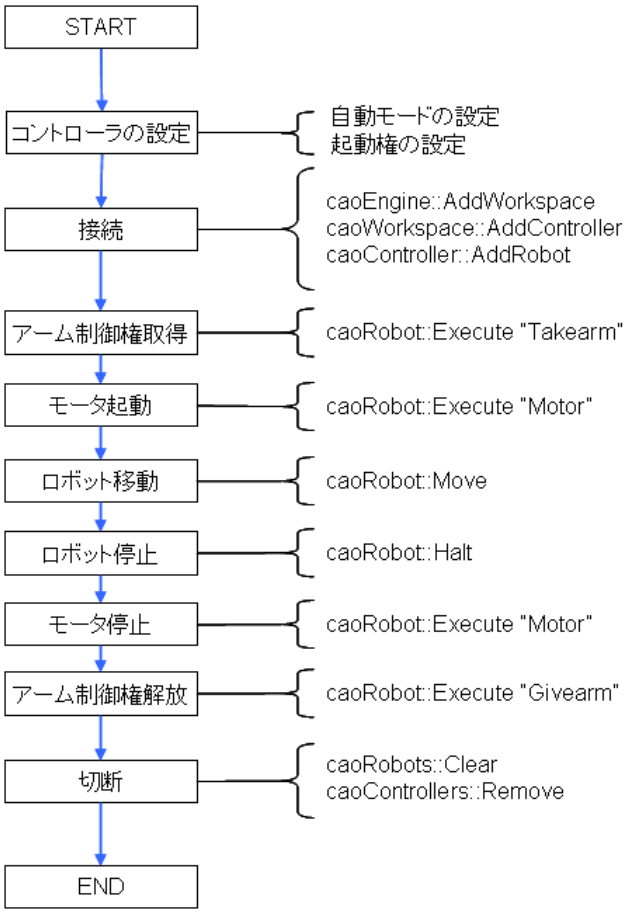


図 4-5 ロボット制御の流れ

4.3.1. 接続

CaoController オブジェクトを生成するまでの手順は「4.1.1 接続」を参照してください．ロボットを動作させるには CaoRobot オブジェクトを生成します．以下にコード例を示します．

```
Dim g_robot as CaoRobot                         '   CaoRobot オブジェクトを格納する変数
Set g_robot = g_ctrl.AddRobot("Arm", "")
```

4.3.2. アーム制御権の取得と解放

ロボット制御を行うには，ロボットのアーム制御権を取得する必要があります．また，コントローラと切断する前にロボットのアーム制御権を解放する必要があります．以下にコード例を示します．

```
g_robot.Execute "Takearm"                    '  アーム制御権の取得
  ':
g_robot.Execute "Givearm"                    '  アーム制御権の解放
```

4.3.3. モータの起動と停止

ロボット制御を行うには，ロボットのモータが起動している必要があります．以下に RC8 プロバイダを通してモータの起動と停止を行うコード例を示します．詳しくは「5.2.27.25CaoRobot::Execute("Motor") コマンド」を参照して下さい．

```
g_robot.Execute "Motor", Array(1, 0)    '  モータの起動
  ':
g_robot.Execute "Motor", Array(0, 0)    '  モータの停止
```

4.3.4. ロボットの移動と停止

ロボットを動作させるには CaoRobot::Move メソッドを用います．Move の詳細は「5.2.24CaoRobot::Move メソッド」を参照してください．Move 時に NEXT オプションをつけることによって，CaoRobot::Halt メソッドでロボットの動作を停止させることができます．

```
g_robot.Move  1,"P(400, 300, 200, 180, 0, 180, 5)","Next" '  ロボットの動作
  ':
g_robot.Halt                                                '  ロボットの停止
```

4.3.5. サンプルプログラム

サンプルプログラムは，デフォルトワークスペースを利用し，ロボットを P10（P 型の 10 番目）に格納された場所へ移動させた後，P11（P 型の 11 番目）に格納された場所へ移動させます．Move 時に NEXT オプションをつけることによって，CaoRobot::Halt メソッドでロボットの動作を停止させることができます．

List 4-3 Robot.frm

```
Dim g_eng As CaoEngine
Dim g_ctrl As CaoController
Dim g_robot As CaoRobot
Dim g_robotVar As CaoVariable
Dim g_haltFlag As Boolean

Private Sub Command1_Click()
  ' アーム停止状態ならばモータ起動
  If g_robotVar.Value = False Then
    g_robot.Execute "Motor", Array(1, 0)
  End If
```

```
End Sub

Private Sub Command2_Click()
  ' アーム停止状態ならばモータ停止
  If g_robotVar.Value = False Then
     g_robot.Execute "Motor", Array(0, 0)
  End If
End Sub

Private Sub Command3_Click()
' ロボット停止
    g_robot.Halt

' ロボット停止が実行されたことを記録
    g_haltFlag = True
End Sub

Private Sub Command4_Click()
    ' アーム動作状態の場合，新しく動作命令を実行しない
    If g_robotVar.Value = True Then
       Exit Sub
    End If

    g_haltFlag = False

    ' ロボット動作
    g_robot.Move 1, "@P P10", "NEXT"

    ' 前動作が完了するまで次の動作に移行しない
    Do Until g_robotVar.Value = False
       DoEvents
    Loop

    ' ロボット停止が実行された場合，次の動作に移行しない
    If g_haltFlag = True Then
       Exit Sub
    End If

    ' ロボット動作
    g_robot.Move 1, "@P P11", "NEXT"
End Sub

Private Sub Form_Load()
    Set g_eng = New CaoEngine

' 接続処理　IP は各コントローラの設定にしてください
    Set g_ctrl = g_eng.Workspaces(0).AddController("RC8", "caoProv.DENSO.RC8", "",
"Server=192.168.0.1")

' CaoRobot オブジェクト生成
    Set g_robot = g_ctrl.AddRobot("Arm")

' アーム動作状態を確認する変数
    Set g_robotVar = g_robot.AddVariable("@BUSY_STATUS")

    ' アーム制御権の取得
    g_robot.Execute "Takearm"

    ' モータ起動
    Command1_Click
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' モータ停止
    Command2_Click
```

```
        ' アーム制御権の解放
        g_robot.Execute "Givearm"

        g_robot.Variables.Clear
        Set g_robotVar = Nothing
        g_ctrl.Robots.Clear
        Set g_robot = Nothing
        g_eng.Workspaces(0).Controllers.Remove g_ctrl.Index
        Set g_ctrl = Nothing
        Set g_eng = Nothing
    End Sub
```

5. コマンドリファレンス

5.1. コマンド一覧

表 5-1 コマンド一覧

カテゴリ	メソッド/プロパティ	機能	
CaoWorkspace			
	Addcontroller	コントローラに接続	P. 49
CaoController			
	AddFile	PacScript・システムファイル，フォルダに接続	P. 51
	AddRobot	ロボットに接続	P. 52
	AddTask	タスク(PacScript)に接続	P. 53
	AddVariable	ユーザ・システム変数に接続	P. 54
	AddExtension	拡張ボードに接続	P. 55
	get_Name	コントローラ名の取得	P. 56
	get_FileNames	ファイル名の一覧の取得	P. 56
	get_TaskNames	タスク(PacScript)名の一覧の取得	P. 56
	get_VariableNames	ユーザ・システム変数の一覧の取得	P. 57
	Execute	コントローラクラスに実装されているコマンドの実行	P. 57
CaoFile			
	AddFile	PacScript ファイルに接続	P. 82
	AddVariable	ファイルのシステム変数に接続	P. 83
	get_VariableNames	システム変数の一覧取得	P. 83
	get_FileNames	ファイル名一覧の取得	P. 83
	get_Size	ファイルのサイズの取得	P. 83
	get_Value	ファイルの内容の取得	P. 83
	put_Value	ファイルの内容の書き換え	P. 84
CaoRobot			
	Accelerate	内部加速度，内部減速度の設定	P. 84
	AddVariable	システム変数の接続	P. 85
	get_VariableNames	システム変数一覧の取得	P. 85
	Halt	ロボット非同期動作時の停止	P. 85
	Change	ロボットのツール座標系/ユーザ座標系の変更	P. 86
	Drive	サポートされていません.	P. 86
	Move	ロボットの動作	P. 87
	Rotate	軸回りの回転動作	P. 90
	Speed	内部移動速度の設定	P. 92
	Execute	ロボット特有の動作の実行	P. 93
CaoTask			
	AddVariable	システム変数の接続	P. 164
	get_VariableNames	システム変数の一覧取得	P. 166

5.2. メソッド・プロパティ

5.2.1. CaoWorkspace::AddController メソッド

RC8 プロバイダでは AddController メソッド実行時に渡されたパラメータを参照し，該当のコントローラと接続を行います.

このときオプション文字列で通信形態，接続パラメータ，タイムアウトの設定を指定します．オプションとオプションの区切りは","で行います.

書式 AddController(<bstrCtrlName:BSTR>,<bstrProvName:BSTR>,<bstrPcName:BSTR >
[,<bstrOption:BSTR>])

bstrCtrlName	:	[in]	コントローラ名 接続単位で重複しない任意の文字列を指定します. ※異なるアプリケーションや別 PC から同一の名前を指定した場合はエラー(0x80000205)になります. 空文字列(““)を指定した場合，Cao エンジンが自動的にユニークなコントローラ名を割り当てます.
bstrProvName	:	[in]	プロバイダ名．固定値 =“ CaoProv.DENSO.RC8".
bstrPcName	:	[in]	プロバイダの実行マシン名 同一マシン上の場合は空文字列(““)を指定します.
bstrOption	:	[in]	オプション文字列＝"<オプション 1>,<オプション 2>,…"

以下にオプション文字列に指定するリストを示します.

表 5-2 CaoWorkspace::AddController のオプション文字列

オプション	説明
Server=<IP アドレス>	接続対象の RC8 コントローラの IP アドレスを指定します. 例："Server=192.168.0.1"
Timeout=<時間>	通信タイムアウト時間を ms 単位で指定します. デフォルト値は 3000ms です. Server オプションを指定したときのみ有効です.
Interval=<時間>	接続しているコントローラからメッセージを取得する周期を ms 単位で指定します.

	デフォルト値は 100ms です. Server オプションを指定したときのみ有効です.
InvokeTimeout=<時間>	コマンド呼び出しタイムアウト時間を ms 単位で指定します. コマンド処理に要する時間がこの時間を超える場合はタイムアウトエラーになります. デフォルト値は 180000ms です. Server オプションを指定したときのみ有効です.
WPJ={<プロジェクトファイル>}	シミュレーションを行う場合, プロジェクトファイルは WINCAPS3 の*.wpj ファイルをフルパスで指定します. 既に起動している最も新しい仮想のコントローラに対して接続を行う場合はプロジェクトファイルに“*”を指定します. プロジェクトファイル名が既に起動している仮想のコントローラと同一の場合はそのコントローラに同時接続します. 省略時は”WPJ={*}”扱いになります. 例: “WPJ={C:¥ Program Files¥WINCAPS3¥Test¥Test.wpj}” “WPJ={*}”
Message=<イベント通知>	5.4 のイベントの通知を指定します. 通知する場合は True, 通知しない場合は False を指定します. 省略時は True 扱いになります. イベント通知を必要としない場合は False を指定することを推奨します.

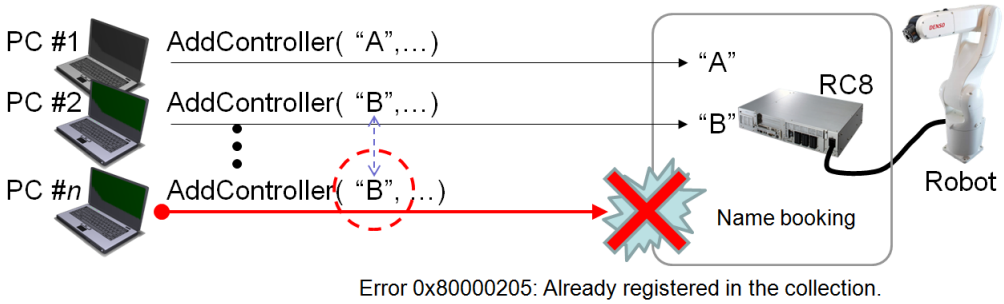
オプション文字列に"@IfNotMember"オプションを指定した場合はコントローラ名に対応する既存オブジェクトを返します. 同名のオブジェクトがない場合は, 指定されたコントローラ名のオブジェクトを新規に作成します. コントローラ名に空白文字列("")を指定した場合は@IfNotMember の指定は無視されます.

使用例 CaoController の作成

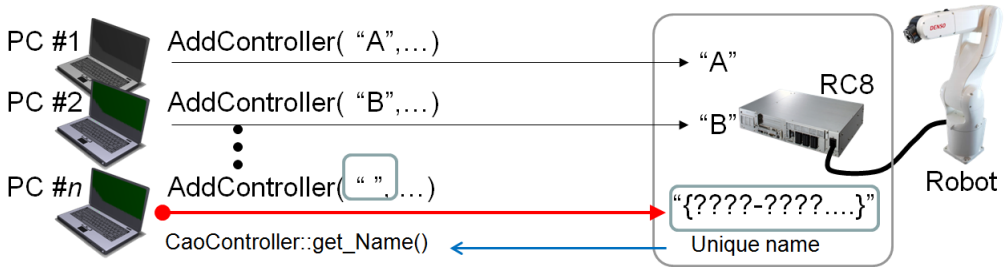
```
Private caoEng As CaoEngine      ' Engine オブジェクト
Private caoWs As CaoWorkspace    ' WorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト

Set caoEng = New CaoEngine
Set caoWS = caoEng.CaoWorkspaces.Item(0)
Set caoCtrl =
CaoWS.AddController( “rc8","CaoProv.DENSO.RC8"," ","Server=192.168.0.1,Timeout=1000")
```

5.2.1.1. 実機に対する複数接続の注意点

RC8 プロバイダによる実機接続では，接続単位（＝AddController 単位）で bCAP プロトコルによる通信がクライアント/サーバーの関係で行われます．このとき，クライアント(PC)が接続を確立するために呼び出す CaoWorkspace::AddController メソッドに指定する第一引数であるコントローラ名は仕様上，ユニークな値である必要がありますのでご注意ください．コントローラ名が重複数と接続が失敗します．



このコントローラ名は上記クライアント同士で重複のないように取り決めるなどして管理が必要ですが，これが行えない場合は，コントローラ名を“”（空文字列）にしてシステム側にユニークな名前を要求することで対応が可能です．この時，自動割り当てされた名前は CaoController::Name プロパティで取得が可能です．



5.2.2. CaoController::AddFile メソッド

CaoController クラスの AddFile メソッドの引数は，ファイル名(BSTR 型)を指定します．ここで指定する"ファイル名"は PacScript プログラム名あるいはシステム予約されているファイル名，またはディレクトリ名を指定します．この引数にファイルパスのみを指定してディレクトリを指定することもできます．また，パスを省略した場合は，デフォルトフォルダであるプロジェクトルート内のファイルであるとして扱われます．

以下に AddFile の仕様を示します．

書式 AddFile(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName	:	[in]	ファイル/ディレクトリ名
bstrOption	:	[in]	オプション文字列

ディレクトリ名を指定する場合はディレクトリ名の後ろに'¥'記号をつけます．

オプションは以下の文字列を使用します．

表 5-3 CaoController::AddFile のオプション文字列

オプション	意味
@Create[=<0～2>]	0:bstrName を作成しません. bstrName が存在しないときはエラーを返します. (デフォルト) 1:bstrName 作成します. すでに存在する場合は存在する bstrName を取得します. 2:bstrName を作成します. 指定した bstrName が存在するときはエラーを返します.

ファイルの一覧を下表に示します.

表 5-4: ファイルの実装状況一覧

	ORiN2ファイル名	形式	説明
1	*.PCS	text	PacScript ソース
2	*.H	text	PacScript ヘッダ
3	*.PNS	text	操作盤ソース

【注意】

CaoFile オブジェクトはファイルへの同時アクセスに対応していません.
必ずアプリケーション側でファイルへの排他制御を行うようにしてください.

使用例 Pro1.pcs ファイルの内容取得

```
Dim caoFl As CaoFile
Dim strText As String
Set caoFl = caoCtrl.AddFile("pro1.pcs"," " )    'pro1.pcs を指定
strText = caoFl.Value
```

5.2.3. CaoController::AddRobot メソッド

AddRobot メソッドを呼び出すと CaoRobot オブジェクトが取得できます. CaoController クラスの AddRobot メソッドの引数は, ロボット名(BSTR 型)を指定します. ここで指定する"ロボット名"は任意の文字列で特に何の制限もありません. 例えば, AddRobot ("Robot1")を指定することができます.

書式 AddRobot(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName	:	[in]	ロボット名
bstrOption	:	[in]	オプション文字列 ID=<ロボット番号> 省略時は ID=0 になります. 選択できる ID は以下になります.

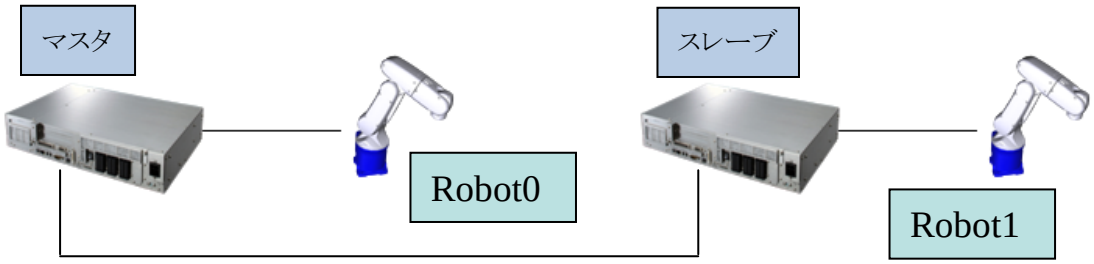
ID 番号	ロボットの種類	備考
0	マスタロボット	-
1	1 台目のスレーブロボット	協調機能時のみ使用

使用例

```
Dim CaoRob as CaoRobot
Set Rob = caoCtrl.AddRobot("Robot0","ID=0 " )    'Robot0 を指定
```

協調機能の場合の使用例

```
Dim MasterRobot as CaoRobot
Dim SlaveRobot as CaoRobot
Set MasterRobot = caoCtrl.AddRobot("Robot0","ID=0 " )    'Robot0 を指定
Set SlaveRobot = caoCtrl.AddRobot("Robot1","ID=1 " )   '協調機能でスレーブを使う場合に使用

MasterRobot.Move 1," J0"
SlaveRobot.Move 1," J1"
```



5.2.4. CaoController::AddTask メソッド

CaoController クラスの AddTask メソッドの引数は，タスク名(BSTR 型)を指定します．ここで指定する"タスク名"は PacScript プログラム名を指定します．例えば，AddTask ("pro1")といった表現で CaoTask オブジェクトが取得できます．

書式 AddTask(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName	:	[in]	タスク名
bstrOption	:	[in]	オプション文字列(未使用)

使用例

```
Dim caoTsk as CaoTask
Set caoTsk = caoCtrl.AddTask("Pro1","" )    'Pro1 を指定
```

5.2.5. CaoController::AddVariable メソッド

この CaoController クラスの AddVariable メソッドは，CAO の一般的意味では，変数にアクセスするためのメソッドです．RC8 プロバイダでは，変数名にはユーザ変数，システム変数のどちらでも指定することができます．

ユーザ変数の場合は，そのまま RC8 コントローラのグローバル変数(I,F,V,P,J,D,T,S)， I/O およびに対応します．

以下に，AddVariable の仕様を示します．

書式 AddVariable(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName : [in] 変数名 "<変数名>[<番号>]"

bstrOption : [in] オプション文字列 "<オプション>"

<変数名> : I，F，V，P，J，D，T，S または IO，IOB，IOW，IOD，IOF 文字はすべて半角指定(全角は無効)で大小の区別はありません．
IO は Bit，IOB は Byte，IOW は Word，IOD は Double Word (Long)，IOF は Float(Single)として I/O の値を処理します．

<番号> : 変数名で指す変数の番号 または ＊ または ＊_<数値>
番号は 10 進数で指定します．
＊を指定した場合，初期値 0 として扱われます．変数の番号は変数オブジェクトの ID プロパティで参照，変更することができます．
すべて半角指定(全角は無効) ．
＊_<数値>の数値は 10 進数で指定します．同種変数のワイルドカード(＊)指定を複数定義可能にするための識別番号で値に特別な意味はありません．

"["および"]"は省略できます．

(例 1) “i0","I[0]" ・・・ I 型変数の 0 番を指定

(例 2) “IO128","io[128]" ・・・ I/O 変数の 128 番を指定

(例 3) “I*","IO[*]" ・・・ ワールドカード指定

(例 4) “I*_1","I*_2" ,"I*_3" ・・・ 複数ワールドカード指定(I 型変数)

システム変数を指定するときは，変数名の最初に"@"をつけます．変数名の最初に"@"がないものは，すべてユーザ変数として扱われます．

RC8 プロバイダで実装されているシステム変数は「5.3 変数一覧」を参照して下さい．

使用例　I/O128 番へのアクセス

```
Dim caoVar as CaoVariable
Set caoVar = caoCtrl.AddVariable("IO128","" )    'I/O128 を指定
caoVar.value = 1
MsgBox caoVar.Value
```

```
Dim caoVar as CaoVariable
Set caoVar = caoCtrl.AddVariable("IO*","" )    'IO*を指定，インデックスは ID で指定する
caoVar.ID = 128
caoVar.value = 1
MsgBox caoVar.Value
```

5.2.6. CaoController::AddExtension メソッド

CaoController クラスの AddExtension メソッドの引数は，拡張機能名(BSTR 型)を指定します．

書式 <caoExt:CaoExtension オブジェクト> = AddExtension (<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName	:	[in]	拡張機能名
bstrOption	:	[in]	オプション文字列(未使用)
戻り値	:	[out]	CaoExtension クラスのオブジェクト

以下に指定可能な拡張機能名リストを示します．

表 5-5 CaoWorkspace:: AddExtension の拡張機能名一覧

拡張機能名	説明
Hand<n>[2]	電動ハンド<n>用オブジェクト[3]

使用例

```
Dim caoExt as CaoExtension
Set caoExt = caoCtrl.AddExtension( "Hand0" )    'Hand0 オブジェクトを取得
```

[2] <n>はボード番号(0～7)を表します．

[3]電動ハンド用オブジェクトを使用するには，オプションの電動ハンドコントロールボードを取り付け，初期設定を行う必要があります．
初期設定方法につきましては，【DENSO ROBOT USER MANUALS Controller Model:RC8 Series】の【電動ハンド使用設定】を参照して下さい．

5.2.7. CaoController::get_Name プロパティ

CaoWorkspace クラスの AddController メソッドで指定されたコントローラ名を取得します.

使用例　自動的に割り当てられたコントローラ名を表示

```
Private caoEng As CaoEngine      ' Engine オブジェクト
Private caoWs As CaoWorkspace    ' WorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト

Set caoEng = New CaoEngine
Set caoWS = caoEng.CaoWorkspaces.Item(0)
Set caoCtrl = CaoWS.AddController( " ","CaoProv.DENSO.RC8"," ","Server=192.168.0.1")

Debug.Print caoCtrl. Name
```

5.2.8. CaoController::get_FileNames プロパティ

AddFile メソッドで指定できるファイル名の一覧を取得します.

使用例　ルートフォルダ以下のファイル名を列挙

```
Dim ln%, lb%, ub%
Dim var As variant

var = caoCtrl.FileNames

lb = LBound( var )
ub = UBound( var )
For ln = lb To ub
        Debug.Print Str( ln ) &"=" & var( ln )
Next
```

5.2.9. CaoController::get_TaskNames プロパティ

AddTask メソッドで指定できるタスク名の一覧を取得します.

使用例　タスク名を列挙

```
Dim ln%, lb%, ub%
Dim var As variant

var = caoCtrl.TaskNames

lb = LBound( var )
ub = UBound( var )
For ln = lb To ub
        Debug.print Str( ln ) &"=" & var( ln )
Next
```

5.2.10. CaoController::get_VariableNames プロパティ

AddVariable メソッドで指定できる変数名とシステム変数名の一覧を取得します.

5.2.11. CaoController::Execute メソッド

CaoController クラスに属するプロバイダ固有の拡張コマンドを実行します.

Execute メソッドの引数は, コマンドを BSTR, パラメータを VARIANT 配列で指定します.

書式 [<vntRet:VARIANT> =] Execute(<bstrCmd:BSTR > [,<vntParam:VARIANT>])

bstrCmd	:	[in]	コマンド名
vntParam	:	[in]	パラメータ
vntRet		[out]	返り値

実行時バインディングの機能を使って本クラスに定義していないメソッドを呼び出した場合, 次の仕様で Execute メソッドが自動的に呼ばれます.

vntRet = Obj.CommandName(Param1, Param2, ...)

↓

vntRet = Obj.Execute("CommandName", Array(Param1, Param2, ...))

1.第一引数にコマンド名が BSTR 文字列で渡る

2.第二引数に全パラメータが VARIANT 配列で渡る

使用例

```
Dim vRes as Variant
vRes = caoCtrl.Execute("ClearError" )   'コントローラのエラークリア
vRes = caoCtrl.ClearError()
```

現在指定可能なコマンドの一覧を示します.

表 5-6 CaoController::Execute メソッドのコマンド一覧

カテゴリ	コマンド名	機能	対応 Version	
エラー処理				
	ClearError	エラーリセット	1.0.0	P.58
	GetErrorDescription	エラー文字列の取得	1.0.0	P.58
	GetCurErrorCount	現在発生しているエラー数の取得	1.0.0	P.71
	GetCurErrorInfo	現在発生しているエラー情報の取得	1.0.0	P.72
タスク制御				

	KillAll	全タスク停止	1.0.0	P.59
	SuspendAll	全タスク瞬時停止	1.0.0	P.60
	StepStopAll	全タスクステップ停止	1.0.0	P.61
	ContinueStartAll	全タスクコンティニュスタート	1.0.0	P.61
	KillAllTsr	実行中の全特権タスクを初期化停止	1.0.0	P.59
	RunAllTsr	モードで指定された特権タスクの起動	1.0.0	P.60
ログ				
	GetErrorLogCount	エラーログ数の取得	1.0.0	P.62
	GetErrorLog	エラーログの取得	1.0.0	P.62
	GetOprLogCount	操作ログ数の取得	1.0.0	P.63
	GetOprLog	操作ログの取得	1.0.0	P.64
変数アクセス				
	GetPublicValue	Public 変数読み込み	1.7.14	P.65
	SetPublicValue	Public 変数書き込み	1.7.14	P.67
その他				
	SysState	コントローラのステータスの取得	1.4.2	P.69
	SysInfo	コントローラのシステム情報を取得	1.7.11	P.69
	SetAllDummyIO	IO を疑似化	1.6.5	P.71

5.2.11.1. CaoController::Execute("ClearError") コマンド

コントローラで発生中のエラーをクリアします.

書式　ClearError ()

引数　：　なし
戻り値　：　なし

使用例

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

5.2.11.2. CaoController::Execute("GetErrorDescription") コマンド

指定エラーコードのエラー内容を取得します.

書式　GetErrorDescription (<lErrCode>)

<lErrCode>　：　[in] エラーコード (VT_I4)
戻り値　：　エラー内容 (VT_BSTR)

使用例

```
Dim strDescription As String
strDescription = caoCtrl.Execute("GetErrorDescription" , &H83500003 )
```

5.2.11.3. CaoController::Execute("KillAll") コマンド

実行中の全タスクを初期化停止します.

書式　KillAll ([<lSync>])

引数　：[in] 同期フラグ(VT_I4)
0:ロボット, プログラムの停止を待たずに処理を抜けます.
1:ロボット, プログラムの停止を待って処理を抜けます.
省略時は0です.

戻り値　：なし

全タスクが初期化停止状態になります.

ただし, 特権タスクは停止しません.

使用例

```
caoCtrl.Execute"KillAll"
```

5.2.11.4. CaoController::Execute("KillAllTsr") コマンド

実行中の全特権タスクを初期化停止します.

書式　KillAllTsr ([<lSync>])

引数　：[in] 同期フラグ(VT_I4)
0:特権タスクの停止を待たずに処理を抜けます.
1:特権タスクの停止を待って処理を抜けます.
省略時は0です.

戻り値　：なし

使用例

```
caoCtrl.Execute"KillAllTsr"
```

5.2.11.5. CaoController::Execute("RunAllTsr") コマンド

モードで指定された特権タスクを起動します.

書式 RunAllTsr ([<Mode>])

引数 : [in] モード(VT_I4)
0:ルートにある特権タスクを起動します.
1:全特権タスクを起動します.
省略時は0です.

戻り値 : なし

使用例

```
caoCtrl.Execute"RunAllTsr"
```

5.2.11.6. CaoController::Execute("SuspendAll") コマンド

実行中の全タスクを一時停止します.

書式 SuspendAll ([<lSync>])

引数 : [in] 同期フラグ(VT_I4)
0:ロボット, プログラムの停止を待たずに処理を抜けます.
1:ロボット, プログラムの停止を待って処理を抜けます.
省略時は 0 です.

戻り値 : なし

全タスクが一時停止状態になります.
ただし, 特権タスクは停止しません.

使用例

```
caoCtrl.Execute"SuspendAll"
```

5.2.11.7. CaoController::Execute("StepStopAll") コマンド

実行中の全タスクをステップ停止します.

書式 StepStopAll ([<lSync>])

引数 : [in] 同期フラグ(VT_I4)
0:ロボット, プログラムの停止を待たずに処理を抜けます.
1:ロボット, プログラムの停止を待って処理を抜けます.
省略時は 0 です.

戻り値 : なし

特権タスクは停止しません.

同期フラグが1の時, ステップ停止する前にプログラムが停止した場合も処理を抜けます.

使用例

```
caoCtrl.Execute"StepStopAll"
```

5.2.11.8. CaoController::Execute("ContinueStartAll") コマンド

一時停止中の全タスクを実行開始します.

書式 ContinueStartAll ( )

引数 : なし

戻り値 : なし

使用例

```
caoCtrl.Execute"ContinueStartAll"
```

5.2.11.9. CaoController::Execute("GetErrorLogCount") コマンド

エラーログの個数を取得します.

書式　GetErrorLogCount ( )

引数　：　なし
戻り値　：　個数 (VT_I4)

使用例

```
Dim lErrCnt As Long
lErrCnt = caoCtrl.Execute("GetErrorLogCount")
```

5.2.11.10. CaoController::Execute("GetErrorLog") コマンド

エラーログの情報を取得します.

書式　GetErrorLog (<lIndex>)

<lIndex>　：　エラーのインデックス番号(VT_I4) 0～<個数-1>
戻り値　：　エラー情報 (VT_VARIANT[VT_VARIANT|VT_ARRAY:15 要素])
[0]:エラーコード (VT_I4)
[1]:年 (VT_I4)
[2]:月 (VT_I4)
[3]:曜日 (VT_I4)
[4]:日 (VT_I4)
[5]:時 (VT_I4)
[6]:分(VT_I4)
[7]:秒 (VT_I4)
[8]:ミリ秒 (VT_I4)
[9]: コントローラ電源立上げからのカウントms (VT_I4)
[10]:エラー発生プログラム名 (VT_BSTR)
[11]:エラー発生行 (VT_I4)
[12]:エラーメッセージ (VT_BSTR)
[13]:オリジナルエラーコード (VT_I4)
[14]:呼び出し元(VT_I4)
-1:未定
0:システム

1:TP

2:IO

3:PC

4:PAC

5:COM2

6:COM3

7:COM4

使用例

```
Dim lErrCnt As Long
lErrCnt = caoCtrl.Execute("GetErrorLogCount")
Dim i As Long
For i=0 To lErrCnt-1
  vDat = caoCtrl.Execute("GetErrorLog", i)
   ' Hex(vDat(0)) エラーコード
   ' vDat(12) エラーメッセージ
   ' :
Next
```

5.2.11.11. CaoController::Execute("GetOprLogCount") コマンド

操作ログの個数を取得します.

書式　GetOprLogCount ( )

引数　：　なし

戻り値　：　個数 (VT_I4)

使用例

```
Dim lOprCnt As Long
lOprCnt = caoCtrl.Execute("GetOprLogCount")
```

5.2.11.12. CaoController::Execute("GetOprLog") コマンド

操作ログの情報を取得します.

書式　GetOprLog (<lIndex>)

<lIndex> : 操作ログのインデックス番号(VT_I4) 0～<個数-1>

戻り値 : 操作情報 (VT_VARIANT[VT_VARIANT|VT_ARRAY:12 要素])

[0]:操作コード (VT_I4)

[1]:年 (VT_I4)

[2]:月 (VT_I4)

[3]:曜日 (VT_I4)

[4]:日 (VT_I4)

[5]:時 (VT_I4)

[6]:分(VT_I4)

[7]:秒 (VT_I4)

[8]:ミリ秒 (VT_I4)

[9]: コントローラ電源立上げからのカウントms (VT_I4)

[10]:呼び出し元 (VT_I4)

-1:未定

0:システム

1:TP

2:IO

3:PC

4:PAC

5:COM2

6:COM3

7:COM4

[11]:操作内容 (VT_BSTR)

使用例

```
Dim lOprCnt As Long
lOprCnt = caoCtrl.Execute("GetOprLogCount")
Dim i As Long
For i=0 To lOprCnt-1
  vDat = caoCtrl.Execute("GetOprLog", i)
   ' Hex(vDat(0)) 操作コード
   ' vDat(11) 操作内容
   ' :
Next
```

5.2.11.13. CaoController::Execute("GetPublicValue") コマンド

Public 変数の値を読み込みます.

書式　GetPublicValue (<bstrTask> , <bstrName>[, <lIndex1>, <lIndex2>, ・・・])

<bstrTask> : タスク名 (VT_BSTR)
<bstrName> : 変数名 (VT_BSTR)
<lIndex(n)> : 次元毎のインデックス番号 (VT_I4)
<lIndex(n+1)> Public pbIArrays(1, 2, 3) As Long
・・・ とあった場合に
(bstrTask, bstrName, 1, 2, 2)
と指定することで
pbIArrays(1, 2, 2)
の一要素を読み込むことができます
対象が一次元配列で省略時は配列の一括読み込みとなります
戻り値 : Public 変数の値

指定タスク内の Public 変数を読み込みます.

Public 変数が配列の場合, 次元毎のインデックス番号を指定することにより指定の配列一つの値を読み込みます.

Public 変数が一次元配列でインデックス番号を省略した場合, 配列の全要素を一括して読み込むことができます.

V 型・・・要素数 3 個の F 型(VT_R4)の配列として読み込み
[0]X, [1]Y, [2]Z

P 型・・・要素数 7 個の F 型(VT_R4)の配列として読み込み
[0]X, [1]Y, [2]Z, [3]Rx, [4]Ry, [5]Rz, [6]Fig

J 型・・・要素数 8 個の F 型(VT_R4)の配列として読み込み
[0]J1, [1]J2, [2]J3, [3]J4, [4]J5, [5]J6, [6]J7, [7]J8

T 型・・・要素数 10 個の F 型(VT_R4)の配列として読み込み
[0]X, [1]Y, [2]Z, [3]Ox, [4]Oy, [5]Oz, [6]Ax, [7]Ay, [8]Az, [9]Fig

[注意事項]

二次元配列以上の Public 変数の一括読み込みは非対応です.

多次元配列に対して一括読み込みした場合, 一次元配列の要素のみ一括読み込みとなります.

V・P・J・T 型は対象の Public 変数がスカラー変数の場合のみ読み込みが可能です.

使用例

PacScript – Pro1.pcs

```
Public pbIValue As Long
Public pbIArrays(1, 2, 3) As Long
Public pbIArray(2) As Long
Public pbPValue As Position

Sub Main

End Sub
```

VisualBasic – 変数読み込み

```
Dim vParam As Variant
Dim iVal As Long

vParam = Array("Pro1", "pbIValue")
iVal = caoCtrl.Execute("GetPublicValue", vParam)
```

VisualBasic – 配列一要素読み込み

```
Dim vParam As Variant
Dim iVal As Long

vParam = Array("Pro1", "pbIArrays", 1, 2, 2)
iVal = caoCtrl.Execute("GetPublicValue", vParam)
```

VisualBasic – 一次元配列一括読み込み

```
Dim vParam As Variant
Dim vArray As Variant

vParam = Array("Pro1", "pbIArray")
vArray = caoCtrl.Execute("GetPublicValue", vParam)
```

VisualBasic – P 型変数読み込み

```
Dim vParam As Variant
Dim vVal As Variant

vParam = Array("Pro1", "pbPValue")
vVal = caoCtrl.Execute("GetPublicValue", vParam)
```

5.2.11.14. CaoController::Execute("SetPublicValue") コマンド

Public 変数の値を書き込みます.

書式　SetPublicValue (<vValue>, <bstrTask> , <bstrName>[, <lIndex1>, <lIndex2>, ・・・])

<vValue>	:	書き込む Public 変数の値 (任意)
<bstrTask>	:	タスク名 (VT_BSTR)
<bstrName>	:	変数名 (VT_BSTR)
<lIndex(n)>	:	次元毎のインデックス番号 (VT_I4)
<lIndex(n+1)>		Public pbIArrays(1, 2, 3) As Long
・・・		とあった場合に
		(vValue, bstrTask, bstrName, 1, 2, 2)
		と指定することで
		pbIArrays(1, 2, 2)
		の一要素を書き込むことができます
		対象が一次元配列で省略時は配列の一括書き込みとなります
戻り値	:	なし

指定タスク内の Public 変数を書き込みます.

Public 変数が配列の場合, 次元毎のインデックス番号を指定することにより指定の配列一つの値を書き込みます.

Public 変数が一次元配列でインデックス番号を省略した場合, 配列の全要素を一括して書き込むことができます.

V 型・・・要素数 3 個の F 型(VT_R4)の配列として書き込み
[0]X, [1]Y, [2]Z

P 型・・・要素数 7 個の F 型(VT_R4)の配列として書き込み
[0]X, [1]Y, [2]Z, [3]Rx, [4]Ry, [5]Rz, [6]Fig

J 型・・・要素数 8 個の F 型(VT_R4)の配列として書き込み
[0]J1, [1]J2, [2]J3, [3]J4, [4]J5, [5]J6, [6]J7, [7]J8

T 型・・・要素数 10 個の F 型(VT_R4)の配列として書き込み
[0]X, [1]Y, [2]Z, [3]Ox, [4]Oy, [5]Oz, [6]Ax, [7]Ay, [8]Az, [9]Fig

[注意事項]

二次元配列以上の Public 変数の一括書き込みは非対応です.

多次元配列に対して一括書き込みした場合, エラーとなります.

Public 変数の一次元配列の一括書き込みは要素数が同じかつ型が同じ場合のみ可能です.

V・P・J・T 型は対象の Public 変数がスカラー変数の場合のみ書き込みが可能です.

使用例

PacScript – Pro1.pcs

```
Public pbIValue As Long
Public pbIArrays(1, 2, 3) As Long
Public pbIArray(2) As Long
Public pbPValue As Position

Sub Main

End Sub
```

VisualBasic – 変数書き込み

```
Dim iVal As Long
Dim vParam As Variant

iVal = 1234
vParam = Array(iVal, "Pro1", "pbIValue")
caoCtrl.Execute "SetPublicValue", vParam
```

VisualBasic – 配列一要素書き込み

```
Dim iVal As Long
Dim vParam As Variant

iVal = 1234
vParam = Array(iVal, "Pro1", "pbIArrays", 1, 2, 2)
caoCtrl.Execute "SetPublicValue", vParam
```

VisualBasic – 一次元配列一括書き込み

```
Dim i As Long
Dim iArray(2) As Long
Dim vParam As Variant

For i = 0 To 2
    iArray(i) = i
Next i
vParam = Array(iArray, "Pro1", "pbIArray")
caoCtrl.Execute "SetPublicValue", vParam
```

VisualBasic – P 型変数書き込み

```
Dim fVals(6) As Single
Dim vParam As Variant

fVals(0) = 1.0
fVals(1) = 2.0
fVals(2) = 3.0
fVals(3) = 1.0
fVals(4) = 2.0
```

```
fVals(5) = 3.0
fVals(6) = -1
vParam = Array(fVals, "Pro1", "pbPValue")
caoCtrl.Execute "SetPublicValue", vParam
```

5.2.11.15. CaoController::Execute("SysState") コマンド

コントローラのステータスを返します.

PacScript 言語の SysState 命令に対応します.

書式　SysState ( )

引数　：　なし

戻り値　：　[out] コントローラのステータス　[VT_I4]

使用例

```
Dim state as long
State = caoCtrl.Execute("SysState")
```

5.2.11.16. CaoController::Execute("SysInfo") コマンド

コントローラのシステム情報を返します.

PacScript 言語の SysInfo 命令に対応します.

書式　SysInfo(<lIndex>)

<lIndex>　：　インデックス番号(VT_I4)

戻り値　：　[out] コントローラのシステム情報

インデックス番号	システム情報	データ型
0	製造番号(コントローラのシリアル番号)	文字列
1	コントローラ内臓のネットワークカードの MAC アドレス	文字列
2	ペンダント接続状態 0:未接続 1:ティーチングペンダントが接続されている 2:ミニペンダントが接続されている	整数型
3	グローバル変数個別情報 配列の 0～7 の各要素は下に示す変数の個数を示す 0:I 型 1:F 型	整数型配列

	2:D 型 3:V 型 4:P 型 5:J 型 6:T 型 7:S 型	
4	CPU 情報 0:不明(仮想環境) 2:ATOM 5:i7	整数型
5	ログインユーザレベル 1000:オペレータ 2000:プログラマ 3000:メンテナ 3001 以上:システム予約	整数型
6	特権タスク動作中 -1:動作中 0:動作中でない	整数型
7	操作盤表示中 -1:表示中 0:表示中でない	整数型
8	総通電時間(分)	整数型
9	総稼働時間(分)	整数型
10	累積通電時間(分)	整数型
11	累積稼働時間(分)	整数型
12	電源入り通電時間(分)	整数型
13	電源入り稼働時間(分)	整数型
14	モータ ON 回数	整数型
15	エンコーダバッテリ点検日 -1:点検日を超えている 0:点検日を超えていない	整数型
16	コントローラバッテリ点検日 -1:点検日を超えている 0:点検日を超えていない	整数型
17	ロボットの接続台数	整数型

使用例

```
Dim sysinfo as long
sysinfo = caoCtrl.Execute("SysInfo",0)
```

5.2.11.17. CaoController::Execute("SetAllDummyIO") コマンド

IO を疑似化します.

書式　SetAllDummyIO (<lMode>)

<lMode>　：　番号(VT_I4)
戻り値　：　なし

インデックス番号	ロボット情報
0	全疑似化解除
1	汎用入力疑似化
2	専用入力疑似化
3	汎用入力, 専用入力とも疑似化

使用例

```
g_caoCtrl.Execute "SetAllDummyIO", 0  '全擬似化解除
```

5.2.11.18. CaoController::Execute("GetCurErrorCount") コマンド

現在発生しているエラーの個数を返します. エラークリアすると個数は 0 になります.

書式　GetCurErrorCount( )

引数　：　なし
戻り値　：　[out] 発生しているエラーの個数　[VT_I4]

使用例

```
Dim ErrCount as long
ErrCount = caoCtrl.Execute("GetCurErrorCount")
```

5.2.11.19. CaoController::Execute("GetCurErrorInfo") コマンド

現在発生しているエラーの情報を返します.

Index は 0 が最新のエラーになります.

書式　GetCurErrorInfo(<lIndex>)

<lIndex>　:　インデックス番号(VT_I4)

戻り値　:　[out] 現在発生しているエラーの情報

1:エラーコード (VT_I4)

2:エラーメッセージ (VT_BSTR)

3:サブコード (VT_I4)

4:ファイル ID+行番号 (VT_I4)

5:プログラム名 (VT_BSTR)

6:行番号 (VT_I4)

7:ファイル ID (VT_I4)

戻り値の 4 は(((ファイル ID << 20) & 0xFFF00000) | (行番号 & 0x000FFFFF))です.

使用例

```
Dim Errinfo as Variant
Errinfo = caoCtrl.Execute("GetCurErrorInfo",0)
```

5.2.12. CaoController::Execute メソッド(COBOTTA 専用)

CaoController クラスに属するプロバイダ固有の拡張コマンドを実行します.

現在指定可能なコマンドの一覧を示します.

表 5-7 CaoController::Execute メソッド(COBOTTA 専用)のコマンド一覧

カテゴリ	コマンド名	機能	対応 Version	
ロボット 状態・値取得				
	GetCCSConnection	無線タブレットの接続状態を取得	2.8.0	P.73
	GetDiretMode	ダイレクトモードの状態を取得	2.8.0	P.74
ロボット その他				
	ManualResetPreparation	安全に関わるコマンド(動作準備, エラー解除)の送信確認を行います	2.8.0	P.74
電動ハンド				

動作				
	HandChuck	ワークの把持動作を行う	2.8.0	P.75
	HandUnChuck	把持状態から所定の位置まで移動	2.8.0	P.75
	HandMoveA	フィンガを絶対位置に移動	2.8.0	P.76
	HandMoveR	フィンガを相対位置に移動	2.8.0	P.76
	HandMoveH	定速移動で把持動作を行う	2.8.0	P.76
	HandMoveAH	加減速付き絶対位置把持動作を行う	2.8.0	P.77
	HandMoveRH	加減速付き相対位置把持動作を行う	2.8.0	P.77
	HandMoveZH	ゾーン付き定速移動の把持動作を行う	2.8.0	P.78
	HandStop	ハンド動作を停止	2.8.0	P.78
	HandMoveVH	吸引もしくは送風動作を開始	2.8.0	P.81
電動ハンド 状態・値取得				
	HandBusyState	ハンドの動作状態を取得	2.8.0	P.79
	HandHoldState	ハンドの把持状態を取得	2.8.0	P.79
	HandInposState	ハンドのフィンガが目標位置に入っているかどうかを取得	2.8.0	P.80
	HandZonState	ハンドのフィンガが，設定された範囲内に位置しているかどうかの状態を取得	2.8.0	P.80
	HandCurPos	ハンドのフィンガの現在位置を取得	2.8.0	P.80
	HandCurPressure	バキュームの圧力を取得	2.8.0	P.81
	HandCurLoad	ハンドの現在の負荷率を取得	2.8.0	P.82
	HandSetDetectThreshold	電動バキュームの吸着検出に利用する閾値を設定	2.8.0	P.82

5.2.12.1. CaoController::Execute(″GetCCSConnection″) コマンド

無線タブレットの接続状態を返します.

書式　GetCCSConnection( )

引数　：　なし

戻り値　：　[out] 無線タブレットの接続状態 (VT_I4)

1:無線タブレット接続中

0:無線タブレットが接続されていない

使用例

```
Dim lVal As Long
lVal = caoCtrl.Execute("GetCCSConnection")
```

5.2.12.2. CaoController::Execute("GetDirectMode") コマンド

ダイレクトモードの状態を取得します.

書式　GetDirectMode()

引数　：なし
戻り値　：[out] ダイレクトモードの状態
0. ダイレクトモード以外
2. ダイレクトモード

使用例

```
Dim lMode As Long
lMode = caoCtrl.Execute("GetDirectMode")
```

5.2.12.3. CaoController::Execute("ManualResetPreparation") コマンド

RC8 プロバイダ以外から COBOTTA を制御する場合(b-CAP から直接 COBOTTA を制御する. など), 安全に関わるコマンド(動作準備, エラーリセット)を送信する前に, そのコマンドを意図して送信していることを確認するためのコマンドです.
このコマンドを事前に送信しないと「83500372」のエラーが発生します.

表 5-8 ManualResetPreparation を事前に送信する必要があるコマンド一覧

コマンド	機能	
MotionPreparation	動作準備を自動で実行します	P. 163
ClearError	エラーリセット	P. 58
@ERROR_CODE	エラーコードを取得, 設定します 0 を設定(エラーリセット)する場合に事前に送信する必要があります	P. 185

書式　ManualResetPreparation

引数　：なし
戻り値　：なし

使用例

```
'非常停止, 防護停止解除する
```

```
'エラークリア前の送信確認
caoRobot.Execute "ManualResetPreparation"
'エラークリア
caoCtrl.Execute "ClearError"

'動作準備未完了なら動作準備をする
  Dim vbState
vbState = caoRobot.Execute("GetMotionPreparationState")
If vbState <> True Then
    '動作準備前の送信確認
    caoRobot.Execute "ManualResetPreparation"
    caoRobot.Execute "MotionPreparation"
End If
```

5.2.12.4. CaoController::Execute("HandChuck") コマンド

COBOTTA ハンド用のコマンドです. 電動グリッパでは指定したポイントデータの設定に従い, ワークの把持動作をします. 電動バキュームでは指定したポイントデータの設定に従い, ワークを吸着します.

書式　HandChuck <lPointNo:LONG> [, < bstrNext:BSTR> or < bstrDetectOn:BSTR >]

<lPointNo>	:	[in] ポイント番号
<bstrNext:BSTR>	:	[in] NEXT:非同期実行オプション(電動グリッパ専用)
<bstrDetectOn:BSTR>	:	[in] DetectOn:把持検出オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute("HandChuck",Array(0, "Next"))
```

5.2.12.5. CaoController::Execute("HandUnChuck") コマンド

COBOTTA ハンド用のコマンドです. 指定したポイントデータのモードに従い動作します. 電動グリッパでは把持状態から所定の位置まで移動します. 電動バキュームでは送風及び停止動作が実行可能です.

書式　HandUnChuck <lPointNo:LONG> [, < bstrNext:BSTR>]

<lPointNo>	:	[in] ポイント番号
<bstrNext:BSTR>	:	[in] NEXT:非同期実行オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute "HandUnChuck", Array(1, "Next")
```

5.2.12.6. CaoController::Execute("HandMoveA") コマンド

COBOTTA ハンド用のコマンドです. フィンガを絶対位置に移動します. 電動グリッパ専用です.

書式 HandMoveA <dblPos:DOUBLE>, < sngSpeed:SINGLE > [, < bstrNext:BSTR>]

<dblPos> : [in]移動後の爪位置(mm)
<sngSpeed> : [in]目標速度(%)
<bstrNext:BSTR> : [in]NEXT:非同期実行オプション

使用例

```
caoCtrl.Execute "HandMoveA", Array(29.5, 100)
```

5.2.12.7. CaoController::Execute("HandMoveR") コマンド

COBOTTA ハンド用のコマンドです. フィンガを相対位置に移動します. 電動グリッパ専用です.

書式 HandMoveR <dblPos:DOUBLE>, < sngSpeed:SINGLE > [, < bstrNext:BSTR>]

<dblPos> : [in]現在爪位置からの移動量(mm)
<sngSpeed> : [in]目標速度(%)
<bstrNext:BSTR> : [in]NEXT:非同期実行オプション

使用例

```
caoCtrl.Execute "HandMoveR", Array(-5.5, 100)
```

5.2.12.8. CaoController::Execute("HandMoveH") コマンド

COBOTTA ハンド用のコマンドです. 定速移動で把持動作します. 電動グリッパ専用です.

書式 HandMoveH <dblForce:DOUBLE>, <bDirection:BOOL >
[, < bstrNext:BSTR> or < bstrDetectOn:BSTR >]

<dblForce> : [in]把持力(N)
<bDirection> : [in]動作方向
True 閉方向

False 開方向

<bstrNext:BSTR> : [in]NEXT:非同期実行オプション

<bstrDetectOn:BSTR> : [in] DetectOn:把持検出オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute "HandMoveH", Array(20, True, "Next")
caoCtrl.Execute "HandMoveA", Array(30, 100)
caoCtrl.Execute "HandMoveH", Array(20, True, "DetectOn")
```

5.2.12.9. CaoController::Execute("HandMoveAH") コマンド

COBOTTA ハンド用のコマンドです. 加減速付き絶対位置把持動作をします. 電動グリッパ専用です.

書式 HandMoveAH <dblPos:DOUBLE>, < sngSpeed:SINGLE >,
<dblForce:DOUBLE>
[, < bstrNext:BSTR> or < bstrDetectOn:BSTR >]

<dblPos> : [in]把持動作に移行する位置(mm)

<sngSpeed> : [in]目標速度(%)

<dblForce> : [in]把持力(N)

<bstrNext:BSTR> : [in]NEXT:非同期実行オプション

<bstrDetectOn:BSTR> : [in] DetectOn:把持検出オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute "HandMoveAH", Array(25, 100, 20, "Next")
```

5.2.12.10. CaoController::Execute("HandMoveRH") コマンド

COBOTTA ハンド用のコマンドです. 加減速付き相対位置把持動作をします. 電動グリッパ専用です.

書式 HandMoveRH <dblPos:DOUBLE>, < sngSpeed:SINGLE >,
<dblForce:DOUBLE>
[, < bstrNext:BSTR> or < bstrDetectOn:BSTR >]

<dblPos> : [in]現在爪位置からの把持動作に
移行する位置までの移動量(mm)

<sngSpeed> : [in]目標速度(%)

<dblForce> : [in]把持力(N)

<bstrNext:BSTR> : [in]NEXT:非同期実行オプション
<bstrDetectOn:BSTR> : [in] DetectOn:把持検出オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute "HandMoveAH", Array(-10, 100, 20, "Next")
```

5.2.12.11. CaoController::Execute("HandMoveZH") コマンド

COBOTTA ハンド用のコマンドです. ゾーン付き定速移動の把持動作をします. 電動グリッパ専用です.

書式 HandMoveZH <dblZonPos1:DOUBLE>, < dblZonPos2:DOUBLE >,
<dblForce:DOUBLE>, <bDirection:BOOL >
[, < bstrNext:BSTR> or < bstrDetectOn:BSTR >]

<dblZonPos1> : [in]ZON 範囲 1 [mm]
<dblZonPos2> : [in]ZON 範囲 2 [mm]
<dblForce> : [in]把持力 [N]
<bDirection> : [in]動作方向
True:閉方向
False:開方向
<bstrNext:BSTR> : [in]NEXT:非同期実行オプション
<bstrDetectOn:BSTR> : [in] DetectOn:把持検出オプション(電動グリッパ専用)

使用例

```
caoCtrl.Execute "HandMoveZH", Array(9.5, 10.5, 20, True, "Next")
```

5.2.12.12. CaoController::Execute("HandStop") コマンド

COBOTTA ハンド用のコマンドです. ハンド動作を停止します. 電動グリッパの動作中にこのコマンドを実行すると, ハンド動作がその場で減速停止します. 電動バキュームの動作中にこのコマンドを実行すると, 電動バキュームが吸引もしくは送風動作を停止します.

書式 HandStop

使用例

```
caoCtrl.Execute "HandStop"
```

5.2.12.13. CaoController::Execute("HandBusyState") コマンド

COBOTTA ハンド用のコマンドです. ハンドの動作状態を返します.

書式　HandBusyState

戻り値　:　[out] 動作状態(VT_BOOL)
True:動作中
False:動作中でない. コマンド受付可

使用例

```
Dim vntBusy as Variant
vntBusy = caoCtrl.Execute("HandBusyState")
If vntBusy = True Then
  caoCtrl.Execute "HandStop"
End If
```

5.2.12.14. CaoController::Execute("HandHoldState") コマンド

COBOTTA ハンド用のコマンドです. 電動グリッパの場合は把持状態を返します. 電動バキュームの場合はワークの吸着状態を返します.

書式　HandHoldState

戻り値　:　[out] 把持状態(VT_BOOL)
True:把持している
False:把持していない

使用例

```
Dim vntHold As Variant, vntBusy As Variant
vntHold = False
vntBusy = True

caoCtrl.Execute "HandMoveH", Array(20, True, "Next")
Do
  vntHold = caoCtrl.Execute("HandHoldState")
  vntBusy = caoCtrl.Execute("HandBusyState")
  Sleep(100)
Loop Until (vntHold = True) Or (vntBusy = False)

caoCtrl.Execute "HandStop"
```

5.2.12.15. CaoController::Execute("HandInposState") コマンド

COBOTTA ハンド用のコマンドです. 電動グリッパのフィンガが目標位置に入っているかどうかを返します. 電動グリッパ専用です.

書式　HandInposState

戻り値　:　[out] 到達状態(VT_BOOL)
True:目標位置に入っている
False:目標位置に入っていない

使用例

```
Dim vntInpos as Variant

caoCtrl.Execute "HandMoveA", Array(29.5, 100, "Next")
Do
  Sleep(100)
  vntInpos = caoCtrl.Execute("HandInposState")
Loop Until vntInpos = True
```

5.2.12.16. CaoController::Execute("HandZonState") コマンド

COBOTTA ハンド用のコマンドです. 電動グリッパのフィンガが, 設定された範囲内に位置しているかどうかの状態を返します. 電動グリッパ専用です.

書式　HandZonState

戻り値　:　[out] ZONE 状態(VT_BOOL)
True:範囲に入っている
False:範囲に入っていない

使用例

```
caoCtrl.Execute "HandMoveZH", Array(9.5, 10.5, 20, True)

Dim vntZon as Variant
vntZon = caoCtrl.Execute("HandZonState")
```

5.2.12.17. CaoController::Execute("HandCurPos") コマンド

COBOTTA ハンド用のコマンドです. フィンガの現在位置を返します. 電動グリッパ専用です.

書式　HandCurPos

戻り値 : [out] 現在位置 [mm](VT_R8)

使用例

```
Dim dblPos As Double
dblPos = caoCtrl.Execute("HandCurPos")
```

5.2.12.18. CaoController::Execute("HandMoveVH") コマンド

COBOTTA ハンド用のコマンドです. 吸引もしくは送風動作を開始します. 電動バキューム専用です.

書式 HandMoveVH(<sngPower :SINGLE>,<bDirection:BOOL>[, <dThreshold:DOUBLE>])

<sngPower > : [in]吸引力 [%] 40～100

<bDirection> : [in]動作方向
True:吸引(正方向回転)
False:送風(負方向回転)

<dThreshold> : [in]吸着検出閾値 [kPa]
オプション引数です(指定せずに実行可能).
ワークの吸着を検出する閾値を指定します. 吸着時は負圧が発生するため, 指定した閾値より圧力センサの値が小さいときにワークの吸着を検出します.
指定しない場合は以下の優先順位に従って検出閾値を設定します.
1. HandSetDetectThreshold コマンドで設定した閾値
2. ハンド画面で設定したデフォルト閾値(出荷時は 0 なので自動)

戻り値 : なし

使用例

```
caoCtrl.Execute "HandMoveVH", Array(100, True)
```

5.2.12.19. CaoController::Execute("HandCurPressure") コマンド

COBOTTA ハンド用のコマンドです. バキュームの圧力を返します. 電動バキューム専用です.

書式 HandCurPressure

戻り値 : [out] 現在の圧力 [kPa](VT_R8)

使用例

```
Dim dblPos As Double
dblPos = caoCtrl.Execute("HandCurPressure")
```

5.2.12.20. CaoController::Execute("HandCurLoad") コマンド

COBOTTA ハンド用のコマンドです. ハンドの現在の負荷率を取得します.

書式　HandCurLoad

戻り値　:　[out] 現在の負荷率 [%](VT_R8)

使用例

```
Dim dblPos As Double
dblPos = caoCtrl.Execute("HandCurLoad")
```

5.2.12.21. CaoController::Execute("HandSetDetectThreshold") コマンド

COBOTTA ハンド用のコマンドです. 電動バキュームの吸着検出に利用する閾値を設定します. 電動バキューム専用です.

書式　HandSetDetectThreshold <dThreshold:DOUBLE>

<dThreshold>　:　[in] 吸着検出閾値 [kPa]

戻り値　:　なし

使用例

```
caoCtrl.Execute "HandSetDetectThreshold", -10
```

5.2.13. CaoFile::AddFile メソッド

前述 5.2.2 と同様にファイルオブジェクトを作成します. 作成した CaoFile オブジェクトに対応しているファイルのパスは"<親オブジェクトのパス>/<AddFile で指定したファイル名>“となります.

使用例　User フォルダ以下の Pro1.pcs ファイルのサイズを表示

```
Dim caoFlDir As CaoFile
Set caoFlDir = caoCtrl.AddFile("User¥"," " )    'User フォルダを指定
```

```
Dim caoFl As CaoFile
Set caoFl = caoFlDir.AddFile("Pro1.pcs" )    'User¥Pro1.pcs ファイルを指定

Debug.Print caoFl.Size
```

5.2.14. CaoFile::AddVariable メソッド

CaoFile クラスの AddVariable メソッドの引数は, システム変数名を指定します.

実装されているシステム変数の一覧は表 5-19 を参照して下さい.

使用例 Pro1.pcs ファイルの CRC を取得

```
Dim caoFl As CaoFile
Set caoFl = caoCtrl.AddFile( "Pro1.pcs")

Dim caoCrc As CaoVariable
Set caoCrc = caoFl.AddVariable( "@CRC")

Debug.Print caoCrc.Value    ' Pro1.pcs の CRC を表示
```

5.2.15. CaoFile::get_VariableNames プロパティ

AddVariable メソッドで指定できる変数名とシステム変数名の一覧を取得します.

5.2.16. CaoFile::get_FileNames プロパティ

オブジェクトに対応しているパスがディレクトリのときのみ実行できます.

実行するとディレクトリ内のファイル名リストを取得します.

5.2.17. CaoFile::get_Size プロパティ

オブジェクトに対応しているファイルのファイルサイズを取得します.

使用例 Pro1.pcs ファイルのサイズを取得

```
Dim caoFl As CaoFile
Set caoFl = caoCtrl.AddFile( "Pro1.pcs")

Debug.Print caoFl.Size    ' Pro1.pcs のサイズを表示
```

5.2.18. CaoFile::get_Value プロパティ

オブジェクトに対応しているファイルの内容を取得します.

使用例 Pro1.pcs ファイルの内容を取得

```
Dim caoFl As CaoFile
Set caoFl = caoCtrl.AddFile( “Pro1.pcs”)

Debug.Print caoFl.Value    ‘ Pro1.pcs の内容を表示
```

5.2.19. CaoFile::put_Value プロパティ

オブジェクトに対応しているファイルの内容を書き換えます.

5.2.20. CaoRobot::Accelerate メソッド

ロボットの内部加速度, 内部減速度を指定します.

このメソッドは PacScript 言語の ACCEL,JACCEL 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

以下に, Accelerate の仕様を示します.

書式 Accelerate <lAxis:LONG >, <fAccel:FLOAT> [,<fDecel:FLOAT>]

lAxis	: [in]	軸番号	-1:手先加速度(ACCEL), 0:全軸加速度(JACCEL)
fAccel	: [in]	加速度	(-1:現在の設定のままに,変更なし)
fDecel	[in]	減速度	(-1:現在の設定のままに,変更なし)

使用例

```
Accelerate 0, 50.0, -1          ’ 加速度 = 50% , 減速度 = 変更なし
Accelerate 0, -1, 60.0          ’ 加速度 =変更なし , 減速度 = 60%
```

5.2.21. CaoRobot::AddVariable メソッド

CaoRobot クラスの AddVariable メソッドの引数は，システム変数名を指定します．

実装されているシステム変数の一覧は表 5-17 を参照してください．

使用例　ロボットの現在位置(P 型)の参照

```
Dim caoRob As CaoRobot
Set caoRob = caoCtrl.AddRobot( “Arm0″)

Dim caoCurPos As CaoVariable
Set caoCurPos = caoRob.AddVariable( “@CURRENT_POSITION″)

Dim vVal As Variant
vVal = caoCurPos.Value
MsgBox vVal(0) &″,″ & vVal(1) &″,″ & vVal(2)    ‘ X,Y,Z の表示
```

5.2.22. CaoRobot::get_VariableNames プロパティ

AddVariable メソッドで指定できる変数名とシステム変数名の一覧を取得します．

5.2.23. CaoRobot::Halt メソッド

動作中のロボットを停止させます．

RC8 コントローラ内のタスクがロボットの制御権をもっている場合(Takearm している場合)は実行時エラーになります．タスク停止に関しては，CaoTask::Stop メソッドで制御してください．

5.2.24. CaoRobot::Change メソッド

ロボットのツール座標系/ワーク座標系を変更します.

このメソッドは PacScript 言語の CHANGETOOL 及び CHANGEWORK 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

以下に, Change の仕様を示します.

書式 Change <bstrName:BSTR>

bstrName	:	[in]	CHANGETOOL 動作の場合="Tool <番号>“
			CHANGEWORK 動作の場合="Work <番号>“

<番号> : 10 進数で表現される数値(デフォルト:0)

使用例

```
Dim caoRob As CaoRobot
Set caoRob = caoCtrl.AddRobot( “Arm0”)

caoRob.Execute"TakeArm", Array(0, 0)

caoRob.Change"Tool1"  'Tool1 に変更
caoRob.Change"Work1"  'Work1 に変更
caoRob.Move 1,"P10"

caoRob.Execute"GiveArm"
```

5.2.25. CaoRobot::Drive メソッド

このメソッドは本プロバイダで直接サポートされていません.

代わりに複数軸同時動作可能な CaoRobot::Execute の"DriveEx","DriveAEx"を使用してください.

5.2.26. CaoRobot::Move メソッド

ロボットを指定座標へ移動します. このメソッドは PacScript 言語の MOVE 命令に対応します. 以下に, Move の仕様を示します.

書式 Move <lComp:LONG >, <vntPose:POSEDATA> [,<vntPose:POSEDATA>…] [, < bstrOpt:BSTR>]

lComp	:	[in]	補間指定 1:MOVE P,... , 2:MOVE L,... , 3:MOVE C,... , 4:MOVE S,...
vntPose	:	[in]	ポーズ列(POSEDATA 型)
bstrOpt	:	[in]	動作オプション [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション

ポーズ列の POSEDATA 型に関しては「POSEDATA 型定義」を参照してください.
POSEDATA 型データで文字列指定する場合の表記形式と意味は下記の通りです.

VT_BSTR 型(文字列)指定 :

・ 補間指定 = 1, 2 の場合
"[<@パス開始変位>]<ポーズ> [<付加軸オプション>]"
ex. "P1", "@P T100", "@E J520"

・ 補間指定 = 3 の場合
"<ポーズ 1>, [<@パス開始変位>] <ポーズ 2> [<付加軸オプション>]"
※ *** ポーズ 1,2 は必ず同じ変数型でなくてはなりません! ***
ex. "P1,@E P2", "T100,@P T101"

・ 補間指定 = 4 の場合
"[<@パス開始変位>]<自由曲線軌道番号> [<付加軸オプション>]"
ex. "1", "@P 20", "@E 5"

<自由曲線軌道番号> : 10 進数で表現される数値(スプライン曲線番号 1～20)
<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>“,"@A<数値>"のいずれか
<付加軸オプション> : 付加軸を扱う場合は下記書式で付加軸オプションを指定可能です[4].
(ポーズデータ後にブランクを挿入し, 続いて付加軸オプションを指定します)

"EX((<軸番号 1>,<相対移動量 1>)[,(<軸番号 2>,<相対移動量 2>)…])"
あるいは
"EXA((<軸番号 1>,<軸座標 1>)[,(<軸番号 2>,<軸座標 2>)…])"

(例 1) Move 1,"@P P1" ,"NEXT" ‘ MOVE P, @P P1, NEXT
(例 2) Move 3,"P1,@E P2" ‘ MOVE C, P1,@E P2
(例 3) Move 2,"@0 P(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 4) Move 4,"@E 2" ‘ MOVE S, @E 2
(例 5) Move 1,"@P P10 EX((7, 30.5))" ,"NEXT" ‘ MOVE P, @P P10 EX((7,30.5)), NEXT
(例 6) Move 2,"@E P20 EXA((7, 30.8), (8, 90.5))" ‘ MOVE L, @E P20 EXA((7, 30.8), (8, 90.5))"
(例 7) Move 1,"P1,@A[50] P2" ‘ MOVE P, P1,@A[50] P2

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.
Move メソッドで対応している PacScript 言語の MOVE コマンドの一覧を示します.

[4]付加軸オプションを使用する場合は必ずコントローラ側で付加軸に対応した設定（アームグループの定義等）を行った上で, TakeArm コマンドで付加軸が制御できるアームグループを予め選択してください.

表 5-9 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*>“
MOVE C,…	MOVE C, P<*n1*>, P<*n2*>	Move 3,"P<*n1*>, P<*n2*>“
	MOVE C, P<*n1*>, @P P<*n2*>	Move 3,"P<*n1*>, @P P<*n2*>“
	MOVE C, P<*n1*>, @E P<*n2*>	Move 3,"P<*n1*>, @E P<*n2*>“
	MOVE C, T<*n1*>, T<*n2*>	Move 3,"T<*n1*>, T<*n2*>“
	MOVE C, T<*n1*>, @P T<*n2*>	Move 3,"T<*n1*>, @P T<*n2*>“
	MOVE C, T<*n1*>, @E T<*n2*>	Move 3,"T<*n1*>, @E T<*n2*>“
	MOVE C, J<*n1*>, J<*n2*>	Move 3,"J<*n1*>, J<*n2*>“
	MOVE C, J<*n1*>, @P J<*n2*>	Move 3,"J<*n1*>, @P J<*n2*>“
	MOVE C, J<*n1*>, @E J<*n2*>	Move 3,"J<*n1*>, @E J<*n2*>“
MOVE S,…	MOVE S, *n1*	Move 4, “*n1*”
	MOVE S, @P *n1*	Move 4, “@P *n1*”
	MOVE S, @E *n1*	Move 4, “@E *n1*”
付加軸対応	MOVE P, P<*n1*> EX((*j1, v1*))	Move 1,"P<*n1*> EX((*j1,v1*))"
	MOVE P, P<*n1*> EX((*j1, v1*),(*j2, v2*))	Move 1,"P<*n1*> EX((*j1,v1*),(*j2, v2*))"

	MOVE P, P<*n1*> EXA((*j1, v1*)) MOVE P, P<*n1*> EXA((*j1, v1*),(*j2, v2*))	Move 1,"P<*n1*> EXA((*j1,v1*))" Move 1,"P<*n1*> EXA((*j1,v1*),(*j2, v2*))"
その他	MOVE P, P<*n1*> +(*x,y,z,rx,ry,rz*) MOVE P, P<*n1*> +(*x,y,z,rx,ry,rz*)H	Move 1, DEV("P<*n1*>","P(*x,y,z,rx,ry,rz*)") Move 1, DEVH("P<*n1*>","P(*x,y,z,rx,ry,rz*)") ※DEV, DEVH は CaoRobot::Execute を参照

<*n1*>, <*n2*>： 整数 0～65535 または"(<要素 1>,<要素 2>,…)"

5.2.27. CaoRobot::Rotate メソッド

指定した軸回りの回転動作を行います.

このメソッドは PacScript 言語の ROTATE 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

以下に, Rotate の仕様を示します.

書式 Rotate <vntRotSuf:POSEDATA>, <fDeg:FLOAT>, <vntPivot:POSEDATA>, <bstrOpt:BSTR>

vntRotSuf	:	[in]	回転面
fDeg	:	[in]	角度(deg)
vntPivot	:	[in]	回転中心
bstrOpt		[in]	パス [@0][@P][@E][@<数値>] 回転オプション [,Pose=n] Pose=1：回転とともに姿勢も変化させます. Pose=2：現在位置(始点)の姿勢を保ったまま軌跡だけ回転動作します. 動作オプション [,SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S)：移動速度を指定します. 意味は SPEED 文と同じです. ACCEL：加速度を指定します. 意味は ACCEL 文と同じです. DECEL：減速度を指定します. 意味は DECEL 文と同じです. TIME：動作にかかる時間を指定します. NEXT：非同期実行オプション

ここで, POSEDATA 型に関しては「POSEDATA 型定義」を参照してください. POSEDATA 型データで文字列指定する場合の表記形式と意味は下記の通りです.

VT_BSTR 型(文字列)指定 ：

・ vntRotSuf（回転面）の場合

“V<n1>,V<n2>,V<n3>“ または"XY","YZ","ZX","XYH","YZH","ZXH"

または"V(<x>,<y>,<z>),V(…),V(…)"

ex."V100,V101,V102"

但し, "XY","YZ","ZX","XYH","YZH","ZXH"は VT_BSTR 型でのみ対応.

・ vntPivot（回転中心）の場合

“V<n4>“ または"V(<x>,<y>,<z>)"

ex."V103"

(例 1) Rotate"V1,V2,V3", 45.8,"V4","@E" ‘ ROTATE V1,V2,V3, @E 45.8, V4

(例 2) Rotate"V(0,0,1),V(0,1,0),V(0,0,0)", 30.0,"V(0,0,0)","@E,pose=1,NEXT"

(例 3) Rotate"XY", 90.0,"V(0,0,0)","@P"

(例 4) Rotate"XYH", -45.0,"V(250,0,0)","@150"

回転面は 3 つの V 型変数の番号を指定して, その 3 点の XYZ 座標からベース座標基準の平面を作ります. 引数 vntRotSuf には BSTR(文字列)型でカンマ (またはスペース, タブ) 区切りの 3 つの V 型変数を指定します.

回転中心 vntPivot には BSTR(文字列)型で 1 つのベクトル型の変数を指定します.

5.2.28. CaoRobot::Speed メソッド

ロボットの内部移動速度を指定します.

このメソッドは PacScript 言語の SPEED,JSPEED 命令に対応します.

実速度は外部速度と内部速度の掛け算で決定されます.

外部移動速度に関しては CaoRobot::Execute の"ExtSpeed"を参照してください.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

以下に, Speed の仕様を示します.

書式 Speed <lAxis:LONG >, <fSpeed:FLOAT>

lAxis	:	[in]	軸番号 -1:手先速度(SPEED), 0:全軸速度(JSPEED)
fSpeed	:	[in]	速度

使用例

```
Dim caoRob As CaoRobot
Set caoRob = caoCtrl.AddRobot( “Arm0”)

caoRob.Execute”TakeArm”, Array(0, 0)

caoRob.Speed -1, 85                    ' 内部速度 85%
caoRob.Execute”ExtSpeed”, 50           ' 外部速度 50% , 実速度 85*50 = 42.5%

caoRob.Execute”GiveArm”
```

Move, Rotate,Drive, Draw などのロボット動作コマンドの SPEED オプションにより内部速度が動作単位で一時的に変更可能ですが, 動作完了後は CaoRobot::Speed で設定された値に戻ります.

内部速度は通常, CaoRobot::Execute の"TakeArm"コマンドで 100%に初期化されます.

5.2.29. CaoRobot::Execute メソッド

Execute メソッドは CaoRobot クラスで対応していないロボット特有の動作コマンドをユーザ自身で定義し実装できるようにする機能を提供します.

書式 [<vntRet:VARIANT> =] Execute(<bstrCmd:BSTR > [,<vntParam:VARIANT>])

bstrCmd	:	[in]	コマンド
vntParam	:	[in]	パラメータ
vntRet		[out]	返り値

実行時バインディングの機能を使って本クラスに定義していないメソッドを呼び出した場合, 次の仕様で Execute メソッドが自動的に呼ばれます.

```
vntRet = Obj.CommandName( Param1, Param2, ...)
↓
vntRet = Obj.Execute("CommandName", Array(Param1, Param2, ... ) )
```

1. 第一引数にコマンド名が BSTR 文字列で渡る
2. 第二引数に全パラメータが VARIANT 配列で渡る

現在指定可能なコマンドの一覧を示します.

表 5-10 CaoRobot::Execute メソッドのコマンド一覧

カテゴリ	コマンド名	機能	対応 Version	
演算				
	TMul	同次変換型同士の積を計算します	1.0.0	P.98
	TInv	同次変換型の逆行列を計算します	1.0.0	P.99
	TNorm	同次変換型の逆行列を計算します	1.0.0	P.99
	J2T	J 型から T 型へ変換	1.0.0	P.99
	T2J	T 型から J 型へ変換	1.0.0	P.100
	J2P	J 型から P 型へ変換	1.0.0	P.100
	P2J	P 型から J 型へ変換	1.0.0	P.101
	T2P	T 型から P 型へ変換	1.0.0	P.101
	P2T	P 型から T 型へ変換	1.0.0	P.102
	Dev	ベース座標系でのオフセットを計算します	1.0.0	P.102
	DevH	ツール座標系でのオフセットを計算します	1.0.0	P.103

	OutRange	可動範囲かを判定します	1.0.0	P.103
	MPS	mps 単位から SPEED コマンドの値を計算します	1.0.0	P.104
	RPM	rpm 単位から SPEED コマンドの値を計算します	1.0.0	P.104
	DPS	deg/s 単位から SPEED コマンドの値を計算します	1.7.2	P.105
位置取得				
	CurPos	現在位置を P 型で取得します	1.0.0	P.105
	DestPos	目標位置を P 型で取得します	1.0.0	P.106
	CurJnt	現在位置を J 型で取得します	1.0.0	P.108
	DestJnt	目標位置を J 型で取得します	1.0.0	P.108
	CurTrn	現在位置を T 型で取得します	1.0.0	P.110
	DestTrn	目標位置を T 型で取得します	1.0.0	P.111
	CurFig	現在の姿勢 Fig の値を取得します	1.0.0	P.113
	CurPosEx	現在位置をタイムスタンプ付きで P 型で取得します	1.4.9	P.106
	DestPosEx	目標位置をタイムスタンプ付きで P 型で取得します	1.4.9	P.107
	HighCurPosEx	リアルタイムに現在位置をタイムスタンプ付きで P 型で取得します	1.4.9	P.107
	CurJntEx	現在位置をタイムスタンプ付きで J 型で取得します	1.4.9	P.109
	DestJntEx	目標位置をタイムスタンプ付きで J 型で取得します	1.4.9	P.109
	HighCurJntEx	リアルタイムに現在位置をタイムスタンプ付きで J 型で取得します	1.4.9	P.110
	CurTrnEx	現在位置をタイムスタンプ付きで T 型で取得します	1.4.9	P.111
	DestTrnEx	目標位置をタイムスタンプ付きで T 型で取得します	1.4.9	P.112
	HighCurTrnEx	リアルタイムに現在位置をタイムスタンプ付きで T 型で取得します	1.4.9	P.112
速度				
	SpeedMode	最適速度制御機能の設定を変更します	1.3.0	P.147
	CurSpd	内部速度の設定値を返します	1.5.1	P.113
	CurAcc	内部加速度の設定値を返します	1.5.1	P.113
	CurDec	内部減速度の設定値を返します	1.5.1	P.114
	CurJSpd	内部軸速度の設定値を返します	1.5.1	P.114
	CurJAcc	内部軸加速度の設定値を返します	1.5.1	P.114
	CurJDec	内部軸減速度の設定値を返します	1.5.1	P.115
ログ				

	StartLog	制御ログの記録を開始します	1.0.0	P.116
	StopLog	制御ログの記録を停止します	1.0.0	P.116
	ClearLog	制御ログのデータをクリアします	1.0.0	P.117
	StartServoLog	サーボログの記録を開始します	1.9.4	P.153
	ClearServoLog	サーボログの記録を消去します	1.9.4	P.153
	StopServoLog	サーボログの記録を停止します	1.9.4	P.153
	GetCtrlLogMaxTime	制御ログの記録時間を取得します	1.0.0	P.154
	SetCtrlLogMaxTime	制御ログの記録時間を設定します	1.0.0	P.154
	GetCtrlLogInterval	制御ログの記録間隔を取得します	1.0.0	P.154
	SetCtrlLogInterval	制御ログの記録間隔を設定します	1.0.0	P.155
	GetLogCount	制御ログデータの個数を取得します	1.0.0	P.160
	GetLogRecord	制御ログデータの情報を取得します	1.0.0	P.161
ロボット動作				
	Motor	モータを ON/OFF します	1.0.0	P.117
	ExtSpeed	外部速度の設定を行います	1.0.0	P.118
	TakeArm	制御権の取得要求を行います	1.0.0	P.118
	GiveArm	制御権の解放要求を行います	1.0.0	P.119
	Draw	ワーク座標系指定で相対動作を行います	1.0.0	P.120
	Approach	ツール座標系指定で絶対動作を行います	1.0.0	P.121
	Depart	ツール座標系指定で相対動作を行います	1.0.0	P.122
	DriveEx	各軸の相対動作を行います	1.0.0	P.123
	DriveAEx	各軸の絶対動作を行います	1.0.0	P.124
	RotateH	アプローチベクトルを軸とした回転動作を行います	1.0.0	P.125
	Arrive	ロボットが指定した動作割合に達するまで待ちます	1.0.0	P.125
	MotionSkip	実行中のロボットの動作を中断します	1.0.0	P.126
	MotionComplete	ロボット動作が完了したかを判断します	1.0.0	P.126
	ArchMove	アーチモーションします	1.4.2	P.136
ツール				
	CurTool	現在のツール番号を取得します	1.0.0	P.127
	GetToolDef	ツール番号で指定したツール定義を取得します	1.0.0	P.127
	SetToolDef	ツール定義を設定します	1.0.0	P.128
ワーク				
	CurWork	現在のワーク番号を取得します	1.0.0	P.128

	GetWorkDef	ワーク番号で指定したワーク定義を取得します	1.0.0	P.128
	SetWorkDef	ワーク定義を設定します	1.0.0	P.129
	WorkAttribute	ワーク番号で指定したワーク属性を取得します	1.7.11	P.129
ベース				
	SetBaseDef	Base 定義を設定します	1.7.16	P.151
	GetBaseDef	Base 番号で指定した Base 定義を取得します	1.7.16	P.151
エリア				
	GetAreaDef	エリア番号で指定したエリアの定義を取得します	1.0.0	P.130
	SetAreaDef	エリアのパラメータを設定します	1.0.0	P.130
	SetArea	エリアチェックを有効化します	1.0.0	P.131
	ResetArea	エリアチェックを無効化します	1.0.0	P.131
	AreaSize	検知エリアの大きさ(各辺の長さ)をベクトル型で返します	1.0.0	P.132
	GetAreaEnabled	エリアの有効/無効を取得します	1.0.0	P.132
	SetAreaEnabled	エリアの有効/無効を設定します	1.0.0	P.132
スプライン				
	AddPathPoint	経路データに経路点を追加します	1.3.1	P.133
	ClrPathPoint	指定経路の全ての経路点のクリアします	1.3.1	P.133
	GetPathPoint	指定経路点の位置データを取得します	1.3.1	P.134
	LoadPathPoint	経路データを読込みます	1.3.1	P.134
	GetPathPointCount	指定経路の経路点の個数を取得します	1.3.1	P.135
力制御				
	ForceCtrl	力制御機能の有効／無効を設定します	1.3.1	P.138
	ForceParam	力制御機能のパラメータを設定します	1.4.2	P.139
	ForceValue	要求データに応じた力制御の値を取得します	1.5.1	P.140
	ForceWaitCondition	力制御の指定した条件になるまで待機します	1.6.0	P.141
	ForceSensor	力覚センサの制御を行います	1.5.1	P.142
	ForceChangeTable	力制御中の力制御テーブルを変更します	1.6.3	P.142
	CurForceParam	現在の力制御機能(コンプライアンス機能)のパラメータを取得します	2.0.10	P.160
協調機能				
	SyncTimeStart	複数台ロボットの同時発着動作の開始指示します	1.7.13	P.149

	SyncTimeEnd	複数台ロボットの同時発着動作を開始します	1.7.13	P.150
	SyncMoveStart	複数台ロボットの協調動作の開始を指示します	1.7.13	P.150
	SyncMoveEnd	複数台ロボットの同時発着動作を開始します	1.7.13	P.151
	SetHandIO	HandIO を設定します	1.8.10	P.152
	GetHandIO	HandIO を取得します	1.8.10	P.152
ロボット排他制御				
	ExclusiveArea	排他エリアを作成，変更します	1.11.1	P.157
	SetExclusiveArea	排他エリアを有効にします	1.11.1	P.158
	ResetExclusiveArea	排他エリアを無効にします	1.11.1	P.158
	ExclusiveControlStatus	排他制御状態を取得します	1.11.1	P.158
精度				
	CrtMotionAllow	Move @C で，停止判定に使用する，手先の「停止位置精度，姿勢精度」を変更します	1.3.3	P.136
	EncMotionAllow	Move @E で，停止判定に使用する，ロボット軸の各軸の「停止時許容角度」を変更します	1.3.3	P.137
	EncMotionAllowJnt	Move @E で，停止判定に使用する，ロボット軸でない軸の「停止時許容角度」を変更します	1.4.2	P.137
	HighPathAccuracy	高軌跡制御機能の有効/無効を切り替えます	1.3.7	P.146
状態，値取得				
	GetSrvData	ロボット軸のサーボ内部データを返します	1.0.0	P.142
	GetSrvJntData	指定軸のサーボ内部データを返します	1.0.0	P.143
	GetAllSrvData	サーボ内部データを一括で取得します	2.3.0	P.159
	RobInfo	ロボットの情報を返します	1.7.11	P.149
機能，条件の設定				
	GrvCtrl	重力補償制御機能の有効/無効を操作します	1.0.0	P.144
	GrvOffset	重力オフセット設定機能の有効/無効を操作します	1.0.0	P.145
	MotionTimeout	動作命令のタイムアウト設定値を変更します	1.4.2	P.146
	ErAlw	偏差許容機能の設定と有効/無効を操作します	1.0.0	P.138
	PayLoad	内部負荷条件の設定値を変更します	1.0.0	P.147
	SingularAvoid	特異点回避機能を有効化，または無効化します	1.3.1	P.147

電流制限				
	CurLmt	電流制限機能の設定と有効/無効を操作します	1.0.0	P.144
	Zforce	第3軸(Z軸)の電流制限機能を推力で指定します	1.4.2	P.145
位置, 動作と IO との連携				
	DetectOn	Detect 機能を有効にします	1.3.0	P.155
	DetectOff	Detect 機能を無効にし, データを格納します	1.3.0	P.155
	AngularTrigger	ロボットが指定した角度(距離)動作するたびに I/O の ON/OFF を切替えます	1.4.1	P.156
バーチャルフェンス				
	VirtualFence	バーチャルフェンスの監視を開始または終了します	1.12.0	P.157
その他				
	GetRobotTypeName	ロボット型式を取得します	1.0.0	P.135
	GetPluralServoData	サーボデータを一括で取得します	1.10.1	P.156
	GenerateNonStopPath	無停止教示点補正機能オプションのコマンドです	1.6.2	P.148

CaoRobot クラスの Execute メソッドの引数は, コマンド番号+パラメータを VARIANT 配列で指定します.

使用例

```
Dim vRes as Variant
vRes = caoRob.Execute( “GetJntData”,Array(1, 6 ) )    ‘ 6 軸のモータ速度現在値[rpm]
caoRob.Execute"ExtSpeed",Array(50.0)                  ‘ 外部速度=50%
caoRob.Execute"APPROACH", Array(1,"P11","@P 100","NEXT")  ‘APPROACH P,P11,@P 100, NEXT
```

5.2.29.1. CaoRobot::Execute("TMul") コマンド

同次変換型同士の積を計算します.

書式　TMul (<Tn1>, <Tn2>)

<Tn1> : [in] T 型 (POSEDATA)
<Tn2> : [in] T 型 (POSEDATA)
戻り値 : <Tn1>と<Tn2>の積
(VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

使用例

```
Dim vResult As Variant
```

```
vResult = caoRob.Execute("TMul",  Array("T10","T20" ) )     'T 型のインデックスを指定して計算

vResult = caoRob.Execute("TMul",  Array("T(400,500,400,  1,0,0,  0,1,0,  5)", _
 "T( 100,0,0,  1,0,0,  0,1,0,  -1)" ) )     'T 型の要素を直接指定して計算
```

5.2.29.2. CaoRobot::Execute("TInv") コマンド

T(同次変換)型の逆行列を計算します.

書式　TInv(<Tn1>)

<Tn1> : [in] T 型 (POSEDATA)
戻り値 : <Tn1>の逆行列
(VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("TInv", "T10" )     'T10 の逆行列
vResult = caoRob.Execute("TInv", "T(400,500,400,  1,0,0,  0,1,0,  5)" )
  'T 型の要素を直接指定して計算
```

5.2.29.3. CaoRobot::Execute("TNorm") コマンド

T(同次変換)型の正規化を行います.

書式　TNorm(<Tn1>)

<Tn1> : [in] T 型 (POSEDATA)
戻り値 : <Tn1>の正規化
(VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("TNorm", "T10" )     'T10 の正規化

vResult = caoRob.Execute("TNorm", "T(400,500,400,  1,0,0,  0,1,0,  5)" )
   'T 型の要素を直接指定して計算
```

5.2.29.4. CaoRobot::Execute("J2T") コマンド

J 型から T 型への変換を行います.

書式　J2T (<Jn1>)

<Jn1> : [in] J 型 (POSEDATA)
戻り値 : T 型
(VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("J2T", "J10" )     'J10 の値を T 型に変換
vResult = caoRob.Execute("J2T", "J(90,90,90, 0,0,0)" )
    'J 型の要素を直接指定して変換
```

5.2.29.5. CaoRobot::Execute("T2J") コマンド

T 型から J 型への変換を行います.

書式 T2J (<Tn1>)

<Tn1> : [in] T 型 (POSEDATA)
戻り値 : J 型
(VT_VARIANT[VT_R8|VT_ARRAY:8 要素])

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("T2J", "T10" )     'T10 の値を J 型に変換
vResult = caoRob.Execute("T2J", "T(400,400,500, 1,0,0, 0,1,0, 5)" )
    'T 型の要素を直接指定して変換
```

5.2.29.6. CaoRobot::Execute("J2P") コマンド

J 型から P 型への変換を行います.

書式 J2P (<Jn1>)

<Jn1> : [in] J 型 (POSEDATA)
戻り値 : P 型
(VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("J2P", "J10" )     'J10 の値を P 型に変換

vResult = caoRob.Execute("J2P", "J(90,90,90, 0,0,0)" )
    'J 型の要素を直接指定して変換
```

5.2.29.7. CaoRobot::Execute("P2J") コマンド

P 型から J 型への変換を行います.

書式　　P2J (<Pn1>)

<Pn1>　　:　[in] P 型 (POSEDATA)
戻り値　　:　J 型
(VT_VARIANT[VT_R8|VT_ARRAY:8 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("P2J", "P10" )     'P10 の値を J 型に変換

vResult = caoRob.Execute("P2J", "P(400,400,500, 180,0,180, 5)" )
    'P 型の要素を直接指定して変換
```

5.2.29.8. CaoRobot::Execute("T2P") コマンド

T 型から P 型への変換を行います.

書式　　T2P (<Tn1>)

<Tn1>　　:　[in] T 型 (POSEDATA)
戻り値　　:　P 型
(VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("T2P", "T10" )     'T10 の値を P 型に変換

vResult = caoRob.Execute("T2P", "T(400,400,500, 1,0,0, 0,1,0, 5)" )
    'T 型の要素を直接指定して変換
```

5.2.29.9. CaoRobot::Execute("P2T") コマンド

P 型から T 型への変換を行います.

書式　P2T (<Pn1>)

<Pn1>	:	[in] P 型 (POSEDATA)
戻り値	:	T 型 (VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("P2T", "P10" )      'P10 の値を T 型に変換

vResult = caoRob.Execute("P2T", "P(400,400,500, 180,0,180, 5)" )
    'P 型の要素を直接指定して変換
```

5.2.29.10. CaoRobot::Execute("Dev") コマンド

ベース座標系で基準位置<Pn1>からのオフセット<Pn2>の座標を計算します. オフセット<Pn2>のFig値は無視されます.

書式　Dev (<Pn1>, <Pn2>)

<Pn1>	:	[in] P 型 (POSEDATA)
<Pn2>	:	[in] P 型 (POSEDATA)
戻り値	:	P 型 (VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("Dev", Array("P10","P(100, 200, 300, 180, 0, 180)" ))
 'P10 + P(100,200,300, 180, 0, 180)の位置を計算
```

5.2.29.11. CaoRobot::Execute("DevH") コマンド

ツール座標系で基準位置<Pn1>からのオフセット<Pn2>の座標を計算します． オフセット<Pn2>のFig値は無視されます.

書式　DevH (<Pn1>, <Pn2>)

<Pn1>	:	[in] P 型 (POSEDATA)
<Pn2>	:	[in] P 型 (POSEDATA)
戻り値	:	P 型 (VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

現在有効になっているツール定義（カレントツール）の座標をベースに計算が行われます.

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("DevH", Array("P10","P(100, 200, 300, 180, 0, 180)" ))
 'P10 + ツール座標 P(100,200,300, 180, 0, 180)の位置を計算
```

5.2.29.12. CaoRobot::Execute("OutRange") コマンド

位置データがロボットの可動範囲内であるかを返します.

<Pose>が J 型指定の場合は，ツール座標，ワーク座標は無視されます. ツール座標，およびワーク座標を POSEDATA 型データで指定できるのは，Version.2.0.*以降のコントローラだけです.

書式　OutRange(<Pose>[, <ToolDef> [, <WorkDef>]])

<Pose>	:	[in] POSEDATA の値（P 型，J 型，T 型のいずれか）
<ToolDef>	:	[in]ツール座標　POSEDATA 型（P 型） か VT_I4 POSEDATA 型（P 型）の値の場合 ： ツール座標 VT_I4 の場合 ： ツール番号（-1(省略時)は現在のツール番号）
<WorkDef>	:	[in]ワーク座標　POSEDATA 型（P 型） か VT_I4 POSEDATA 型（P 型）の場合 ： ワーク座標 VT_I4 の場合 ： ワーク番号（-1(省略時)は現在のワーク番号）
戻り値	:	VT_I4 0:可動範囲内 1～63:ソフトリミットである軸のビット -1:軸構成上計算不可能な位置 -2:特異点

使用例　可動範囲なら移動させる

```
Dim lRet As Long

lRet = caoRob.Execute("OutRange", "P(400, 400, 300, 180, 0, 180, 5)" )
```

5.2.29.13. CaoRobot::Execute("MPS") コマンド

動作速度 mm/sec の値から SPEED コマンドの値%に変換します.

書式　Mps(<mps>)

<mps>	:	[in] スピード mm/sec の値 (VT_R4)
戻り値	:	SPEED コマンドの値% (VT_R4)

使用例　絶対速度から相対速度への換算

```
Dim vSp As Variant

vSp = caoRob.Execute("MPS",  200.0 )  ' 200.0 mm/sec
caoRob.Speed -1, vSP
```

5.2.29.14. CaoRobot::Execute("RPM") コマンド

回転速度 rpm の値から SPEED コマンドの値%に変換します.

書式　Rpm(<Axis>, <rpm>)

<Axis>	:	[in] 軸番号 (VT_I4)
<rpm>	:	[in] 回転速度 rpm の値 (VT_R4)
戻り値	:	SPEED コマンドの値% (VT_R4)

使用例　回転速度 rpm から相対速度(%)への換算

```
Dim vSp As Variant

vSp = g_caoRobot.Execute("RPM", Array(1, 60))  ' 1 軸, 60.0 rpm
caoRob.Speed -1, vSP
```

5.2.29.15. CaoRobot::Execute("DPS") コマンド

指定した軸番号の回転速度(deg/s)を PTP 動作時の最大内部速度に対する割合(%)に変換します.
軸番号を指定しない場合は回転速度(deg/s)を CP 動作時の最大内部速度に対する割合(%)に変換します.

書式　DPS(<Axis>, < deg/s >)

<Axis>	:	[in] 軸番号 (VT_I4)
< deg/s >	:	[in] 回転速度 deg/s の値 (VT_R4)
戻り値	:	SPEED コマンドの値% (VT_R4)

使用例　回転速度 deg/s から相対速度(%)への換算

```
Dim vSp As Variant

' 50(Deg/sec)で移動(回転動作の場合)
vSp = g_caoRobot.Execute("DPS", 50)
  caoRob.Speed -1,vSp

  ' 1 軸を 50(deg/sec)で移動
vSp = g_caoRobot.Execute("DPS", Array(1, 50))
caoRob.Speed 0,vSp
```

5.2.29.16. CaoRobot::Execute("CurPos") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置を P 型で取得します.

書式　CurPos( )

引数	:	なし
戻り値	:	P 型 (VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

システム変数の"@Current_Position"で得られる値と等価です.

使用例

```
Dim vResult As Variant

vResult = caoRob.Execute("CurPos" )     '現在位置取得
```

5.2.29.17. CaoRobot::Execute("DestPos") コマンド

目標位置を P 型で取得します.

書式　DestPos( )

引数　：なし
戻り値　：P 型 (VT_VARIANT[VT_R8|VT_ARRAY:7 要素])

システム変数の"@Dest_Position"で得られる値と等価です.

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestPos" )　'目標位置取得
```

5.2.29.18. CaoRobot::Execute("CurPosEx") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置をタイムスタンプ付きで P 型で取得します.
タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　CurPosEx( )

引数　：なし
戻り値　：タイムスタンプ＋P 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+7 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("CurPosEx" )　'タイムスタンプ+現在位置を取得
```

5.2.29.19. CaoRobot::Execute("DestPosEx") コマンド

目標位置をタイムスタンプ付きで P 型で取得します.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　DestPosEx( )

引数　：なし

戻り値　：タイムスタンプ＋P 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+7 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestPosEx" )　'タイムスタンプ+目標位置を取得
```

5.2.29.20. CaoRobot::Execute("HighCurPosEx") コマンド

リアルタイムに現在位置をタイムスタンプ付きで P 型で取得します.

マシンロック時以外はエンコーダ値を返します. コントローラに高負荷がかかる場合があります.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　HighCurPosEx( )

引数　：なし

戻り値　：タイムスタンプ＋P 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+7 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("HighCurPosEx" )　'タイムスタンプ+現在位置を取得
```

5.2.29.21. CaoRobot::Execute("CurJnt") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置を J 型で取得します.

書式　CurJnt( )

引数　：なし
戻り値　：J 型 (VT_VARIANT[VT_R8|VT_ARRAY:8 要素])

システム変数の"@Current_Angle"で得られる値と等価です.

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("CurJnt")    '現在位置取得
```

5.2.29.22. CaoRobot::Execute("DestJnt") コマンド

目標位置を J 型で取得します.

書式　DestPos( )

引数　：なし
戻り値　：P 型 (VT_VARIANT[VT_R8|VT_ARRAY:8 要素])

システム変数の"@Dest_Angle"で得られる値と等価です.

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestJnt")    '目標位置取得
```

5.2.29.23. CaoRobot::Execute("CurJntEx") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置をタイムスタンプ付きで J 型で取得します.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　CurJntEx( )

引数　: なし

戻り値　: タイムスタンプ＋J 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+8 要素])
{Time, J1, J2, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("CurJntEx" )　'タイムスタンプ+現在位置を取得
```

5.2.29.24. CaoRobot::Execute("DestJntEx") コマンド

目標位置をタイムスタンプ付きで J 型で取得します.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　DestJntEx( )

引数　: なし

戻り値　: タイムスタンプ＋J 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+8 要素])
{Time, J1, J2, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestJntEx" )　'タイムスタンプ+目標位置を取得
```

5.2.29.25. CaoRobot::Execute("HighCurJntEx") コマンド

リアルタイムに現在位置をタイムスタンプ付きで J 型で取得します.

マシンロック時以外はエンコーダ値を返します. コントローラに高負荷がかかる場合があります.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　HighCurJntEx( )

引数　：なし

戻り値　：タイムスタンプ＋J 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+8 要素])
{Time, J1, J2, …}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("HighCurJntEx")　'タイムスタンプ+現在位置を取得
```

5.2.29.26. CaoRobot::Execute("CurTrn") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置を T 型で取得します.

書式　CurTrn( )

引数　：なし

戻り値　：T 型 (VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

システム変数の"@Current_Trans"で得られる値と等価です.

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("CurTrn")　'現在位置取得
```

5.2.29.27. CaoRobot::Execute("DestTrn") コマンド

目標位置を T 型で取得します.

書式　DestTrn( )

引数　：なし
戻り値　：T 型 (VT_VARIANT[VT_R8|VT_ARRAY:10 要素])

システム変数の"@Dest_Trans"で得られる値と等価です.

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestTrn" )　'目標位置取得
```

5.2.29.28. CaoRobot::Execute("CurTrnEx") コマンド

コントローラ内部で一定周期(8ms)に更新された現在位置をタイムスタンプ付きで T 型で取得します.
タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　CurTrnEx( )

引数　：なし
戻り値　：タイムスタンプ＋T 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+10 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("CurTrnEx" )　'タイムスタンプ+現在位置を取得
```

5.2.29.29. CaoRobot::Execute("DestTrnEx") コマンド

目標位置をタイムスタンプ付きで T 型で取得します.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　DestTrnEx( )

引数　：なし

戻り値　：タイムスタンプ＋T 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+10 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("DestTrnEx" )　'タイムスタンプ+目標位置を取得
```

5.2.29.30. CaoRobot::Execute("HighCurTrnEx") コマンド

リアルタイムに現在位置をタイムスタンプ付きで T 型で取得します.

マシンロック時以外はエンコーダ値を返します. コントローラに高負荷がかかる場合があります.

タイムスタンプ:コントローラ電源 ON からの時間(msec)

書式　HighCurTrnEx( )

引数　：なし

戻り値　：タイムスタンプ＋T 型
(VT_VARIANT[VT_R8|VT_ARRAY:1+10 要素])
{Time, X, Y, ...}

使用例

```
Dim vResult As Variant
vResult = caoRob.Execute("HighCurTrnEx" )　'タイムスタンプ+現在位置を取得
```

5.2.29.31. CaoRobot::Execute("CurFig") コマンド

現在の姿勢を表す Fig の値を取得します.

書式　CurFig( )

引数　：なし

戻り値　：Fig の値 (VT_I4)

使用例

```
Dim vFig As Variant
vFig = caoRob.Execute("CurFig" )　　'現在の Fig を取得
```

5.2.29.32. CaoRobot::Execute("CurSpd") コマンド

内部速度の設定値を返します.

書式　CurSpd([<arm group>])

引数　：アームグループ(VT_I4)

戻り値　：内部速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( "CurSpd" 0)
```

5.2.29.33. CaoRobot::Execute("CurAcc") コマンド

内部加速度の設定値を返します.

書式　CurAcc([<arm group>])

引数　：アームグループ(VT_I4)

戻り値　：内部加速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( "CurAcc" 0)
```

5.2.29.34. CaoRobot::Execute("CurDec") コマンド

内部減速度の設定値を返します.

書式 CurDec([<arm group>])

引数 : アームグループ(VT_I4)

戻り値 : 内部減速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( "CurDec" 0)
```

5.2.29.35. CaoRobot::Execute("CurJSpd") コマンド

内部軸速度の設定値を返します.

書式 CurJSpd([<arm group>])

引数 : アームグループ(VT_I4)

戻り値 : 内部軸速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( "CurJSpd" 0)
```

5.2.29.36. CaoRobot::Execute("CurJAcc") コマンド

内部軸加速度の設定値を返します.

書式 CurJAcc([<arm group>])

引数 : アームグループ(VT_I4)

戻り値 : 内部軸加速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( "CurJAcc" 0)
```

5.2.29.37. CaoRobot::Execute("CurJDec") コマンド

内部軸減速度の設定値を返します.

書式 CurJDec([<arm group>])

引数 : アームグループ(VT_I4)

戻り値 : 内部軸減速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( “CurJDec” 0)
```

5.2.29.38. CaoRobot::Execute("CurExtSpd") コマンド

外部速度の設定値を返します.

書式 CurExtSpd()

引数 : なし

戻り値 : 外部速度(VT_R4)

使用例

```
Dim vSpd As Variant
vSpd = caoRob.Execute( “CurExtSpd” 0)
```

5.2.29.39. CaoRobot::Execute("CurExtAcc") コマンド

外部加速度の設定値を返します.

書式 CurExtAcc()

引数 : なし

戻り値 : 外部加速度(VT_R4)

使用例

```
Dim vAcc As Variant
vAcc = caoRob.Execute( “CurExtAcc” 0)
```

5.2.29.40. CaoRobot::Execute("CurExtDec") コマンド

外部減速度の設定値を返します.

書式 CurExtDec()

引数 : なし

戻り値 : 外部減速度(VT_R4)

使用例

```
Dim vDec As Variant
vDec = caoRob.Execute( “CurExtDec” 0)
```

5.2.29.41. CaoRobot::Execute("StartLog") コマンド

CLearLog で制御ログの記録開始後, StartLog 時点からサンプリング可能なログ個数に達した場合, ログの記録を停止します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 StartLog( )

引数 : なし

戻り値 : なし

記録を可能にするには予め, ClearLog コマンドを実行してください.

使用例

```
caoRob.Execute"StartLog"
```

5.2.29.42. CaoRobot::Execute("StopLog") コマンド

CLearLog で制御ログの記録開始後, StopLog 時点で制御ログの記録を停止します.

書式 StopLog( )

引数 : なし

戻り値 : なし

記録を可能にするには予め, ClearLog コマンドを実行してください.

使用例

```
caoRob.Execute"StopLog"
```

5.2.29.43. CaoRobot::Execute("ClearLog") コマンド

制御ログの記録を開始します.

SlaveMode 中は使用できません.

書式 ClearLog( )

引数 : なし

戻り値 : なし

制御ログデータをクリアすると共に, リングバッファへサンプリングを開始します.

使用例

```
caoRob.Execute"ClearLog"
```

5.2.29.44. CaoRobot::Execute("Motor") コマンド

モータを ON/OFF します.

b-CAP Slave のライセンスキーが追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 Motor (<State> [,<NoWait>])

State : [in]モータ状態(VT_I4)

0:モータ OFF

1:モータ ON

NoWait : [in]完了待ち(VT_I4)

0:完了待ちする (デフォルト値)

1:完了待ちしない

戻り値 : なし

使用例

```
caoRob.Execute"Motor",Array(1,0)  'モータ ON して, モータ ON の完了を待つ
```

5.2.29.45. CaoRobot:: Execute("ExtSpeed") コマンド

外部速度, 外部加速度, 外部減速度を設定します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 ExtSpeed (<Speed> [,<Accel> [,<Decel>]])

Speed	:	[in]外部速度(VT_R4)
Accel	:	[in]外部加速度(VT_R4) -1 現在の設定から変更しない -2(デフォルト) 外部速度の二乗を 100 で割った値 省略時-2 扱い
Decel	:	[in]外部減速度(VT_R4) -1 現在の設定から変更しない -2(デフォルト) 外部速度の二乗を 100 で割った値 省略時-2 扱い
戻り値	:	なし

使用例

```
caoRob.Execute"ExtSpeed", Array(50.0,25.0,25.0)  '外部速度=50% 加速度=25%,減速度=25%
caoRob.Execute"ExtSpeed", Array(50.0)    ' 外部速度=50% (加速度,減速度は自動設定)
```

実速度は外部速度と内部速度の掛け算で決定されます. 内部速度は Speed コマンドで設定します.

5.2.29.46. CaoRobot::Execute("TakeArm") コマンド

制御権の取得要求を行います.

PacScript 言語の TAKEARM 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

Takearm した状態でアプリケーションが異常終了した場合, Takearm が解放されない場合があります. その場合, VRC パラメータの 31 番「非常停止時 PC からの Takearm を解放」を 1 にして非常停止を入力することで解放されます.

書式 TakeArm ([<ArmGroup> , [<Keep>]])

ArmGroup	:	[in]アームグループ番号(VT_I4) 0~31 (省略時 0) (0 は付加軸を含まないロボットのみのグループ)
Keep	:	[in]初期化設定値(VT_I4) 0:内部速度を 100, カレントツール番号とカレントワーク番号を 0 にする 1:現在の内部速度, カレントツール番号, カレントワーク番号を変更せず, 保持

(省略時 0)

※Keep オプションを指定しなかった場合は内部速度=100,カレントツール番号=0,カレントワーク番号=0 に初期化されます.

戻り値 : なし

使用例

```
caoRob.Execute"Takearm",Array(0,0)  ‘内部速度=100,カレントツール=0,カレントワーク=0 に初期化
:

‘ 内部速度 = 50, カレントツール=1, カレントワーク=0 だった場合
caoRob.Execute"Takearm",Array(0,1)  ‘内部速度,カレントツール,カレントワークは初期化されず,
‘制御権のみを取得する
```

5.2.29.47. CaoRobot::Execute("GiveArm") コマンド

制御権の解放要求を行います.

PacScript 言語の GIVEARM 命令に対応します.

書式 GiveArm ()

引数 : なし

戻り値 : なし

使用例

```
‘Takearm されたプログラム終了時は, ロボットが瞬時停止してから Givearm します.
‘Next 動作の完了を待ってからプログラムを終了したい場合は
‘明示的に Givearm コマンドを実行してください.

caoRob.Execute"Takearm",Array(0,0)
caoRob.Move 1,” @0 J(0,45,90,0,45,0)”
caoRob.Move 1,” @0 J(90,45,90,0,45,0)” ,” Next”
Set IO[129]
Set IO[130]
caoRob.Execute"GiveArm"       ’ Next の動作完了待ちをする’
```

5.2.29.48. CaoRobot::Execute("Draw") コマンド

ワーク座標系指定で相対動作を行います.

PacScript 言語の DRAW 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　Draw (<lComp>,<vntPose>[,<strOpt>])

lComp	:	[in]補間方法(VT_I4) 1:PTP 動作 2:CP 動作
vntPose	:	[in]移動量(POSEDATA 型 C1 書式) "[<@パス開始変位>]<並進移動量>“
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
caoRob.Execute"Draw",  Array(1,"VO")
caoRob.Execute"Draw",  Array(2,"V(100, 100, 100)")
```

5.2.29.49. CaoRobot::Execute("Approach") コマンド

基準位置から指定距離離れたアプローチ位置へ移動します.

PacScript 言語の APPROACH 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　Approach (<lComp>,<vntPoseBase>,<vntPoseLen>[,<strOpt>])

lComp	:	[in]補間方法(VT_I4) 1:PTP 動作 2:CP 動作
vntPoseBase	:	[in] 基準位置(POSEDATA 型 C0 書式) "<位置:P,T,J 型>“ POSEDATA 型 C0 書式のパス開始変位指定はエラーです.
vntPoseLen	:	[in] アプローチ長(POSEDATA 型 C2 書式) "[パス開始変位] <値(mm)>“
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
caoRob.Execute"Approach",Array(1," P1","@P 100","S=50")
caoRob.Execute"Approach",Array(2," P(400, 200, 350, 180, 0, 180, 5)","@E 56.8","S=30, NEXT")
```

5.2.29.50. CaoRobot::Execute("Depart") コマンド

現在位置からツール座標-Z 方向に移動します.

PacScript 言語の DEPART 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　Depart (<lComp>,<vntPoseLen>[,<strOpt>])

lComp	:	[in]補間方法(VT_I4) 1:PTP 動作 2:CP 動作
vntPoseLen	:	[in] デパート長(POSEDATA 型 C2 書式) "[パス開始変位] <値(mm)>“
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
caoRob.Execute"Depart",Array(1,"@P 100","S=50")
caoRob.Execute"Depart",Array(2"@E 56.8","S=30, NEXT")
```

5.2.29.51. CaoRobot::Execute("DriveEx") コマンド

各軸の相対動作を行います.

PacScript 言語の DRIVE 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　DriveEx (<vntPoses> [, < strOpt >])

vntPoses	:	[in]軸番号と移動量(POSEDATA 型 C3 書式) 動作させたい軸と移動量を POSEDATA 型で最大 8 軸分指定します.
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
vntPoses = "@0 (1, 10), (2, 10)"
caoRob.Execute "DriveEX", Array(vntPoses, "S=10, NEXT")
```

5.2.29.52. CaoRobot::Execute("DriveAEx") コマンド

各軸の絶対動作を行います.

PacScript 言語の DRIVEA 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　DriveAEx (<vntPoses> [, < strOpt >])

vntPoses	:	[in]軸番号と移動量(POSEDATA 型 C3 書式) 動作させたい軸と軸座標を POSEDATA 型で最大 8 軸分指定します.
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
vntPose1 = Array(Array(1, 10), -1, "@0")
vntPose2 = Array(Array(2, 10), -1)
vntPoses = Array(vntPose1, vntPose2)
caoRob.Execute "DriveAEX", Array(vntPoses, "S=10, NEXT")
caoRob.Execute "DriveAEX", Array( "@0 (1,10), (2,10)", "S=10, NEXT")
```

5.2.29.53. CaoRobot::Execute("RotateH") コマンド

アプローチベクトルを軸とした回転動作を行います.

PacScript 言語の ROTATEH 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 RotateH (<vntPoseAxis> [,<strOpt>])

vntPoseAxis	:	[in] アプローチベクトル回りの相対回転角(POSEDATA 型 C2 書式) "[パス開始変位] <値(degree)>“
strOpt	:	[in]動作オプション(VT_BSTR) [SPEED=n][,ACCEL=n][,DECEL=n][,TIME=n][,NEXT] SPEED (S):移動速度を指定します. 意味は SPEED 文と同じです. ACCEL:加速度を指定します. 意味は ACCEL 文と同じです. DECEL:減速度を指定します. 意味は DECEL 文と同じです. TIME:動作にかかる時間を指定します. NEXT:非同期実行オプション
戻り値	:	なし

使用例

```
caoRob.Execute"RotateH", Array("@P 32.5" ,"S=50" )
```

5.2.29.54. CaoRobot::Execute("Arrive") コマンド

ロボットが指定した動作割合に達するまで待ちます.

PacScript 言語の ARRIVE 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 Arrive (<動作割合>)

引数	:	[in](VT_I4)動作割合
戻り値	:	なし

使用例

```
caoRob.Move 1,"P1","Next"       ‘非同期動作実行
caoRob.Execute"Arrive", 50      ‘50%完了するまで待つ
```

5.2.29.55. CaoRobot::Execute("MotionSkip") コマンド

実行中のロボットの動作を中断します.

PacScript 言語の MOTIONSKIP 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　MotionSkip ([<ArmGroup>[, <Parameter>]])

ArmGroup	:	[in]アームグループ番号(VT_I4)
		-1(デフォルト):現在取得しているアームグループ
Parameter	:	[in]動作継続パターン(VT_I4)
		0(デフォルト):パス開始変位を@0 で指定し, 最大減速で繋ぎます
		1:パス開始変位を@P で指定し, 最大減速で繋ぎます
		2:パス開始変位を@0 で指定し, 設定減速度で繋ぎます
		3:パス開始変位を@P で指定し, 設定減速度で繋ぎます
戻り値	:	なし

使用例

```
caoRob.Execute  "MotionSkip", Array(0, 1)
```

5.2.29.56. CaoRobot::Execute("MotionComplete") コマンド

ロボット動作命令または, ロボット動作が完了したかを判断します.

PacScript 言語の MOTIONCOMPLETE 命令に対応します.

書式　MotionComplete ([<ArmGroup> [,<Mode>]])

ArmGroup	:	[in]アームグループ番号(VT_I4)
		-1(デフォルト):現在取得しているアームグループ
Mode	:	[in]モード
		0(デフォルト) :動作命令完了状態取得
		1:動作完了状態取得
戻り値	:	[out]状態<VT_BOOL>
		Mode 0 の時
		動作命令完了状態:VARIANT_TRUE,
		動作中, 一時停止中, コンティニュ停止中:VARIANT_FALSE
		Mode 1 の時
		ロボット停止中:VARIANT_TRUE,
		ロボット動作中:VARIANT_FALSE

使用例　非同期動作と完了待ち

```
caoRob.Move  1,"P1","Next"  ' P1 へ非同期動作
Do
   ' <移動中の処理>
Loop While( Not caoRob.Execute("MotionComplete", Array(-1, 1) )) ' 動作完了確認
```

5.2.29.57. CaoRobot::Execute("CurTool") コマンド

現在のツール番号を取得します.

書式　CurTool ( )

引数	:	なし
戻り値	:	現在のツール番号(VT_I4)

使用例

```
Debug.Print caoRob.Execute( "CurTool")
```

5.2.29.58. CaoRobot::Execute("GetToolDef") コマンド

ツール番号で指定したツール定義を取得します.

書式　GetToolDef (<ToolNo>)

ToolNo	:	[in]ツール番号(VT_I4)
戻り値	:	ツール定義(VT_R8|VT_ARRAY) X, Y, Z, RX, RY, RZ

使用例

```
Dim vVal As Variant
vVal = caoRob.Execute( "GetToolDef", 1)
Debug.Print"X= " & vVal(0) &", Y= " & vVal(1) &", Z= " & vVal(2)
Debug.Print"RX= " & vVal(3) &", RY= " & vVal(4) &", RZ= " & vVal(5)
```

5.2.29.59. CaoRobot::Execute("SetToolDef") コマンド

ツール定義を設定します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 SetToolDef (<ToolNo>, <ToolDef>)

ToolNo	:	[in]ツール番号(VT_I4)
ToolDef		[in]ツール定義(P 型 Fig は無視されます)
		X, Y, Z, RX, RY, RZ
戻り値	:	なし

使用例

```
caoRobot.Execute "SetToolDef", Array(1, "P2")
caoRobot.Execute "SetToolDef", Array(2, "P(100, 200, 300, 180, 0, 180)")
```

5.2.29.60. CaoRobot::Execute("CurWork") コマンド

現在のワーク番号を取得します.

書式 CurWork ( )

引数	:	なし
戻り値	:	現在のワーク番号(VT_I4)

使用例

```
Debug.Print caoRob.Execute( "CurWork")
```

5.2.29.61. CaoRobot::Execute("GetWorkDef") コマンド

ワーク番号で指定したワーク定義を取得します.

書式 GetWorkDef (<WorklNo>)

WorkNo	:	[in]ワーク番号(VT_I4)
戻り値	:	ワーク定義(VT_R8|VT_ARRAY)
		X, Y, Z, RX, RY, RZ, 属性

使用例

```
Dim vVal As Variant
vVal = caoRob.Execute( "GetWorkDef", 1)
Debug.Print"X= " & vVal(0) &", Y= " & vVal(1) &", Z= " & vVal(2)
Debug.Print"RX= " & vVal(3) &", RY= " & vVal(4) &", RZ= " & vVal(5)
Debug.Print"ATTR= " & vVal(6)
```

5.2.29.62. CaoRobot::Execute("SetWorkDef") コマンド

ワーク定義を設定します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 SetWorkDef (<WorkNo>, <WorkDef>,<WorkAttribute>)

WorkNo	:	[in]ワーク番号(VT_I4)
WorkDef	:	[in]ワーク定義(P 型 Fig は無視されます)
		X, Y, Z, RX, RY, RZ
WorkAttribute	:	[in]ワーク属性(VT_I4) (省略時は 0 になります)
		0:標準(デフォルト)
		1:固定工具(Fix)
戻り値	:	なし

使用例

```
caoRobot.Execute "SetWorkDef", Array(1, "P2",1)
caoRobot.Execute "SetWorkDef", Array(2, "P(100, 200, 300, 180, 0, 180)")
```

5.2.29.63. CaoRobot::Execute("WorkAttribute") コマンド

ワーク番号で指定したワーク属性を取得します.

書式 WorkAttribute (<WorklNo>)

WorkNo	:	[in]ワーク番号(VT_I4)
戻り値	:	ワーク属性(VT_I4)

使用例

```
Dim vVal As Variant
vVal = caoRob.Execute( "WorkAttribute", 1)
```

5.2.29.64. CaoRobot::Execute("GetAreaDef") コマンド

指定したエリア番号のエリアの定義を取得します.

書式　GetAreaDef (<AreaNo>)

引数　：[in]エリア番号(VT_I4)

戻り値　：エリア定義(VT_R8|VT_ARRAY)
X,Y,Z,RX,RY,RZ,DX,DY,DZ,IO,Position,Error,Time,DRX,DRY,
DRZ,Margin,Position1,Margin1,Position2,Margin2,Position3,
Margin3,Position4,Margin4,Position5,Margin5,Position6,Margin6,
Position7,Margin7,Position8,Margin8,Enable

使用例

```
Debug.Print caoRob.Execute( "GetAreaDef", 1)
```

5.2.29.65. CaoRobot::Execute("SetAreaDef") コマンド

エリアのパラメータを設定します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　SetAreaDef (<エリア番号>, <中心>, <大きさ>, <I/O 番号>, <変数格納番号>[,<エリア検知設定>])

書式 2　SetAreaDef (<エリア番号>, <エリア定義>)

引数　：書式1:
[in]エリア番号(VT_I4)
[in]中心点の位置と回転(傾き)(P 型)
[in]エリアの大きさ(V 型)
[in]I/O 番号(VT_I4)
[in]変数格納番号(VT_I4)
[in]エリア検知設定(VT_I4)
書式2:
[in]エリア定義(VT_R8|VT_ARRAY)
X,Y,Z,RX,RY,RZ,DX,DY,DZ,IO,Position[,Error,Time,DRX,DRY,
DRZ,Margin,Position1,Margin1,Position2,Margin2,Position3,
Margin3,Position4,Margin4,Position5,Margin5,Position6,Margin6,
Position7,Margin7,Position8,Margin8,Enable]

戻り値　：なし

使用例

```
caoRobot.Execute "SetAreaDef", Array(1, "P0", "V0", 24, 0, 0)
caoRobot.Execute "SetAreaDef", Array(2, "P(400, 250, 140, 180, 0, 180)",
                                        "V(200, 125, 70)", 24, 0, 0)
```

5.2.29.66. CaoRobot::Execute("SetArea") コマンド

エリアチェックを有効化します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　SetArea (<AreaNum>)

<AreaNum> : エリア番号 (VT_I4)
戻り値 : なし

使用例

```
caoRobot.Execute "SetArea", 1
```

5.2.29.67. CaoRobot::Execute("ResetArea") コマンド

エリアチェックを無効化します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　ResetArea (<AreaNum>)

<AreaNum> : エリア番号(VT_I4)
戻り値 : なし

使用例

```
caoRobot.Execute "ResetArea", 1
```

5.2.29.68. CaoRobot::Execute("AreaSize") コマンド

検知エリアの大きさ(各辺の長さ)をベクトル型で返します.

書式 AreaSize (<AreaNum>)

<AreaNum> : エリア番号(VT_I4)

戻り値 : エリアサイズ(VT_R8|VT_ARRAY)

X,Y,Z

使用例

```
Dim vVal As Variant
vVal = caoRob.Execute("AreaSize", 1 )    'Area1 のサイズを取得
Debug.Print"X=" & vVal(0) &", Y=" & vVal(1) &", Z=" & vVal(2)
```

5.2.29.69. CaoRobot::Execute("GetAreaEnabled") コマンド

エリアの有効/無効を取得します.

書式 GetAreaEnabled (<AreaNum>)

<AreaNum> : [in]エリア番号(VT_I4)

戻り値 : 有効/無効(VT_BOOL)

使用例

```
Debug.Print caoRob.Execute("GetAreaEnabled", 1 )    'Area1 の有効/無効を取得
```

5.2.29.70. CaoRobot::Execute("SetAreaEnabled") コマンド

エリアの有効/無効を設定します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 SetAreaEnabled (<AreaNum>, <有効/無効>)

<AreaNum> : [in]エリア番号(VT_I4)

<有効/無効> : [in]エリア番号(VT_BOOL)

戻り値 : なし

使用例

```
caoRob.Execute"SetAreaEnabled", Array( 1, True )    'Area1 を有効に
```

5.2.29.71. CaoRobot::Execute(“AddPathPoint”) コマンド

経路データに経路点を追加します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　AddPathPoint (<PathNum>, <Pose>)

< PathNum > : [in]経路番号 (1～20) [VT_I4]

<Pose> : [in] POSEDATA の値(P 型, J 型, T 型のいずれか)

戻り値 : なし

経路データに経路点を追加します.

指定した経路番号の経路データにすでに経路点があった場合は, そのデータの後ろに追加します.

本コマンドは NetwoRC プロバイダ(RC7 コントローラ用プロバイダ)の SetSplinePoint コマンドの互換コマンドとなります.

使用例

```
caoRobot.Execute "AddPathPoint", Array(2, "P(400, 250, 140, 180, 0, 180)")
```

5.2.29.72. CaoRobot::Execute(“ClrPathPoint”) コマンド

指定した経路の全ての経路点をクリアします.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　ClrPathPoint (<PathNum>)

< PathNum > : [in]経路番号 (1～20) [VT_I4]

戻り値 : なし

本コマンドは NetwoRC プロバイダ(RC7 コントローラ用プロバイダ)の ClrSplinePoint コマンドの互換コマンドとなります.

使用例

```
caoRobot.Execute "ClrPathPoint", 2
```

5.2.29.73. CaoRobot::Execute(“GetPathPoint”) コマンド

指定した経路点の位置データを返します.

書式　GetPathPoint (<PathNum>, < PointNum>)

< PathNum > : [in]経路番号 (1～20) [VT_I4]
< PointNum > : [in]経路点番号 (1～5000) [VT_I4]
戻り値 : [out]指定した経路点を P 型データで返します.

本コマンドは NetwoRC プロバイダ(RC7 コントローラ用プロバイダ)の GetSplinePoint コマンドの互換コマンドとなります.

使用例

```
Dim vntPos as Variant
vntPos = caoRobot.Execute("GetPathPoint", Array(2, 1))
```

5.2.29.74. CaoRobot::Execute(“LoadPathPoint”) コマンド

経路データを読み込みます.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　LoadPathPoint (<PathNum >)

< PathNum > : [in]経路番号 (1～20) [VT_I4]
戻り値 : なし

指定した経路番号の経路データをクリアし, 保存用メモリ内の経路データを読込みます.

PacScript 言語の LOADPATHPOINT 命令に対応します.

使用例

```
caoRobot.Execute "LoadPathPoint", 2
```

5.2.29.75. CaoRobot::Execute(“GetPathPointCount”) コマンド

指定した経路の経路点の個数を返します.

書式　GetPathPointCount (<PathNum >)

< PathNum >	:	[in]経路番号 (1～20)　[VT_I4]
戻り値	:	[out]指定した経路の経路点の個数 [VT_I4]

PacScript 言語の GETPATHPOINTCOUNT 命令に対応します.

使用例

```
Dim lCount As Long
lCount = caoRobot.Execute("GetPathPointCount", 2)
```

5.2.29.76. CaoRobot::Execute("GetRobotTypeName") コマンド

ロボット型式を取得します.

書式　GetRobotTypeName ( )

引数	:	なし
戻り値	:	ロボット型式(VT_BSTR)

使用例

```
Debug.Print caoRob.Execute("GetRobotTypeName" )
```

5.2.29.77. CaoRobot::Execute(“ArchMove”) コマンド

アーチモーションします.

PacScript 言語の ArchMove 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　ArchMove (<TgtPose>,<Hight>[,<ArchStartPos>[,<ArchEndPos>]])

< TgtPose >	:	[in] 目標位置(POSEDATA) (P 型, J 型, T 型のいずれか)
< Hight >	:	[in] 高さ[mm] (VT_R4)
< ArchStartPos >	:	[in] アーチ開始位置[mm] (VT_R4) 引数省略時は 0 指定として扱われます.
< ArchEndPos >	:	[in] アーチ完了位置[mm] (VT_R4) 引数省略時は 0 指定として扱われます.
戻り値	:	なし

使用例

```
caoRobot.Execute "ArchMove",Array( "P10",50,30,30 )
```

5.2.29.78. CaoRobot::Execute(“CrtMotionAllow”) コマンド

Move @C で, 停止判定に使用する, 手先の「停止位置精度, 姿勢精度」を変更します.

PacScript 言語の CrtMotionAllow 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　CrtMotionAllow (<True>,<Position>[,<Posture>])

書式 2　CrtMotionAllow (<False>)

< True/False >	:	[in]有効／無効　[VT_I4] 有効:True(0 以外) 無効:False(0)
< Position >	:	[in]位置精度[mm]　[VT_R4]
< Posture >	:	[in]姿勢精度[degree]　[VT_R4]
戻り値	:	なしs

使用例

```
caoRobot.Execute "CrtMotionAllow", Array(True, 1, 1 )
caoRobot.Move 1, "@C J2"
caoRobot.Execute "CrtMotionAllow", False
```

5.2.29.79. CaoRobot::Execute(“EncMotionAllow”) コマンド

Move @E で, 停止判定に使用する, ロボット軸の各軸の「停止時許容角度」を変更します.

PacScript 言語の EncMotionAllow 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　EncMotionAllow (<True>,<Angle>[,<Mode>])

書式 2　EncMotionAllow (<False>)

< True/False >	:	[in]有効／無効　[VT_I4] 有効:True（0 以外） 無効:False（0）
< Angle >	:	[in]許容角度　[VT_R4]
< Mode >	:	[in]モード　[VT_I4] 0（デフォルト）:角度［degree］／距離[mm] 1:パルス幅
戻り値	:	なし

使用例

```
caoRobot.Execute "EncMotionAllow", Array(True, 1, 1 )
caoRobot.Move 1, "@E J2"
caoRobot.Execute "EncMotionAllow", False
```

5.2.29.80. CaoRobot::Execute(“EncMotionAllowJnt”) コマンド

Move @E で, 停止判定に使用する, ロボット軸でない軸の「停止時許容角度」を変更します.

PacScript 言語の EncMotionAllowJnt 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　EncMotionAllowJnt (<True>,<Axis>,<Angle>[,<Mode>])

書式 2　EncMotionAllowJnt (<False>,<Axis>)

< True/False >	:	[in]有効／無効　[VT_I4] 有効:True（0 以外） 無効:False（0）
< Axis >	:	[in] 軸番号 (VT_I4)
< Angle >	:	[in]許容角度　[VT_R4]
< Mode >	:	[in]モード　[VT_I4] 0（デフォルト）:角度［degree］／距離[mm]

1:パルス幅

戻り値 : なし

使用例

```
caoRobot.Execute "EncMotionAllowJnt", Array(True, 7, 0.01, 1)
caoRobot.Move 1, "@E J2 EXA(7, 30.5)"
caoRobot.Execute "EncMotionAllowJnt", Array(False, 7)
```

5.2.29.81. CaoRobot::Execute(“ErAlw”) コマンド

偏差許容機能の設定と有効/無効を操作します.

PacScript 言語の ErAlw 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1 ErAlw (<True>,<Axis>,<Value>)

書式 2 ErAlw (<False>,<Axis>)

< True/False > : [in]有効／無効 [VT_I4]
有効:True(0 以外)
無効:False(0)

< Axis > : [in] 軸番号 (VT_I4)
無効の時のみ 0:全軸

< Value > : [in]設定値 [VT_R4]
回転軸:[degree]
直同軸:[mm]

戻り値 : なし

使用例

```
caoRobot.Execute "ErAlw", Array(True, 1, 0.01)
caoRobot.Execute "ErAlw", Array(True, 2, 0.01)
caoRobot.Execute "ErAlw", Array(False, 0)
```

5.2.29.82. CaoRobot::Execute(“ForceCtrl”) コマンド

力制御機能(コンプライアンス機能)の有効／無効を設定します.

PacScript 言語の ForceCtrl 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1 ForceCtrl (<True>,<CtrlNum>)

書式 2　ForceCtrl (<False>)

< True/False >	:	[in]無効／有効 [VT_I4] 有効:True(0 以外) 無効:False(0)
< CtrlNum >	:	[in]力制御番号(1～10)　[VT_I4]
< Mode>	:	[in]制御モード [VT_I4] 0:コンプライアンス機能 1:力センサ有コンプライアンス機能 省略時は力制御番号のテーブルで定義されている値で設定されます.
戻り値	:	なし

使用例

```
caoRobot.Execute "ForceCtrl", Array(True, 1)
caoRobot.Execute "ForceCtrl", False
```

5.2.29.83. CaoRobot::Execute(“ForceParam”) コマンド

力制御機能(コンプライアンス機能)のパラメータを設定します.

PacScript 言語の ForceParam 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　ForceParam (<CtrlNum>,<Coordinates>,<Force>,[<PosEralw>,[<Spring>,[<Damp>,
[<Mass>,[<CurLmt>,[<OffSet>,[<Eralw>,[<lMode>,[<Rate>,[<SpMax>,<RSpMax>]]]]]]]]])

< CtrlNum >	:	[in]力制御番号 (1～10)　[VT_I4]
< Coordinates >	:	[in]座標系　[VT_I4] 0:ベース座標系 1:ツール座標系 2:ワーク座標系
< Force >	:	[in]力　[P 型]
< PosEralw >	:	[in]位置偏差許容　[P 型]
< Spring >	:	[in]柔らかさ　[P 型]
< Damp >	:	[in]粘性　[P 型]
< Mass >	:	[in]慣性　[P 型]
< CurLmt >	:	[in]電流制限値　[J 型]

< OffSet >	:	[in]オフセット値　[P 型]
< Eralw >	:	[in]各軸偏差許容 [J 型]
<lMode>	:	[in]速度制御モード [VT_I4] (将来の予約領域です. 0 を設定してください)
<Rate>	:	[in]制御割合　[P 型]
<SpMax>	:	[in]最大並進速度　[VT_R8]
<RSpMax>		[in]最大回転速度　[VT_R8]
戻り値	:	なし

使用例

```
caoRobot.Execute "ForceParam", Array(1, 1, "P10")
```

5.2.29.84. CaoRobot::Execute(“ForceValue”) コマンド

要求データに応じた力制御の値を取得します.

書式　ForceValue (<DataNo> [,<Mode>])

< DataNo >	:	[in]データ番号　[VT_I4] 0:センサ値[N|Nm]制御座標系でのセンサ値を P 型で取得します. 1:センサ値[pulse]センサ出力値を P 型で取得します. 2:制御座標系での力とモーメントのプラスピーク[N|Nm] 3:制御座標系での力とモーメントのマイナスピーク[N|Nm] 4:力制御開始時からの制御座標系での手先位置の変位量(指令値)[mm|deg] 5:力制御開始時からの制御座標系での手先位置の変位量のプラスピーク(指令値)[mm|deg] 6:力制御開始時からの制御座標系での手先位置の変位量のマイナスピーク(指令値)[mm|deg] 7:力制御開始時からの制御座標系での手先位置の変位量(現在値)[mm|deg] 8:力制御開始時からの制御座標系での手先位置の変位量のプラスピーク(現在値)[mm|deg] 9:力制御開始時からの制御座標系での手先位置の変位量のマイナスピーク(現在値)[mm|deg] 10:力制御開始時からの制御座標系での指令値と力制御指令値の偏差[mm|deg]

11:力制御開始時からの制御座標系での指令値と力制御指令値の偏差のプラスピーク[mm|deg]
12:力制御開始時からの制御座標系での指令値と力制御指令値の偏差マイナスピーク[mm|deg]

< Mode > : [in]モード [VT_I4]
0:データを取得
-1:ピーク値をリセット
省略時は 0

戻り値 : [out] 力制御の値 [VT_VARIANT]
モード 0 時:データ番号に応じた力制御の値
モード-1 時:リセットする前の直前の値

使用例

```
vntVal = caoRobot.Execute("ForceValue", Array(1, 0))
```

5.2.29.85. CaoRobot::Execute(“ForceWaitCondition”) コマンド

力制御の指定した条件になるまで待機します.
b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 ForceWaitCondition ([<Position> [,<Force> [,<Time> [,<Mode> [,<Timeout>]]]]])

< Position > : [in]変位量 [P 型]
制御開始からの手先の変位量[mm], [deg]を設定します.
省略した場合は”-1”を指定したことになります.

<Force> : [in]力とモーメント [P 型]
力制御座標系に換算した力とモーメント[N], [Nm]を設定します.
省略した場合は”-1”を指定したことになります.

<Time> : [in]経過時間 [VT_I4]
制御開始からの経過時間[ms]を設定します.
省略した場合は”-1”を指定したことになります.

<Mode> : [in]終了モード
条件達成時のロボットと力制御の終了モードを設定します.
0:ロボット動作, 力制御は終了しない.
1:ロボット動作は瞬時停止, 力制御は終了しない.
2:ロボット動作, 力制御は終了.
省略した場合は”0”を指定したことになります.

<Timeout> : [in]タイムアウト [VT_I4]

タイムアウト時間[ms]を設定します.

使用例

```
caoRobot.Execute "ForceWaitCondition", Array("P0","P1")
```

5.2.29.86. CaoRobot::Execute(“ForceSensor”) コマンド

力覚センサの制御を行います.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 ForceSensor (<FuncNo>)

< FuncNo> : [in]機能番号 [VT_I4]
0:力覚センサのリセット

使用例

```
caoRobot.Execute "ForceSensor", 0
```

5.2.29.87. CaoRobot::Execute(“ForceChangeTable”) コマンド

力制御中の力制御テーブルを変更します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 ForceChangeTable (<TableNo>)

< TableNo > : [in]テーブル番号 (1～10) [VT_I4]

使用例

```
caoRobot.Execute "ForceChangeTable", 1
```

5.2.29.88. CaoRobot::Execute(“GetSrvData”) コマンド

ロボット軸のサーボ内部データを返します.

PacScript 言語の GetSrvData 命令に対応します.

書式 GetSrvData (<DataNum>)

< DataNum >	:	[in]データ番号 (1,2,4,5,7,8,17,18,19,20) [VT_I4]
戻り値	:	[out]指定したサーボ内部データ [J 型]
		1:モータ速度現在値 (rpm)
		2:モータ角度偏差 (mm or deg)
		4:モータ電流絶対値 (定格比%)
		5:モータトルク指令値(重力補償分は除く) (定格比%)
		7:負荷率 (%)
		8:各軸位置, 角度指令値 (mm or deg)
		17:ツール端速度[ワーク座標] (mm/s)
		※位置 3 成分のみ取得
		18:ツール端偏差[ワーク座標系] (mm)
		※位置 3 成分のみ取得
		19:ツール端速度[ツール座標] (mm/s)
		※位置 3 成分のみ取得
		20:ツール端偏差[ツール座標系] (mm)
		※位置 3 成分のみ取得

使用例

```
Dim vntVal As Variant
vntVal = caoRobot.Execute("GetSrvData", 2)
```

5.2.29.89. CaoRobot::Execute(“GetSrvJntData”) コマンド

指定軸のサーボ内部データを返します.

PacScript 言語の GetSrvJntData 命令に対応します.

書式 GetSrvJntData (<DataNum>,<Axis>)

< DataNum >	:	[in]データ番号 (1,2,4,5,8) [VT_I4]
< Axis >	:	[in] 軸番号 (VT_I4)
戻り値	:	[out]指定したサーボ内部データ [VT_R4]
		1:モータ速度現在値 (rpm)
		2:モータ角度偏差 (mm or deg)
		4:モータ電流絶対値 (定格比%)
		5:モータトルク指令値(重力補償分は除く) (定格比%)
		8:各軸位置, 角度指令値 (mm or deg)

使用例

```
Dim fData As Single
fData = caoRobot.Execute("GetSrvJntData", 2, 1)
```

5.2.29.90. CaoRobot::Execute(“GrvCtrl”) コマンド

重力補償制御機能の有効/無効を操作します.

PacScript 言語の GrvCtrl 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　GrvCtrl (<True/False>)

< True/False >	:	[in]有効／無効　[VT_I4] 有効:True(0 以外) 無効:False(0)
戻り値	:	なし

使用例

```
caoRobot.Execute "GrvCtrl", True
caoRobot.Execute "GrvCtrl", False
```

5.2.29.91. CaoRobot::Execute(“CurLmt”) コマンド

電流制限機能の設定と有効/無効を操作します.

PacScript 言語の CurLmt 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　CurLmt (<True>,<Axis>,<Value>)

書式 2　CurLmt (<False>,<Axis>)

< True/False >	:	[in]有効／無効　[VT_I4] 有効:True(0 以外) 無効:False(0)
< Axis >	:	[in] 軸番号 (VT_I4) 無効の時のみ 0:全軸
< Value >	:	[in]設定値[%]　[VT_I4]
戻り値	:	なし

使用例

```
caoRobot.Execute "GrvCtrl", True
caoRobot.Execute "CurLmt", Array(True, 1, 10.5)
caoRobot.Execute "CurLmt", Array(True, 2, 50.3)
caoRobot.Execute "CurLmt", Array(False, 0)
caoRobot.Execute "GrvCtrl", False
```

5.2.29.92. CaoRobot::Execute(“Zforce”) コマンド

H シリーズロボットの第 3 軸(Z 軸)の電流制限機能を推力で指定します.

PacScript 言語の Zforce 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 Zforce (<Thrust>)

< Thrust > : [in]推力 [VT_R4]

戻り値 : なし

使用例

```
caoRobot.Execute "GrvCtrl", True
caoRobot.Execute "Zforce", 50
caoRobot.Execute "GrvCtrl", False
```

5.2.29.93. CaoRobot::Execute(“GrvOffset”) コマンド

重力オフセット設定機能の有効/無効を操作します.

PacScript 言語の GrvOffset 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 GrvOffset (<True/False>)

< True/False > : [in]有効／無効 [VT_I4]
有効:True(0 以外)
無効:False(0)

戻り値 : なし

使用例

```
caoRobot.Execute "GrvOffset", True
caoRobot.Execute "GrvOffset", False
```

5.2.29.94. CaoRobot::Execute(“HighPathAccuracy”) コマンド

高軌跡制御機能の有効/無効を切り替えます.

PacScript 言語の HighPathAccuracy 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　HighPathAccuracy (<True/False>)

< True/False > : [in]有効／無効　[VT_I4]
有効:True(0 以外)
無効:False(0)

戻り値 : なし

使用例

```
caoRobot.Execute "HighPathAccuracy", True
caoRobot.Execute "HighPathAccuracy", False
```

5.2.29.95. CaoRobot::Execute(“MotionTimeout”) コマンド

動作命令のタイムアウト設定値を変更します.

PacScript 言語の MotionTimeout 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 1　MotionTimeout (<True>,<Timeout>)

書式 2　MotionTimeout (<False>)

< True/False > : [in]有効／無効　[VT_I4]
有効:True(0 以外)
無効:False(0)

< Timeout > : [in]タイムアウト時間[msec](1～30000)　[VT_I4]

戻り値 : なし

使用例

```
caoRobot.Execute "MotionTimeout", Array(True, 1000)
caoRobot.Execute "MotionTimeout", False
```

5.2.29.96. CaoRobot::Execute(“SingularAvoid”) コマンド

特異点回避機能を有効化, または無効化します.

PacScript 言語の SingularAvoid 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　SingularAvoid (<Mode>)

< Mode >	:	[in]モード　[VT_I4]
		0:無効
		2:有効
戻り値	:	なし

使用例

```
caoRobot.Execute "SingularAvoid", 2
caoRobot.Move 1, "@0 P2"
caoRobot.Execute "SingularAvoid", 0
```

5.2.29.97. CaoRobot::Execute(“SpeedMode”) コマンド

最適速度制御機能の設定を変更します.

PacScript 言語の SpeedMode 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式　SpeedMode (<Mode>)

< Mode >	:	[in]モード　[VT_I4]
		0:無効
		1:PTP のみ有効
		2:CP のみ有効
		3:PTP,CP 共に有効
戻り値	:	なし

使用例

```
caoRobot.Execute "SpeedMode", 1
```

5.2.29.98. CaoRobot::Execute(“PayLoad”) コマンド

内部負荷条件の設定値を変更します.

PacScript 言語の PayLoad 命令に対応します.

b-CAP Slave のライセンスキーを追加し, クライアントから b-CAP Slave Mode で制御中は使用できません.

書式 PayLoad (<Payload>[,<Gravity>[,<Inertia>]])

< Payload >	:	[in]先端負荷質量[g] [VT_I4]
< Gravity >	:	[in]負荷重心位置 [V 型] 引数省略時は 0 ベクトル(V(0,0,0))として扱われます.
< Inertia >	:	[in]負荷重心イナーシャ [V 型] 引数省略時は 0 ベクトル(V(0,0,0))として扱われます.
戻り値	:	なし

使用例

```
caoRobot.Execute "PayLoad", Array(2000, "V(0, 100, 150)" , "V(0, 10, 10)")
```

5.2.29.99. CaoRobot::Execute(“GenerateNonStopPath”) コマンド

無停止教示点補正機能オプションのコマンドです.

詳細は,「付録 D. 無停止教示点補正機能(外観検査軌道生成)」を参照ください.

書式 GenerateNonStopPath (<Teaching Points>, <Area Information>, <Teaching Point Number>, <Total Speed Rate>, <Convergence Coefficient>)

< Teaching Points >	:	[in]座標情報(教示点) [<座標情報> | VT_ARRAY]
< Area Information >	:	[in]エリア情報 [<エリア情報> | VT_ARRAY]
< Teaching Point Number >	:	[in]教示点数 [VT_I4]
<Total Speed Rate>	:	[in]総速度比 [VT_R8] 0.0～1.0 無停止動作全体の速度を変更する比率です. ロボットの外部速度に該当します.
<Convergence Coefficient>	:	[in]補正係数 [VT_R8] 0.0～1.0 動作点を収束計算で求める際の係数となります. 通常は, 0.7 を使用してください.
戻り値	:	[out]座標情報(動作点) [<座標情報> | VT_ARRAY]

使用例

```
vntMovePos = caoRobot.Execute( "GenerateNonStopPath", Array(vntTeachPos, vntAreaInfo,
Ubound(vntTeachPos) + 1, 100.0 * 0.01, 0.7))
```

5.2.29.100. CaoRobot::Execute("RobInfo") コマンド

ロボットの情報を返します.

PacScript 言語の RobInfo 命令に対応します.

書式 RobInfo (<lIndex>)

<lIndex> : インデックス番号(VT_I4)

戻り値 : [out] ロボット情報

インデックス番号	ロボット情報	データ型
0	ロボット型式毎にもっているユニークな数値	整数型
1	ロボット型式名	文字列
2	総動作距離	Joint 型
3	ロボット ID 番号	整数型

使用例

```
Dim RobotInfo as long
RobotInfo = caoRobot.Execute("RobInfo",0)
```

5.2.29.101. CaoRobot::Execute("SyncTimeStart") コマンド

複数台ロボットの同時発着動作の開始指示します.

書式 SyncTimeStart ( )

引数 : なし

戻り値 : なし

使用例

```
Dim caoRobot0 As CaoRobot
Dim caoRobot1 As CaoRobot

Set CaoRobot0 = caoCtrl.AddRobot( "Robot0" ," ID=0" )
Set CaoRobot1 = caoCtrl.AddRobot( "Robot1" ," ID=1" )

CaoRobot0.Execute  "SyncTimeStart"
   caoRobot0.Move 1, "P1"   'リーダロボットを P1 へ動作指示
   caoRobot1.Move 1, "P3"   'フォロワロボットを P3 へ動作指示
caoRobot0.Execute  "SyncTimeEnd"   'Robot0 と Robot1 を同時発着動作開始
```

5.2.29.102. CaoRobot::Execute("SyncTimeEnd") コマンド

複数台ロボットの同時発着動作を開始します.

書式　SyncTimeEnd (<lOption>)

< lOption >　:　動作オプション(VT_I4)
0:なし(省略時は 0 になります)
1:NEXT

戻り値　:　なし

使用例

```
Dim caoRobot0 As CaoRobot
Dim caoRobot1 As CaoRobot

Set CaoRobot0 = caoCtrl.AddRobot( “Robot0”,” ID=0” )
Set CaoRobot1 = caoCtrl.AddRobot( “Robot1”,” ID=1” )

CaoRobot0.Execute  “SyncTimeStart”
  caoRobot0.Move 1, “P1”  ‘リーダロボットを P1 へ動作指示
  caoRobot1.Move 1, “P3”  ‘フォロワロボットを P3 へ動作指示
caoRobot0.Execute  “SyncTimeEnd”,1  ‘Robot0 と Robot1 を NEXT オプションで同時発着動作開始
```

5.2.29.103. CaoRobot::Execute("SyncMoveStart") コマンド

複数台ロボットの協調動作の開始を指示します.

PTP 動作では使用できません.

書式　SyncMoveStart (<lFollowerRobotID> , <lFollowerRobotID>. . .)

<lFollowerRobotID>　:　フォロワ対象ロボットのロボット ID(VT_I4|VT_ARRAY)

戻り値　:　なし

使用例

```
Dim caoRobot0 As CaoRobot
Dim caoRobot1 As CaoRobot

Set CaoRobot0 = caoCtrl.AddRobot( “Robot0”,” ID=0” )
Set CaoRobot1 = caoCtrl.AddRobot( “Robot1”,” ID=1” )

CaoRobot0.Execute  “SyncMoveStart”,Array(1)
  caoRobot0.Move 2, “J(0,45,90,0,45,0,0,0)”  ‘リーダロボットを指定位置へ動作指示
caoRobot0.Execute  “SyncMoveEnd”  ‘Robot0 と Robot1 を協調動作開始
```

5.2.29.104. CaoRobot::Execute("SyncMoveEnd") コマンド

複数台ロボットの同時発着動作を開始します.

書式 SyncMoveEnd (<lOption>)

< lOption >	:	動作オプション(VT_I4)
		0:なし(省略時は 0 になります)
		1:NEXT
戻り値	:	なし

使用例

```
Dim caoRobot0 As CaoRobot
Dim caoRobot1 As CaoRobot

Set CaoRobot0 = caoCtrl.AddRobot( “Robot0”,” ID=0” )
Set CaoRobot1 = caoCtrl.AddRobot( “Robot1”,” ID=1” )

CaoRobot0.Execute “SyncMoveStart”,Aarray(1)
  caoRobot0.Move 2,“J3” ‘リーダロボットを J3 へ動作指示
caoRobot0.Execute “SyncMoveEnd”,1 ‘Robot0 と Robot1 を NEXT オプションで協調動作開始
```

5.2.29.105. CaoRobot::Execute("SetBaseDef") コマンド

Base 定義を設定します.

書式 SetBaseDef (<lBaseNo>, <BaseDef>, <lBaseAttribute>)

< lBaseNo >	:	Base 番号(VT_I4)
<BaseDef>	:	Base 定義(P 型)
		X, Y, Z, RX, RY, RZ
< lBaseAttribute >	:	現在未使用
戻り値	:	なし

使用例

```
caoRobot.Execcute “SetBaseDef”, Array(1,” P2” )
caoRobot.Execute “SetWorkDef”,Array(1,” P(100,200,300,180,0,180)” )
```

5.2.29.106. CaoRobot::Execute("GetBaseDef") コマンド

Base 番号で指定した Base 定義を取得します.

書式　GetBaseDef (<lBaseNo>)

< lBaseNo >	:	Base 番号(VT_I4)
戻り値	:	Base 定義(VT_R8|VT_ARRAY) X, Y, Z, RX, RY, RZ, 属性

使用例

```
Dim vVal As Variant
vVal = caoRobot.Execcute ( “GetBaseDef”,1)
```

5.2.29.107. CaoRobot::Execute(″SetHandIO″) コマンド

HandIO を設定します.

書式　SetHandIO (<lIONo>, <lValue>, <lRange>)

< lIONo >	:	IO の開始番号(VT_I4)
<lValue>	:	セットする値(VT_I4)
<lRange>	:	設定する範囲(VT_I4)
戻り値	:	なし

使用例

```
caoRobot.Execcute  “SetHandIO”,Array(48,8,4)
```

5.2.29.108. CaoRobot::Execute(″GetHandIO″) コマンド

HandIO を取得します.

書式　GetHandIO (<lIONo>, <lRange>)

< lIONo >	:	IO の開始番号(VT_I4)
<lRange>	:	設定する範囲(VT_I4)
戻り値	:	設定範囲の HandIO の値(VT_I4)

使用例

```
Dim lVal As Integer
lVal = caoRobot.Execcute ( “GetHandIO”,Array(48,,4))
```

5.2.29.109. CaoRobot::Execute("StartServoLog") コマンド

サーボログの記録を開始します.

書式　StartServoLog ( )

引数　：　なし
戻り値　：　なし

使用例

```
caoRobot.Execcute “StartServoLog”
```

5.2.29.110. CaoRobot::Execute("ClearServoLog") コマンド

サーボログの記録を消去します.

書式　ClearServoLog ( )

引数　：　なし
戻り値　：　なし

使用例

```
caoRobot.Execcute “ClearServoLog”
```

5.2.29.111. CaoRobot::Execute("StopServoLog") コマンド

サーボログの記録を停止します.

書式　StopServoLog ( )

引数　：　なし
戻り値　：　なし

使用例

```
caoRobot.Execcute “StopServoLog”
```

5.2.29.112. CaoRobot::Execute("GetCtrlLogMaxTime") コマンド

制御ログの記録時間を取得します.

書式　GetCtrlLogMaxTime ( )

引数　：　なし
戻り値　：　制御ログの記録時間(VT_I4)

使用例

```
Dim lVal As Integer
lVal = caoRobot.Execcute ( "GetCtrlLogMaxTime" )
```

5.2.29.113. CaoRobot::Execute("SetCtrlLogMaxTime") コマンド

制御ログの記録時間を設定します.

書式　SetCtrlLogMaxTime (lTime)

引数　：　設定する記録時間(VT_I4)
戻り値　：　なし

使用例

```
caoRobot.Execcute  "SetCtrlLogMaxTime",10
```

5.2.29.114. CaoRobot::Execute("GetCtrlLogInterval") コマンド

制御ログの記録間隔を取得します.

書式　GetCtrlLogInterval ( )

引数　：　なし
戻り値　：　制御ログの記録間隔(VT_I4)

使用例

```
Dim lVal As Integer
lVal = caoRobot.Execcute ( "GetCtrlLogInterval" )
```

5.2.29.115. CaoRobot::Execute("SetCtrlLogInterval ") コマンド

制御ログの記録間隔を設定します.

書式　SetCtrlLogInterval (lTime)

引数	:	設定する記録間隔(VT_I4)
戻り値	:	なし

使用例

```
caoRobot.Execcute “SetCtrlLogInterval”,8
```

5.2.29.116. CaoRobot::Execute("DetectOn") コマンド

Detect 機能を有効にします.

書式　DetectOn (<lPortNo>, <lVarType>, <lStartNo>, <lMaxNum>, <lResultVarNo>, <lMode>)

<lPortNo>	:	トリガとして入力する IO 信号のポート番号(VT_I4)
<lVarType>	:	格納するデータ型(VT_I4) 257:ポジション型 258:ジョイント型 259:同時変換型
<lStartNo>	:	格納するグローバル変数の先頭の Index 番号
<lMaxNum>	:	グローバル変数に格納するデータの個数
< lResultVarNo >	:	機能有効中にトリガとなった入力信号の回数を格納する整数型グローバル変数の Index 番号
<lMode>	:	入力信号を検知するエッジ 0:立ち上がりのみ(省略時) 1:立下がりのみ 2:両方
戻り値	:	なし

使用例

```
caoRobot.Execcute “DetectOn”,Array(8, 257, 2, 10, 11)
```

5.2.29.117. CaoRobot::Execute("DetectOff") コマンド

Detect 機能を無効にし, データを格納します.

書式　DetectOff (<lPortNo>, <lMode>)

<lPortNo>	:	機能を無効にするポート番号(VT_I4)
<lMode>	:	入力信号を検知するエッジ 0:立ち上がりのみ(省略時) 1:立下がりのみ 2:両方
戻り値	:	なし

使用例

```
caoRobot.Execcute  “DetectOff”,Array(8,0)
```

5.2.29.118. CaoRobot::Execute(″GetPluralServoData″) コマンド

サーボデータを一括で取得します.

書式　GetPluralServoData ()

引数	:	なし
戻り値	:	サーボデータ(VT_VARIANT|VT_ARRAY) モータ速度現在値, モータ角度偏差, モータ電流絶対値, モータトルク指令値, 各軸位置・角度指令値, ツール端速度, ツール端偏差

使用例

```
Dim vVal As Variant
vVal = caoRobot.Execcute ( “GetPluralServoData” )
```

5.2.29.119. CaoRobot::Execute(″AngularTrigger″) コマンド

ロボットが指定した角度(距離)動作するたびに I/O の ON/OFF を切替えます.

書式　AngularTrigger(<lVal>, <lJointNo>, <lPortNo>, <dwidth>, <lMode>, <dStartAngle>)

<lVal>	:	有効/無効 (VT_I4) True(0 以外):有効 False(0):無効
<lJointNo>	:	対象軸 (VT_I4)
<lPortNo>	:	ON/OFF させる I/O 番号 (VT_I4)

<dWidth> : 動作量 (VT_R8)
省略時は 0
回転軸:deg
直動軸:mm
< lMode > : 開始角度指定 (VT_I4)
0:開始角度指定なし(省略時)
1:開始角度指定あり
<dStartAngle> : 開始角度 (VT_R8)
省略時は現在角度または現在位置
戻り値 : なし

使用例

```
caoRobot.Execcute “AngularTrigger”,Array(True, 1, 24, 10)
```

5.2.29.120. CaoRobot::Execute(″VirtualFence″) コマンド

バーチャルフェンスの監視を開始または終了します.

書式 VirtualFence (<lVal>, <lModelID>)

<lVal> : 有効/無効 (VT_I4)
True(0 以外):有効
False(0):無効
<lModelID> : モデル ID (VT_I4)
戻り値 : なし

使用例

```
caoRobot.Execcute “VirtualFence”,Array(True, 1)
```

5.2.29.121. CaoRobot::Execute(″ExclusiveArea″) コマンド

排他エリアを作成, 変更します.

書式 ExclusiveArea (<lVal>, <Position>, <Size>)

<lVal> : 排他エリア番号 (VT_I4)
<Position> : 位置 (P 型)

<Size> : 大きさ (V 型)
戻り値 : なし

使用例

```
caoRobot.Execcute “ExclusiveArea”,Array(1, "P0", "V0")
```

5.2.29.122. CaoRobot::Execute("SetExclusiveArea") コマンド

排他エリアを有効にします.

書式 SetExclusiveArea (<lAreaNo >)

<lAreaNo> : エリア番号 (VT_I4)
戻り値 : なし

使用例

```
caoRobot.Execcute “SetExclusiveArea”, 1
```

5.2.29.123. CaoRobot::Execute("ResetExclusiveArea") コマンド

排他エリアを無効にします.

書式 ResetExclusiveArea (<lAreaNo >)

<lAreaNo> : エリア番号 (VT_I4)
戻り値 : なし

使用例

```
caoRobot.Execcute “ResetExclusiveArea”, 1
```

5.2.29.124. CaoRobot::Execute("ExclusiveControlStatus") コマンド

排他制御状態を取得します.

書式 ExclusiveControlStatus (<lMode>, <lCtrlNo>)

<lMode> : 取得したいモード (VT_I4)
0:排他制御有効状態

1:排他制御侵入状態
2:排他制御待機状態

<lCtrlNo> : 排他コントローラ番号 (VT_I4)

戻り値 : 指定モードの状態(VT_I4)
32 ビットのビット列の該当ビットにセットして返します
モードの状態が ON の場合は"1", OFF の場合は"0"を返します

使用例

```
Dim lVal As integer
lVal = caoRobot.Execute("ExclusiveControlStatus", Array(1, 2))
```

5.2.29.125. CaoRobot::Execute("GetAllSrvData") コマンド

サーボ内部データを一括で取得します.

書式 GetAllSrvData ()

引数 : なし

戻り値 : サーボ内部データ(VT_VARIANT | VT_ARRAY)
タイムスタンプ(us) : VT_R8
各軸位置, 角度指令値(mm or deg) : J 型(VT_R8 | VTARRAY)
各軸位置, 角度現在値(mm or deg) : J 型(VT_R8 | VTARRAY)
モータ角度偏差(mm or deg) : J 型(VT_R8 | VTARRAY)
モータ速度現在値(rpm) : J 型(VT_R8 | VTARRAY)
モータ電流絶対値(定格比 %) : J 型(VT_R8 | VTARRAY)
モータトルク指令値(定格比 %) : J 型(VT_R8 | VTARRAY)
負荷率(%) : J 型(VT_R8 | VTARRAY)
ツール端速度[ワーク座標](mm/s) : J 型(VT_R8 | VTARRAY)
※位置 3 成分のみ取得. それ以外の要素は 0
ツール端偏差[ワーク座標](mm) : J 型(VT_R8 | VTARRAY)
※位置 3 成分のみ取得. それ以外の要素は 0
ツール端速度[ツール座標](mm/s) : J 型(VT_R8 | VTARRAY)
※位置 3 成分のみ取得. それ以外の要素は 0
ツール端偏差[ツール座標](mm) : J 型(VT_R8 | VTARRAY)
※位置 3 成分のみ取得. それ以外の要素は 0

使用例

```
Dim vntVal As Variant
vntVal = caoRobot.Execute("GetAllSrvData")
```

5.2.29.126. CaoRobot::Execute("CurForceParam") コマンド

現在の力制御機能(コンプライアンス機能)のパラメータを取得します.

書式　CurForceParam (<<lNo>)

< lNo >　：　制御番号 (VT_I4)

戻り値　：　現在の力制御パラメータ(VT_VARIANT | VT_ARRAY)

座標系 : [VT_I4]
0:ベース座標系
1:ツール座標系
2:ワーク座標系
力 : [P 型]
位置偏差許容 : [P 型]
柔らかさ : [P 型]
粘性 : [P 型]
慣性 : [P 型]
電流制限値 : [J 型]
オフセット値 : [P 型]
各軸偏差許容 : [J 型]
速度制御モード : [VT_I4] (0 が取得されます)
制御割合 : [P 型]
最大並進速度 : [VT_R8]
最大回転速度 : [VT_R8]

使用例

```
Dim VntVal As Variant
caoRobot.Execute "TakeArm", Array(0, 0)
VntVal = caoRobot.Execute( “CurForceParam”,1)  ‘現在の力制御設定を取得
VntVal(1) = “P10”  ’力の値を変更
caoRobot.Execute  “ForcaParam”,Array(1,VntVal)  ‘力制御設定を更新
```

5.2.29.127. CaoRobot::Execute("GetLogCount") コマンド

制御ログデータの個数を取得します.

書式　GetLogCount ( )

引数　：なし
戻り値　：個数 (VT_I4)

使用例

```
Dim lLogCnt As Long
lLogCnt = caoRobot.Execute("GetLogCount")
```

5.2.29.128. CaoRobot::Execute("GetLogRecord") コマンド

制御ログデータの情報を取得します.

書式　GetLogRecord (<lIndex>)

<lIndex>　：制御ログデータのインデックス番号(VT_I4) 0～<個数-1>
戻り値　：制御ログデータ (VT_VARIANT[VT_VARIANT|VT_ARRAY:39 要素])
[0-7]:J1-J8 指令値 (VT_R8)
[8-15]:J1-J8 エンコーダ値 (VT_R8)
[16-23]: J1-J8 電流値 (VT_R8)
[24-31]: J1-J8 負荷率 (VT_R8)
[32]: ユーザーデータ (VT_R8)
[33]: トレースデータ (VT_I4)
[34]: プログラム名 (VT_BSTR)
[35]: 実行行番号 (VT_I4)
[36]: 経過時間(起動時間)[ms] (VT_I4)
[37]: Tool 番号 (VT_I4)
[38]: Work 番号 (VT_I4)

使用例

```
Dim lCnt As Long
lCnt = caoRobot.Execute("GetLogCount")
Dim i As Long
For i=0 To lCnt-1
  vDat = caoRobot.Execute("GetLogRecord", i)
   ' vDat(0) : J1 指令値, …, vDat(7) : J8 指令値
   ' vDat(8) : J1 エンコーダ値, …, vDat(15) : J8 エンコーダ値
   ' :
Next
```

5.2.29.129. CaoRobot::Execute("GetForceLogRecord") コマンド

力制御ログデータの情報を取得します.

書式 GetForceLogRecord (<lIndex>)

<lIndex> : 力制御ログデータのインデックス番号(VT_I4) 0～<個数-1>

戻り値 : 力制御ログデータ (VT_VARIANT[VT_VARIANT|VT_ARRAY:36 要素])
[0-6]:手先指令値 (VT_R8)
[7-13]:手先エンコーダ値 (VT_R8)
[14-19]: 力センサ値 (VT_R8)
[20-25]: 制御開始からの手先の変位量(指令値) (VT_R8)
[26-31]: 制御開始からの手先の変位量(エンコーダ値) (VT_R8)
[32]: 力制御有効状態 (VT_BOOL)
[33]: プログラム IC (VT_I4)
[34]: プログラム名 (VT_BSTR)
[35]: 実行行番号 (VT_I4)

使用例

```
Dim lCnt As Long
lCnt = caoRobot.Execute("GetLogCount")
Dim i As Long
For i=0 To lCnt-1
  vDat = caoRobot.Execute("GetForceLogRecord", i)
   ‘ vDat(0) ～ vDat(6) : 手先指令値
   ‘ vDat(7) ～ vDat(13) : 手先エンコーダ値
   ‘ :
Next
```

5.2.30. CaoRobot::Execute メソッド(COBOTTA 専用)

現在指定可能なコマンドの一覧を示します.

表 5-11 CaoRobot::Execute メソッド(COBOTTA 専用)のコマンド一覧

カテゴリ	コマンド名	機能	対応 Version	
動作準備 / 保守				
	AutoCal	起動時の CALSET を自動で実施する	2.8.0	P.163

		コマンドです		
	MotionPreparation	動作準備を自動で実行します	2.8.0	P.163
	GetMotionPreparationState	動作準備完了状態を取得します	2.8.0	P.164
ロボット 状態・値取得				
	GetMechaButtonState	ボタン状態を取得します	2.13.0	P.164
	ClearMechaButtonState	ボタンを離した回数のカウンタをリセットします	2.13.0	P.165

5.2.30.1. CaoRobot::Execute("AutoCal") コマンド

起動時の CALSET を自動で実施するコマンドです. PAC コマンドの AutoCal と同じ動作をします.
コマンドを使用するためには, [使用条件]パラメータの 252 番[起動時 CALSET]を[1:表示されない.]に設定し, TP での起動時 CALSET 画面の表示を止める必要があります. また起動権を Ethernet にする必要があります. AutoCal 詳細は COBOTTA のマニュアルを参照ください.

書式　AutoCal

引数　：なし
戻り値　：なし

使用例

```
'起動時の一連の処理
caoRobot.Execute "AutoCal"
caoRobot.Execute "Motionpreparation"
```

5.2.30.2. CaoRobot::Execute("MotionPreparation") コマンド

動作準備を自動で実行します. PAC コマンドの MotionPriparation と同じ動作をします.
※COBOTTA パラメータツールで COBOTTA にセーフティデータを送った直後は, パラメータの確認が必要であるため TP での動作準備が必要です. セーフティデータを送った直後このコマンドは実行できません.
MotionPreparation 詳細は COBOTTA のマニュアルを参照ください.

書式　MotionPreparation

引数　：なし
戻り値　：なし

使用例

```
'起動時の一連の処理
```

```
caoRobot.Execute “AutoCal”
caoRobot.Execute “Motionpreparation”
```

5.2.30.3. CaoRobot::Execute(″GetMotionPreparationState″) コマンド

動作準備完了状態を取得します. 動作準備完了状態でない場合, AutoCal 以外でロボット動作できません. 動作準備完了状態は, セーフティのエラー発生, 非常停止 ON, 防護停止 ON, セーフティパラメータの送信で, 未完了状態になります.

書式　GetMotionPreparationState

引数　：　なし
戻り値　：　完了状態/未完了状態(VT_BOOL)

使用例

```
‘エラー発生, 非常停止 ON, 防護停止 ON からの復帰処置

'非常停止, 防護停止解除する

'エラークリア
caoCtrl.Execute "ClearError"

'動作準備未完了なら動作準備をする
Dim vbState
vbState = caoRobot.Execute("GetMotionPreparationState")
If vbState <> True Then
    caoRobot.Execute "MotionPreparation"
End If
```

5.2.30.4. CaoRobot::Execute(″GetMechaButtonState ″) コマンド

COBOTTA のアーム上のボタンを押した回数と, 現在のボタンの状態を取得します. ボタンを押した回数は, ClearMechaButtonState コマンド実行時と COBOTTA の電源を切ったときにリセットされます.

書式　GetMechaButtonState< lBtnType>

< lBtnType >　：　状態を取得するボタン種類 (VT_I4)

0 : ファンクションボタン
1 : ハンドプラスボタン
2 :ハンドマイナスボタン

戻り値 : ボタン状態の配列(VT_I4 | VT_ARRAY)
[0] : ボタンを押した回数
※ボタンを押した回数は，ボタンを離したタイミングでカウントされます.
[1] : ボタンの状態
0 : ボタンが押されていない
1 : ボタンが押されている

使用例

```
Dim vntRet As Variant
Dim ICnt As Integer
Dim IState As Integer
vntRet = caoRob.Execute("GetMechaButtonState", 0)  ' ファンクションボタンを指定
ICnt = vntRet(0)      ' 押した回数
IState = vntRet(1)    ' 今のボタン状態
```

5.2.30.5. CaoRobot::Execute("ClearMechaButtonState") コマンド

COBOTTA のアーム上のボタンが押された回数をクリアします. ボタンを押した回数は，COBOTTA の電源を切ったときにもクリアされます.

書式 ClearMechaButtonState < lBtnType>

< lBtnType > : 回数をクリアするボタン種類 (VT_I4)
0 : ファンクションボタン
1 : ハンドプラスボタン
2 :ハンドマイナスボタン

戻り値 : なし

使用例

```
caoRob.Execute("ClearMechaButtonState", 0)      ' ファンクションボタンを指定し，クリア
```

5.2.31. CaoTask::AddVariable メソッド

CaoTask クラスの AddVariable メソッドの引数は，システム変数名を指定します．

実装されているシステム変数の一覧は表 5-18 を参照して下さい．

5.2.32. CaoTask::get_VariableNames プロパティ

AddVariable メソッドで指定できる変数名とシステム変数名の一覧を取得します．

5.2.33. CaoTask::Start メソッド

オブジェクトに対応している PAC プログラムを実行します．

以下に，Start の仕様を示します．

書式 Start <lMode:LONG>[, <bstrOpt:BSTR>]

lMode	:	[in]	開始モード 1:1 サイクル実行，2:連続実行，3:ステップ送り
bstrOpt	:	[in]	オプション（未使用）

5.2.34. CaoTask::Stop メソッド

オブジェクトに対応している PAC プログラムを停止します．

以下に，Stop の引数仕様を示します．

書式 Stop <lMode:LONG>[, <bstrOpt:BSTR>]

lMode	:	[in]	停止モード 0:デフォルト停止，1:瞬時停止，2:ステップ停止，3:サイクル停止，4:初期化停止
bstrOpt	:	[in]	オプション（未使用）

停止モードで"0:デフォルト停止"を指定したときは，"1:瞬時停止"として扱われます．

5.2.35. CaoTask::Execute メソッド

コマンドを実行します．

Execute メソッドの引数は，コマンドを BSTR，パラメータを VARIANT 配列で指定します．

書式 [<vntRet:VARIANT> =] Execute(<bstrCmd:BSTR > [,<vntParam:VARIANT>])

bstrCmd	:	[in]	コマンド名
vntParam	:	[in]	パラメータ
vntRet		[out]	返り値

実行時バインディングの機能を使って本クラスに定義していないメソッドを呼び出した場合，次の仕様で Execute メソッドが自動的に呼ばれます．

vntRet = Obj.CommandName(Param1, Param2, ...)

↓

vntRet = Obj.Execute("CommandName", Array(Param1, Param2, ...))

1.第一引数にコマンド名が BSTR 文字列で渡る

2.第二引数に全パラメータが VARIANT 配列で渡る

使用例

```
Dim vRes As Variant
Dim caoTsk As CaoTask

Set caoTsk = caoCtrl.AddTask("pro1" )
vRes = caoTsk.Execute("GetStatus" )    'タスクの状態取得
```

現在指定可能なコマンドの一覧を示します.

表 5-12　CaoTask::Execute メソッドのコマンド一覧

カテゴリ	コマンド名	機能	
タスク状態			
	GetStatus	タスクの状態取得	P. 167
プライオリティ			
	GetThreadPriority	優先度(プライオリティ)の取得	P. 168
	SetThreadPriority	優先度(プライオリティ)の設定	P. 168

5.2.35.1. CaoTask::Execute("GetStatus") コマンド

タスクの状態を取得します.

書式　GetStatus()

引数　：なし

戻り値　：状態(VT_I4)

0:TASK_NON_EXISTENT,	タスクが存在しない
1:TASK_SUSPEND,	一時停止中
2:TASK_READY,	レディ
3:TASK_RUN,	実行中
4:TASK_STEPSTOP,	ステップ停止

使用例

```
Dim lStatus As Long
```

```
lStatus = caoTsk.Execute("GetStatus" )
```

5.2.35.2. CaoTask::Execute("GetThreadPriority") コマンド

タスクの実行優先度を取得します.

書式 GetThreadPriority( )

引数	:	なし
戻り値	:	優先度(VT_I4) 2:THREAD_PRIORITY_HIGHEST 1:THREAD_PRIORITY_ABOVE_NORMAL 0:THREAD_PRIORITY_NORMAL -1:THREAD_PRIORITY_BELOW_NORMAL -2:THREAD_PRIORITY_LOWEST

5.2.35.3. CaoTask::Execute("SetThreadPriority") コマンド

タスクの実行優先度を設定します.

書式 SetThreadPriority([<lPriority>])

<lPriority>	:	優先度(VT_I4) 2:THREAD_PRIORITY_HIGHEST 1:THREAD_PRIORITY_ABOVE_NORMAL 0:THREAD_PRIORITY_NORMAL -1:THREAD_PRIORITY_BELOW_NORMAL -2:THREAD_PRIORITY_LOWEST 引数省略時は 0 指定として扱われます.
戻り値	:	なし

5.2.36. CaoVariable::get_Value プロパティ

オブジェクトに対応している変数の値を取得します.

変数の実装状況およびデータ型は「5.3 変数一覧」を参照して下さい.

5.2.37. CaoVariable::put_Value プロパティ

オブジェクトに対応している変数に値を設定します.

変数の実装状況およびデータ型は「5.3 変数一覧」を参照して下さい.

5.2.38. CaoExtension::Execute メソッド

拡張機能のコマンドを実行します.

Execute メソッドの引数は, コマンドを BSTR, パラメータを VARIANT 配列で指定します.

書式 [<vntRet:VARIANT> =] Execute(<bstrCmd:BSTR > [,<vntParam:VARIANT>])

bstrCmd	:	[in]	コマンド名
vntParam	:	[in]	パラメータ
vntRet		[out]	返り値

実行時バインディングの機能を使って本クラスに定義していないメソッドを呼び出した場合, 次の仕様で Execute メソッドが自動的に呼ばれます.

vntRet = Obj.CommandName(Param1, Param2, ...)

↓

vntRet = Obj.Execute(“CommandName”, Array(Param1, Param2, ...))

1.第一引数にコマンド名が BSTR 文字列で渡る

2.第二引数に全パラメータが VARIANT 配列で渡る

使用例 Hand オブジェクトの操作

```
Dim caoExt As CaoExtension

Set caoExt = caoCtrl.AddExtension( “Hand0” )
CaoExt.Execute “Motor”, true ‘電動ハンドモータ
caoExt.Execute “Org”                    ‘原点復帰

caoExt.Execute “Chuck”, 0     ‘チャック動作実行

caoExt.Execute “UnChuck”, 1   ‘アンチャック動作実行
```

使用例 K3Hand オブジェクトの操作

```
Dim caoExt As CaoExtension

Set caoExt = caoCtrl.AddExtension( “K3Hand” )
caoExt.Motor, true     ‘サーボ ON
caoExt.Speed, 50       ‘速度設定

caoExt.MoveJ, ″J(0,10,0,0)″     ‘’ サーボ 0,サーボ 2,サーボ 3 が 0 度に,サーボ 1 が 10 度に移動
```

現在指定可能なコマンドの一覧を示します.

表 5-13 CaoExtension::Execute メソッドのコマンド一覧

5.2.38.1. Hand オブジェクト– CaoExtension::Execute(“Chuck”) コマンド

指定ポイントデータでチャック動作をします.

書式　Chuck(<No>)

引数　：　No　[in]ポイント番号(0～31)　[VT_I4]
戻り値　：　なし

指定したポイントデータの設定に従い，ワークの把持動作をします．
あらかじめポイントデータに把持動作を設定してください．
動作状態が動作中(get_BusyState が 0 以外)の場合，実行することはできません．

使用例

```
caoExt.Execute “Chuck”, 0
```

5.2.38.2. Hand オブジェクト– CaoExtension::Execute(“UnChuck”) コマンド

指定ポイントデータでアンチャック動作をします．

書式　UnChuck(<No>)

引数　：　No　[in]ポイント番号(0～31)　[VT_I4]
戻り値　：　なし

指定したポイントデータの設定に従い，把持状態から電動ハンドを所定の位置まで移動させます．
あらかじめポイントデータには移動動作を設定してください．
動作状態が動作中(get_BusyState が 0 以外)の場合，実行することはできません．

使用例

```
caoExt.Execute “UnChuck”, 1
```

5.2.38.3. Hand オブジェクト– CaoExtension::Execute(“Motor”) コマンド

モータ電源を入/切します．

書式　Motor　(<State>)

引数　：　State　[in]モータ状態　[VT_I4]
　　0:モータ Off
　　0 以外:モータ On
戻り値　：　なし

電動ハンドのモータを On または Off します．電動ハンドが非常停止中はモータ On コマンドを実行してもモータ On 状態になりません．また，電動ハンドがすでにモータ On 状態のときに，モータ On コマンドを実行し

ても電動ハンドはモータ On のままです.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “Motor”, 1
```

5.2.38.4. Hand オブジェクト- CaoExtension::Execute(“Org”) コマンド

原点復帰を行います.

書式 Org( )

引数 : なし

戻り値 : なし

原点復帰を行います.

電動ハンドの電源投入後, 一度はこのコマンドを実行する必要があります. またエラーが発生した場合もエラー解除後に原点復帰が必要となります.

原点復帰が未完了の状態で電動ハンドの動作コマンドを実行するとエラーとなります.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “Org”
```

5.2.38.5. Hand オブジェクト- CaoExtension::Execute(“MoveP”) コマンド

ポイント動作します.

書式 MoveP (<No>)

引数 : No [in]ポイント番号(0～31) [VT_I4]

戻り値 : なし

指定したポイントデータの設定に従い, ハンド動作を実行します.

あらかじめポイントデータに動作を設定してください.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “MoveP” , 1
```

5.2.38.6. Hand オブジェクト- CaoExtension::Execute(“MoveA”) コマンド

絶対位置移動動作します.

書式　MoveA (<Pos>, <Speed>)

引数	:	Pos	[in]位置(-999.90～999.90 [mm])	[VT_R4]
	:	Speed	[in]速度(20～100[%])	[VT_I4]
戻り値	:	なし		

指定した位置, 指定した速度で電動ハンドを絶対位置移動動作させます.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “MoveA” , Array(5.00, 20)
```

5.2.38.7. Hand オブジェクト- CaoExtension::Execute(“MoveR”) コマンド

絶対位置移動動作します.

書式　MoveR (<Pos>, <Speed>)

引数	:	Pos	[in]位置(-999.90～999.90 [mm])	[VT_R4]
	:	Speed	[in]速度(20～100[%])	[VT_I4]
戻り値	:	なし		

指定した位置, 指定した速度で電動ハンドを相対位置移動動作させます.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “MoveR” , Array(-3.00, 100)
```

5.2.38.8. Hand オブジェクト- CaoExtension::Execute(“MoveAH”) コマンド

加減速付き絶対位置把持動作をします.

書式　MoveAH (<Pos>, <Speed>, <Force>)

引数	:	Pos	[in]位置(-999.90～999.90 [mm])	[VT_R4]

: Speed [in]速度(20～100[%]) [VT_I4]
: Force [in]把持力(30～100[%]) [VT_I4]

戻り値 : なし

指定した位置，速度，把持力で電動ハンドを絶対位置移動把持動作させます.
動作状態が動作中(get_BusyState が 0 以外)の場合，実行することはできません.

使用例

```
caoExt.Execute “MoveAH” , Array(2.50, 100, 100)
```

5.2.38.9. Hand オブジェクト- CaoExtension::Execute(“MoveRH”) コマンド

加減速付き相対位置把持動作をします.

書式 MoveRH (<Pos>, <Speed>, <Force>)

引数 : Pos [in]位置(-999.90～999.90 [mm]) [VT_R4]
: Speed [in]速度(20～100[%]) [VT_I4]
: Force [in]把持力(30～100[%]) [VT_I4]

戻り値 : なし

指定した位置，速度，把持力で電動ハンドを相対位置移動把持動作させます.
動作状態が動作中(get_BusyState が 0 以外)の場合，実行することはできません.

使用例

```
caoExt.Execute “MoveRH” , Array(2.50, 100, 100)
```

5.2.38.10. Hand オブジェクト- CaoExtension::Execute(“MoveH”) コマンド

定速移動で把持動作をします.

書式 MoveH (<Speed>, <Force>, <Direct>)

引数 : Speed [in]速度(20～50[%]) [VT_I4]
: Force [in]把持力(30～100[%]) [VT_I4]
: Direct [in]移動方向 [VT_I4]
0:開方向
0 以外:閉方向

戻り値 : なし

指定した速度，把持力，移動方向で電動ハンドを定速移動把持動作させます.
動作状態が動作中(get_BusyState が 0 以外)の場合，実行することはできません.

使用例

```
caoExt.Execute “MoveH” , Array(50, 100, 1)
```

5.2.38.11. Hand オブジェクト- CaoExtension::Execute(“MoveZH”) コマンド

ゾーン付き定速移動で把持動作をします.

書式 MoveZH (<Speed>, <Force>, <Direct>)

引数	:	ZON1 [in]ZON 範囲 1(-999.90～999.90 [mm])	[VT_R4]
	:	ZON2 [in]ZON 範囲 2 (-999.90～999.90 [mm])	[VT_R4]
	:	Speed [in]速度(20～50[%])	[VT_I4]
	:	Force [in]把持力(30～100[%])	[VT_I4]
	:	Direct [in]移動方向	[VT_I4]
		0:開方向	
		0 以外:閉方向	
戻り値	:	なし	

指定した ZON 範囲, 速度, 把持力, 移動方向で電動ハンドを定速移動把持動作させます.

ZON 範囲 1, ZON 範囲 2 の範囲内に入ると get_ZonState が範囲内(0 以外)になります.

動作状態が動作中(get_BusyState が 0 以外)の場合, 実行することはできません.

使用例

```
caoExt.Execute “MoveZH” , Array(1.00, 4.00, 50, 100, 1)
```

5.2.38.12. Hand オブジェクト- CaoExtension::Execute(“Stop”) コマンド

動作停止します.

書式 Stop ()

引数	:	なし
戻り値	:	なし

電動ハンドが動作中にこのコマンドを実行すると, 瞬時に動作を停止します.

使用例

```
caoExt.Execute “Stop”
```

5.2.38.13. Hand オブジェクト- CaoExtension::Execute(“CurPos”) コマンド

現在位置を返します.

書式　CurPos　()

引数　：　なし

戻り値　：　[out]　現在位置[mm]　[VT_R4]

電動ハンドの現在位置[mm]を返します.

タイミングにより最大 10ms かかることがあります.

使用例

```
Dim handPos as Single
handPos = caoExt.Execute(  “CurPos”  )
```

5.2.38.14. Hand オブジェクト- CaoExtension::Execute(“GetPoint”) コマンド

ポイントデータの要素を返します.

書式　GetPoint　(<No>, <Index>)

引数　：　No　[in]ポイント番号(0～31)　[VT_I4]

：　Index　[in]ポイントデータの要素(0～5)　[VT_I4]

戻り値　：　[out]　指定したポイントデータの指定要素の値

0 ： 動作モード

1 ： 移動量 [mm]　「VT_R4]

2 ： 速度 [mm]　[VT_I4]

3 ： 把持力 [%]　[VT_I4]

4 ： ZON 範囲 1　[mm]　[VT_R4]

5 ： ZON 範囲 2　[mm]　[VT_R4]

指定したポイントデータの指定要素の値を返します.

使用例

```
Dim Speed as Long
Speed = caoExt.Execute(  “GetPoint”, Array(0, 2) )
```

5.2.38.15. Hand オブジェクト– CaoExtension::Execute(“get_EmgState”) コマンド

非常停止入力状態を表します.

書式　get_EmgState ()

引数　：なし

戻り値　：非常停止入力状態　[VT_I4]

0:非常停止状態

0 以外:非常停止が解除されている状態

(非常停止入力が短絡)

電動ハンドの非常停止状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_EmgState” )
```

5.2.38.16. Hand オブジェクト– CaoExtension::Execute(“get_ZonState”) コマンド

電動ハンドが設定された範囲内に位置しているかどうかの状態を表します.

書式　get_EmgState ()

引数　：なし

戻り値　：ZON 状態　[VT_I4]

0:範囲指定の外に位置している

0 以外:範囲指定 1 から範囲指定 2 の間に位置している

電動ハンドが設定された範囲内に位置しているかどうかの状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_ZonState” )
```

5.2.38.17. Hand オブジェクト– CaoExtension::Execute(“get_OrgState”) コマンド

原点復帰状態を表します.

書式　get_OrgState ()

引数　：なし

戻り値　：原点復帰状態　[VT_I4]

0:原点復帰が完了していない(原点未了)

0 以外:原点復帰が完了している

原点復帰状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_OrgState” )
```

5.2.38.18. Hand オブジェクト- CaoExtension::Execute(“get_HoldState”) コマンド

電動ハンドの把持状態を表します.

書式 get_HoldState ()

引数 : なし

戻り値 : 把持状態 [VT_I4]

0:把持していない

0 以外:ワークを設定した把持力で把持している

電動ハンドの把持状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_HoldState” )
```

5.2.38.19. Hand オブジェクト- CaoExtension::Execute(“get_InposState”) コマンド

目標位置に入っているかどうか(INPOS 状態)を表します.

書式 get_InposState ()

引数 : なし

戻り値 : INPOS 状態 [VT_I4]

0:目標位置の範囲外, または移動中

0 以外:原点復帰, 位置決め動作後, 目標位置の範囲内

目標位置に入っているかどうか(INPOS 状態) を返します.

目標位置の範囲はパラメータの「位置決め完了距離」によって決まります.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_InposState” )
```

5.2.38.20. Hand オブジェクト- CaoExtension::Execute(“get_Error”) コマンド

電動ハンドのエラー状態を表します.

書式　get_Error ()

引数　: なし

戻り値　: エラーコード(10 進形式データ)　[VT_I4]

0:正常な状態

0 以外:エラー発生状態. 値はエラーコードを表します.

電動ハンドのエラー状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_Error” )
```

5.2.38.21. Hand オブジェクト- CaoExtension::Execute(“get_BusyState”) コマンド

動作状態を表します.

書式　get_BusyState ()

引数　: なし

戻り値　: 動作状態　[VT_I4]

0:動作コマンドの受け付けが可能

0 以外:動作中. 動作コマンドが入りその信号を受け付けたとき

電動ハンドの動作状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_BusyState” )
```

5.2.38.22. Hand オブジェクト- CaoExtension::Execute(“get_ MotorState”) コマンド

モータ電源状態を表します.

書式　get_MotorState ()

引数　: なし

戻り値　: モータ電源状態　[VT_I4]

0:モータ電源 Off

0 以外:モータ電源 On

電動ハンドのモータ電源状態を返します.

使用例

```
Dim State as Long
State = caoExt.Execute( “get_MotorState” )
```

5.2.38.23. K3Hand オブジェクト- CaoExtension::Execute(“Motor”) コマンド

サーボの ON/OFF を設定します． K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式　Motor <vbEnable>

vbEnable	:	[in] 設定するサーボ状態 (VT_BOOL) True：全サーボ ON False：全サーボ OFF
戻り値	:	なし

使用例

```
caoExt.Motor True
```

5.2.38.24. K3Hand オブジェクト- CaoExtension::Execute(“Speed”) コマンド

動作速度を設定します． K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式　Speed <sngSpeed>

sngSpeed	:	[in] 動作速度 [%](VT_R4)
戻り値	:	なし

使用例

```
caoExt.Speed 100
```

5.2.38.25. K3Hand オブジェクト- CaoExtension::Execute(“MoveJ”) コマンド

目標位置までハンドを動作させます. 引数は lPointNo, bstrEachPose, bstrAllPose の書式のうちどれか 1 つを設定します． K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式　MoveJ (<lPointNo> or <bstrEachPose> or <bstrAllPose>)

lPointNo : [in] 教示したポイントの番号[0 ～ 99] (VT_I4)

bstrEachPose : [in] サーボ毎の目標位置[deg] (VT_BSTR)

・サーボ番号は 0 番開始

・サーボ番号毎に目標位置を指定して()で括る

・サーボ番号を複数記載する際はカンマで区切る

※サーボ 1 を 50 度, サーボ 4 を-30 度に動かす場合の指定例

"(1,50),(4,-30) "

bstrAllPose : 連続したサーボの目標位置[deg] (VT_BSTR)

・サーボ 0 から順番に目標位置を指定する

・連続して目標位置を指定する場合はカンマで区切る

・閉じ括弧までの合計サーボ数が最大数を超えた場合エラー

・先頭に大文字の"J"を付ける

※サーボ 5 までを全て 0 度に動かす場合の指定例

"J(0,0,0,0,0,0) "

戻り値 : なし

使用例

```
caoExt.MoveJ 1                          ' ポイント 1 のデータに従ってサーボが移動
caoExt.MoveJ "(0,50),(7,-50)"           ' サーボ 0 を 50 度, サーボ 7 を-50 度に移動
caoExt.MoveJ "(0,10,0,0)"               ' サーボ 0, サーボ 2, サーボ 3 が 0 度に, サーボ 1 が 10 度に移動
```

5.2.38.26. K3Hand オブジェクト- CaoExtension::Execute("BusyState") コマンド

動作状態を取得します． K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式 BusyState

戻り値 : 動作状態(VT_BOOL)

True:動作中

False:停止中

使用例

```
Dim vntBusy As Variant
vntBusy = caoExt.BusyState
```

5.2.38.27. K3Hand オブジェクト- CaoExtension::Execute("CurPos") コマンド

現在位置を取得します． K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式　CurPos

戻り値　：　各サーボの現在位置(VT_ARRAY|VT_R8)
サーボ合計数分の現在位置[deg]の配列が返却される

使用例

```
Dim vntPos As Variant
vntPos = caoExt.CurPos
```

5.2.38.28. K3Hand オブジェクト- CaoExtension::Execute(“DestPos”) コマンド

現在指令値を取得します． K3 ハンドが装着された COBOTTA でのみ使用可能です．

書式　CurPos

戻り値　：　各サーボの現在指令値(VT_ARRAY|VT_R8)
サーボ合計数分の現在指令値[deg]の配列が返却される

使用例

```
Dim vntPos As Variant
vntPos = caoExt.DestPos
```

5.2.38.29. K3Hand オブジェクト- CaoExtension::Execute(“GetParam”) コマンド

指定したパラメータの値を取得します． K3 ハンドが装着された COBOTTA でのみ使用可能です．

書式　GetParam <lIndex>

lIndex　：　[in] 取得するパラメータのインデックス (VT_I4)
戻り値　：　取得パラメータ (VT_VARIANT)

使用例

```
Dim vntRet As Variant
vntRet = caoExt.GetParam(1)
```

5.2.38.30. K3Hand オブジェクト- CaoExtension::Execute(“SetParam”) コマンド

指定したパラメータの値を設定します． K3 ハンドが装着された COBOTTA でのみ使用可能です．

書式　SetParam <lIndex>, <vntParam>

lIndex　：　[in] 取得するパラメータのインデックス (VT_I4)
vntParam　：　[in] 設定パラメータ
戻り値　：　なし

使用例

```
caoExt.SetParam 1, 10
```

SetParam 及び GetParam で操作可能なパラメータを以下の表に示します.

表 5-14　K3Hand パラメーター覧

INDEX	パラメータ名	データ型	説明	属性	
				Get	Set
0	サーボ状態	VT_I4	各サーボの ON/OFF をビット列で操作	○	○
1	動作状態	VT_BOOL	動作状態を取得	○	
2	動作速度	VT_R4	動作速度を操作	○	○
3	サーボ合計数	VT_I4	サーボ合計数を取得	○	
4	動作範囲	VT_VARIANT | VT_ARRAY	各サーボの可動範囲を取得 正方向 VT_R4 | VT_ARRAY サーボ合計数分 負方向 VT_R4 | VT_ARRAY サーボ合計数分	○	
5	ハンドタイプ	VT_I4	ハンドのマイナータイプを取得	○	
6	ハードリビジョン	VT_I4	ハンドのハードリビジョンを取得	○	
7	サーボ電流値	VT_R4 | VT_ARRAY	各サーボの電流値を取得	○	
8	バス電圧	VT_R4	バス電圧を取得	○	
9	バス電流	VT_R4	バス電流を取得	○	

5.2.38.31. K3Hand オブジェクト- CaoExtension::Execute(“GetPoint”) コマンド

ポイントデータの値を取得します. ポイントデータは各サーボの位置を教示するために使用できるデータ領域で, ハンド画面もしくは SetPoint コマンドで設定可能です. K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式　GetPoint <lIndex>

lIndex : [in] ポイント番号 0 ～ 99 (VT_I4)
戻り値 : 保存されている各軸位置(VT_R4 | VT_ARRAY)

使用例

```
Dim vntRet As Variant
vntRet = caoExt.GetPoint(1)    ' ポイント番号 1 のデータを取得
```

5.2.38.32. K3Hand オブジェクト- CaoExtension::Execute(“SetPoint”) コマンド

ポイントデータの値を設定します. ポイントデータは各サーボの位置を教示するために使用できるデータ領域で, ハンド画面もしくは SetPoint コマンドで設定可能です. K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式 SetPoint <lIndex>, <vntData>

lIndex : [in] ポイント番号 0 ～ 99 (VT_I4)
vntData : [in] ポイントデータ (VT_R4 | VT_ARRAY)
サーボ合計数分の各軸位置のデータ配列
戻り値 : なし

使用例

```
caoExt.SetPoint 1, Array(0,0,0,0,0,0,0,0)        ' ポイント番号 1 のデータを設定
```

5.2.38.33. K3Hand オブジェクト- CaoExtension::Execute(“SavePoint”) コマンド

ポイントデータの値を保存します. ポイントデータは各サーボの位置を教示するために使用できるデータ領域で, ハンド画面もしくは SetPoint コマンドで設定可能です. K3 ハンドが装着された COBOTTA でのみ使用可能です.

書式 SavePoint <lIndex>

lIndex : [in] ポイント番号 0 ～ 99 (VT_I4)
戻り値 : なし

使用例

```
caoExt.SavePoint 1      ' ポイント番号 1 のデータを保存
```

5.3. 変数一覧

5.3.1. コントローラクラス

表 5-15 コントローラクラス ユーザ変数一覧

変数名	データ型	説明	属性	
			get	put
I	VT_I4	I 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
F	VT_R4	F 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
D	VT_R8	D 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
V	VT_ARRAY | VT_R4	V 型変数. 変数名の後ろに変数番号(0～)を指定します. データの要素数は 3	○	○
P	VT_ARRAY | VT_R4	P 型変数. 変数名の後ろに変数番号(0～)を指定します. データの要素数は 7	○	○
J	VT_ARRAY | VT_R4	J 型変数. 変数名の後ろに変数番号(0～)を指定します. データの要素数は 8	○	○
T	VT_ARRAY | VT_R4	T 型変数. 変数名の後ろに変数番号(0～)を指定します. データの要素数は 10	○	○
S	VT_BSTR	S 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
IARRAY	VT_ARRAY |VT_I4	I 型変数を I4 型の配列として扱います. IARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
FARRAY	VT_ARRAY |VT_R4	F 型変数を R4 型の配列として扱います. FARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○

DARRAY	VT_ARRAY|VT_R8	D 型変数を R8 型の配列として扱います. DARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
VARRAY	(VT_ARRAY|VT_R4)*配列サイズ	V 型変数を配列として扱います. V 型変数は R4 型の配列で指定します. VARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
PARRAY	(VT_ARRAY|VT_R4)*配列サイズ	P 型変数を配列として扱います. P 型変数は R4 型の配列で指定します. PARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
JARRAY	(VT_ARRAY|VT_R4)*配列サイズ	J 型変数を配列として扱います. J 型変数は R4 型の配列で指定します. JARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
TARRAY	(VT_ARRAY|VT_R4)*配列サイズ	T 型変数を配列として扱います. T 型変数は R4 型の配列で指定します. TARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
SARRAY	VT_ARRAY|VT_BSTR	S 型変数を BSTR 型の配列として扱います. SARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
IO	VT_BOOL	IO 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
IOB	VT_I1	IO 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
IOW	VT_I2	IO 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
IOD	VT_I4	IO 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○

IOF	VT_R4	IO 型変数. 変数名の後ろに変数番号(0～)を指定します.	○	○
IOARRAY	VT_ARRAY |VT_BOOL	IO を BOOL 型の配列として扱います. IOARRAY*(*は数値)で, 変数名を表します, 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
IOBARRAY	VT_ARRAY |VT_I1	IOをI1型の配列として扱います. IOBARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
IOWARRAY	VT_ARRAY |VT_I2	IO を I2 型の配列として扱います. IOWARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
IODARRAY	VT_ARRAY |VT_I4	IOをI4型の配列として扱います. IODARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○
IOFARRAY	VT_ARRAY |VT_R4	IO を R4 型の配列として扱います. IOFARRAY*(*は数値)で, 変数名を表します. 開始番号と配列サイズを指定して変数を定義します. その後配列変数としてアクセスします. 下記使用例参照.	○	○

使用例

```
'IARRAY
'変数定義 I 型変数の 0 番～2 番の 3 要素の配列. として扱う変数.
Dim IArray0 as CaoVariable
Set IArray0 = caoCtrl.AddVariable("IArray0", "StartNo = 0, ArraySize = 3")
IArray0 = Array(1,2,3)    'I0 に 1, I1 に 2, I2 に 3 が入る.

'PARRAY
'変数定義 P 型変数の 1 番～2 番の 2 要素の配列. として扱う変数.
Dim PArray0 as CaoVariable
Set PArray0 = caoCtrl.AddVariable("PArray0", "StartNo = 1, ArraySize = 2")
PArray0 = Array(Array(1,2,3,4,5,6,-1), Array(1,2,3,4,5,6,-1))
```

```
'IOARRAY
'変数定義  VT_BOOL 型, IO 番号 128 から開始, 3 要素の配列.
Dim IOArray0 As CaoVariable
Set IOArray0 = g_caoCtrl.AddVariable("IOArray0", "StartIoNo = 128, ArraySize = 3")
'値設定
IOArray0 = Array(True, False, True)
'値取得
vntVal = IOArray0

'IOFARRAY
'変数定義  VT_R4 型, IO 番号 160 から開始, 4 要素の配列.
Dim IOFArray0 As CaoVariable
Set IOFArray0 = g_caoCtrl.AddVariable("IOFArray0", "StartIoNo = 160, ArraySize = 4")
'値設定
IOFArray0 = Array(123.45, 234.56, 0.88, 345.67)
'値取得
vntVal = IOFArray0
```

表 5-16 コントローラクラス システム変数一覧

変数名	データ型	説明	属性	
			get	put
@VAR_I_LEN	VT_I4	グローバル I 型変数のサイズ	○	○
@VAR_F_LEN	VT_I4	グローバル F 型変数のサイズ	○	○
@VAR_D_LEN	VT_I4	グローバル D 型変数のサイズ	○	○
@VAR_V_LEN	VT_I4	グローバル V 型変数のサイズ	○	○
@VAR_J_LEN	VT_I4	グローバル J 型変数のサイズ	○	○
@VAR_P_LEN	VT_I4	グローバル P 型変数のサイズ	○	○
@VAR_T_LEN	VT_I4	グローバル T 型変数のサイズ	○	○
@VAR_S_LEN	VT_I4	グローバル S 型変数のサイズ	○	○

@VAR_IO_LEN	VT_I4	I/O 点数（Bit 数）	○	-
@MODE	VT_I4	1: 手動, 2: ティーチチェック, 3:自動	○	△[5]
@LOCK	VT_BOOL	true: マシンロック ON, false: マシンロック OFF	○	○
@TIME	VT_I4	マシン起動時からの経過実時間(msec)	○	-
@CURRENT_TIME	VT_DATE	現在時刻	○	-
@BUSY_STATUS	VT_BOOL	true=プログラム動作中, false=プログラム停止中	○	-
@TSR_BUSY_STATUS	VT_BOOL	true=特権タスク動作中, false=特権タスク停止中	○	
@NORMAL_STATUS	VT_BOOL	true=正常, false=異常（エラー発生中）	○	-
@ERROR_CODE	VT_I4	発生中のエラーの番号を 10 進数の値で取得します. エラーが発生していないときは, 0 を返します. 0 を書き込んだ場合はエラークリアを行います.	○	○
@ERROR_CODE_HEX	VT_BSTR	発生中のエラーの番号を16進数文字列の値で取得します. エラーが発生していないときは, "00000000"を返します.	○	-
@ERROR_DESCRIPTION	VT_BSTR	発生中のエラーの内容	○	-
@EMERGENCY_STOP	VT_BOOL	true=非常停止中 false=非常停止中ではありません	○	-
@DEADMAN_SW	VT_BOOL	デッドマンの状態	○	-
@AUTO_ENABLE	VT_BOOL	自動イネーブルの状態	○	-
@MAKER_NAME	VT_BSTR	“DENSO CORPORATION”	○	-
@TYPE	VT_BSTR	“RC8 Controller”	○	-
@VERSION	VT_BSTR	コントローラのバージョン	○	-
@SERIAL_NO	VT_BSTR	コントローラのシリアル番号	○	-
@PROTECTIVE_STOP	VT_BOOL	防護停止	○	-

[5]本機能は VRC（バーチャルロボットコントローラ）にのみ使用することができます.

@IO_ALLOC_MODE	VT_I4	I/O 割付モード 0: Mini I/O 専用モード 1: 標準モード 2: RC3 互換モード 3: 全汎用モード	○	-

5.3.2. ロボットクラス

表 5-17 ロボットクラス システム変数一覧

変数名	データ型	説明	属性	
			get	put
@CURRENT_POSITION	VT_ARRAY | VT_R8	ロボットの現在位置. 単位は任意 P 型変数.	○	-
@CURRENT_ANGLE	VT_ARRAY | VT_R8	ロボットの現在位置(各軸値). 単位は任意. J 型変数	○	-
@SERVO_ON	VT_BOOL	true=サーボ ON, false=サーボ OFF	○	○
@BUSY_STATUS	VT_BOOL	true=アーム動作中, false=アーム停止中	○	-
@TYPE_NAME	VT_BSTR	ロボットの形式	○	-
@TYPE	VT_I4	ロボットタイプデータ	○	-
@CURRENT_TRANS	VT_ARRAY | VT_R8	ロボットの現在位置. T 型	○	-
@CURRENT_TOOL	VT_I4	現在使用中のツール番号	○	○
@CURRENT_WORK	VT_I4	現在使用中のワーク番号	○	○
@SPEED	VT_R4	内部移動速度	○	○
@ACCEL	VT_R4	内部移動加速度	○	○
@DECEL	VT_R4	内部移動減速度	○	○
@JSPEED	VT_R4	内部軸速度	○	○
@JACCEL	VT_R4	内部軸加速度	○	○
@JDECEL	VT_R4	内部軸減速度	○	○

@EXTSPEED	VT_R4	外部移動速度	○	○
@EXTACCEL	VT_R4	外部移動加速度	○	○
@EXTDECEL	VT_R4	外部移動減速度	○	○
@HIGH_CURRENT_POSITION	VT_ARRAY | VT_R8	ロボットの現在位置. P 型変数. 動作仕様: マシンロック時以外はエンコーダ値を返します.	○	-
@HIGH_CURRENT_ANGLE	VT_ARRAY | VT_R8	ロボットの現在位置(各軸値). J 型変数. 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_R8	ロボットの現在位置. T 型. 動作仕様に関しては@HIGH_CURRENT_POSITION を参照してください.	○	-
@DEST_ANGLE	VT_ARRAY | VT_R8	直前の動作命令目標位置. J 型変数. ロボット停止時は, 現在位置(指令値)を返します.	○	-
@DEST_POSITION	VT_ARRAY | VT_R8	直前の動作命令目標位置. P 型変数. ロボット停止時は, 現在位置(指令値)を返します.	○	-
@DEST_TRANS	VT_ARRAY | VT_R8	直前の動作命令目標位置. T 型変数. ロボット停止時は, 現在位置(指令値)を返します.	○	-
@STATE	VT_I4	ロボットの動作状態 0: 動作完了 1: 停止(コンティニュー可) 2: 動作中(手動) 3: 動作中(自動)	○	-
@ASP_MODE	VT_I4	最適速度制御設定 0: 無効 1: PTP 2: CP 3: PTP・CP	○	○
@ERROR_CODE	VT_I4	発生中のエラーの番号を 10 進数の値で取得します. エラーが発生していないときは, 0 を返します. 0 を書き込んだ場合はエラークリアを行います.	○	○

@ERROR_DESCRIPTION	VT_BSTR	発生中のエラーの内容	○	-
@LOCK	VT_BOOL	true=マシンロック ON, false=マシンロック OFF	○	○
Tool*	VT_ARRAY | VT_R8	*で指定した番号のツール定義 X,Y,Z,RX,RY,RZ	○	○
Work*	VT_ARRAY | VT_R8	*で指定した番号のワーク定義 X,Y,Z,RX,RY,RZ,Attribute	○	○
Area*	VT_ARRAY | VT_R8	*で指定した番号のエリア定義 X,Y,Z,RX,RY,RZ,DX,DY,DZ,IO,Position,Error,Time,DRX,DRY,DRZ,Margin,Position1,Margin1,Position2,Margin2,Position3,Margin3,Position4,Margin4,Position5,Margin5,Position6,Margin6,Position7,Margin7,Position8,Margin8,Enable	○	○
Base*	VT_ARRAY | VT_R8	*で指定した番号のベース定義(1 を指定してください) X,Y,Z,RX,RY,RZ,Attribute Attribute は 0 を指定してください	○	○

5.3.3. タスククラス

表 5-18 タスククラス システム変数一覧

変数名	データ型	説明	属性	
			get	Put
@STATUS	VT_I4	タスクの状態. 0: タスクが生成されていません(NON_EXISTENT) 1: 一時停止中 2: 停止中 3: 実行中 4: ステップ停止中	○	-
@PRIORITY	VT_I4	タスクの優先度. 未対応. SetThreadPriority(), GetThreadPriority()を参照	-	-
@LINE_NO	VT_I4 | VT_ARRAY	タスクが実行中の行番号とファイル ID. [0] = 行番号 [1] = 実行中のファイル ID（CaoFile::get_ID()に対応）	○	-
@CYCLE_TIME	VT_I4	タスクの 1 サイクルの実行時間. ㎳単位.	○	-

@START	VT_I4	タスクの開始. 値の意味は CaoTask::Start メソッドの Mode 引数と同じ. モードは1:1 サイクル実行, 2:連続実行, 3:1ステップ送り Start メソッドのようにオプションを指定することはできません.	-	○
@STOP	VT_I4	タスクの停止. 値の意味は CaoTask::Stop メソッドの Mode 引数と同じ. モードは 0:デフォルト停止, 1:瞬時停止, 2:ステップ停止, 3:サイクル停止, 4:初期化停止 Stop メソッドのようにオプションを指定することはできません. デフォルト停止(0)は瞬時停止(1)に対応します.	-	○
@ELAPSED_TIME	VT_I4	タスク実行開始からの経過時間. ms単位.	○	-
@STATUS_DETAILS	VT_I4	タスク状態の詳細情報. TASK_NON_EXISTENT = 0, タスクが存在しない TASK_SUSPEND = 1, 一時停止中 TASK_READY = 2, レディ TASK_RUN = 3, 実行中 TASK_STEPSTOP = 4, ステップ停止 TASK_CNTSTP = 5, コンティ停 TASK_PEND = 6, Pending TASK_DELAY = 7, Delay	○	-

5.3.4. ファイルクラス

表 5-19 ファイルクラス システム変数一覧

変数名	データ型	説明	属性	
			get	Put
@CRC	VT_I4	CRC32	○	-

5.4. イベント一覧

コントローラのエラー通知や状態の変化を OnMessage イベントとし受け取ることが可能です.

表 5-20 OnMessage イベント一覧

イベント	イベント番号 Number	Value		備考
		型	値	
エラー発生	3	VT_I4	エラーコード	レベル 0 のエラーも発生
非常停止	5	VT_I4	ON:-1 OFF:0	-
防護停止	6	VT_I4	ON:-1 OFF:0	-
自動イネーブル	7	VT_I4	ON:-1 OFF:0	-
モード切替	9	VT_I4	MANUAL:1 TEACH:2 AUTO:3	-
ロボット動作終了	13	-	-	ロボット動作コマンド(Move 等)の動作終了時に発生します. 以下のロボット動作の停止もしくは完了時にもイベントは発生します. ・PacScript プログラムが「停止中」になった時 ・手動操作での変数移動完了した時 (※「一時停止中」の場合は発生しません. 「一時停止中」から動作再開後, 動作完了した場合に発生します.)

[注意事項]

システムでは他にもイベントを発生させていますが, 上記以外のイベントに関してはサポート外です.

使用例

```
Dim g_eng As CaoEngine
Dim WithEvents g_ctrl As CaoController
Dim lEventNo As Long
Dim vntVal As Variant

Private Sub Form_Load()
    Set g_eng = New CaoEngine
    ' 接続処理 IP は各コントローラの設定にしてください
    Set g_ctrl = g_eng.Workspaces(0).AddController("RC8", "CaoProv.DENSO.RC8", "",
"Server=192.168.0.1")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' コントローラオブジェクトの破棄
    g_eng.Workspaces(0).Controllers.Remove g_ctrl.Index
```

```
    Set g_ctrl = Nothing
    ' CaoEngine の破棄
    Set g_eng = Nothing
End Sub

Private Sub g_ctrl_OnMessage(ByVal pICaoMess As CAOLib.ICaoMessage)
    lEventNo = pICaoMess.Number
    vntVal = pICaoMess.Value
End Sub
```

付録A. CaoController オブジェクトの生成

ORiN の CaoController を作成するには以下の手順を取ります.

(1) オブジェクトを保持するための変数を用意します.
(2) CaoEngine オブジェクトを生成します.
(3) CaoWorkspace オブジェクトを取得もしくは生成します.
(4) CaoController オブジェクトを生成します.

以下に詳しい内容を説明します. ここでは Visual Basic 6.0 言語を使用した場合の事例で, オブジェクトの生成には New キーワードを用います.

(5) 最初にオブジェクトを保持するために変数を宣言します. ここでコントローラオープンに必要なオブジェクトは CaoEngine オブジェクトと CaoWorkspace オブジェクトです. また, AddController メソッドによって CaoController オブジェクトが生成されます. よってこの三つに対応する変数を用意します. 以下に, プライベート変数として各オブジェクト型の変数を宣言する例を示します.

```
Private caoEng As CaoEngine      ' Engine オブジェクト
Private caoWs As CaoWorkspace    ' CaoWorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト
```

(6) 次に CaoEngine オブジェクトを生成します. これは New キーワードで生成して, Set ステートメントで変数に代入します.

```
Set caoEng = New CaoEngine
```

(7) CaoEngine オブジェクトは生成時にデフォルトの CaoWorkspaces, CaoWorkspace オブジェクトを一つずつ作成しています. デフォルトの CaoWorkspace オブジェクトを取得するときは CaoWorkspaces.Item(0)を使用します. 以下にデフォルトの CaoWorkspace オブジェクトを取得する例を示します.

```
Set caoWs = caoEng.CaoWorkspaces.Item(0)
```

(8) CaoController オブジェクトは CaoWorkspace オブジェクトの AddController メソッドで生成することができます.

```
' CaoCtrl と CaoWS はオブジェクトを保持するための変数です
Set CaoCtrl = caoWs.AddController( "RC8","CaoProv.DENSO.RC8"," ","Server=192.168.0.1")
```

使用例

```
Private caoEng As CaoEngine      ' Engine オブジェクト
Private caoWs As CaoWorkspace    ' WorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト

Set caoEng = New CaoEngine
Set caoWS = caoEng.CaoWorkspaces.Item(0)
Set caoCtrl = CaoWs.AddController( “RC8","CaoProv.DENSO.RC8"," “,"Server=192.168.0.1")
```

付録B. POSEDATA 型定義

RC8 プロバイダではデンソーロボットのポーズデータ型およびベクトル型を VARIANT 型変数で扱えるよう"POSEDATA 型"と称して次に示す型定義を行っています.

POSEDATA 型(VARIANT)

- VT_BSTR[6] "[<パス>] [<変数型>]<インデックス> [<付加軸>]"
  または
  "[<パス>] [<変数型>](<要素 1>,<要素 2>,...) [<付加軸>]"
- VT_R4|VT_ARRAY[7] <即値> = (<要素 1:VT_R4>,<要素 2:VT_R4>,...)[8]
- VT_VARIANT|VT_ARRAY (<値>[,<変数型>[,<パス>[, <付加軸>]]])
  - <値> <インデックス:VT_R4>
    または
    <即値:VT_R4|VT_ARRAY>
  - <変数型> P, T, J, V 型の VT_I4 または VT_BSTR 指定 (省略時＝P 扱い)
  - <パス> @P, @E, @0, @<数値>の VT_I4 または VT_BSTR 指定
    (省略時＝@0)
  - <付加軸> <付加軸オプション:VT_VARIANT|VT_ARRAY>
    (省略時＝付加軸指定なし)

<パス> : @P, @E, @0, @<数値>

表記	@P	@E	@0	@<数値:n>	なし
VT_BSTR	"@P"	"@E"	"@0"	"@n"	""
VT_I4	-1	-2	0	n	0

<変数型> : P 型, T 型, J 型, V 型

表記	P	T	J	V	なし
VT_BSTR	"P"	"T"	"J"	"V"	""
VT_I4	0	1	2	3	-1

<インデックス> : <数値:VT_R4>

<要素 n> : <数値:VT_R4>

<付加軸オプション> : (<EX または EXA>, (<軸 1:VT_I4>,<値 1:VT_R8>)[,(<軸 2>,<値 2>)...])

表記	EX	EXA	なし

[6] VT_BSTR のみ複数 POSEDATA 型の", "カンマ区切りでの同時指定も可能です.
[7] <変数型>,<パス>は指定できないため, それぞれデフォルトで P 型, @0 と同じ扱いになります.
[8] <変数型>,<パス>は指定できないため, それぞれデフォルトで P 型, @0 と同じ扱いになります.

VT_BSTR	"EX"	"EXA"	""
VT_I4	1	2	0

POSEDATA 型を使用して次の PacScript 言語の書式を表現することが出来ます.

[<パス開始変位>] <ポーズ:P,T,J 型> [<付加軸>] (C0 書式)

[<パス開始変位>] <移動量:V 型> (C1 書式)

[<パス開始変位>] <値> [<付加軸>] (C2 書式)

[<パス開始変位>] (<要素 1>,<要素 2>,...) [<付加軸>] (C3 書式)

付録B.1. 表記例

[<パス開始変位>] <ポーズ> [<付加軸>] (C0)

ex1. T200

文字列指定	"T200"
VARIANT 型配列指定 (変数型は文字列指定)	Array(200,"T")[9]
VARIANT 型配列指定 (変数型は数値指定)	Array(200,1)

ex2. @P J100

文字列指定	"@P J100"
VARIANT 型配列指定 (変数型, パスは文字列指定)	Array(100,"J","@P")
VARIANT 型配列指定 (変数型, パスは数値指定)	Array(100,2,-1)

ex3. @E P(10.0, 10.5, 34.6, 0.0, 90.0, 0.0, -1.0)

文字列指定	"@E P(10.0, 10.5, 34.6, 0.0, 90.0, 0.0, -1.0)"
VARIANT 型配列指定 (即値指定. 変数型, パスは文字列指定)	Dim p(6) as Single Dim vP as Variant p(0) = 10.0 : p(1) = 10.5 : p(2) = 34.6 : p(3) = 0.0 p(4) = 90.0 : p(5) = 0.0 : p(6) = -1.0 vP = p() Array(vP,"P","@E")
VARIANT 型配列指定 (即値指定. 変数型, パスは数値指定)	Dim p(6) as Single Dim vP as Variant p(0) = 10.0 : p(1) = 10.5 : p(2) = 34.6 : p(3) = 0.0 p(4) = 90.0 : p(5) = 0.0 : p(6) = -1.0 vP = p() Array(vP, 0, -2)

ex4. @P J100 EXA((7, 30.5), (8, 90.5))

[9] Array(…)は渡された要素を配列に代入してその配列を返す関数を表しています.(VB6 の Array 関数)

文字列指定	"@P J100 EXA((7, 30.5), (8, 90.5))"
VARIANT 型配列指定 (変数型, パス, 付加軸は文字列指定)	Array(100,"J","@P", Array("EXA",Array(7,30.5), Array(8,90.5)))
VARIANT 型配列指定 (変数型, パス, 付加軸は数値指定)	Array(100,2,-1, Array(2, Array(7,30.5), Array(8,90.5)))

[<パス開始変位>] <移動量> (C1)

ex1. @P V20

文字列指定	"@P V20"
VARIANT 型配列指定 (変数型, パスは文字列指定)	Array(20,"V","@P")
VARIANT 型配列指定 (変数型, パスは数値指定)	Array(20,3,-1)

ex2. @E V(0.0, 125.5, 50.0)

文字列指定	"@E V(0.0, 125.5, 50.0)"
VARIANT 型配列指定 (即値指定. 変数型, パスは文字列指定)	Dim v(2) as Single Dim vV as Variant v(0) = 0.0 : v(1) = 125.5 : v(2) = 50.0 vV = v() Array(vV,"V","@E")
VARIANT 型配列指定 (即値指定. 変数型, パスは数値指定)	Dim v(2) as Single Dim vV as Variant v(0) = 0.0 : v(1) = 125.5 : v(2) = 50.0 vV = v() ' = VT_R4 | VT_ARRAY Array(vV, 3, -2)

[<パス開始変位>] <値> [<付加軸>] (C2)

ex1. @P 1

文字列指定	"@P 1"
VARIANT 型配列指定 (変数型, パスは文字列指定)	Array(1," ","@P")
VARIANT 型配列指定 (変数型, パスは数値指定)	Array(1,-1,-1)

ex2. @P 1.56

文字列指定	"@P 1.56"
VARIANT 型配列指定	Array(1.56," ","@P")

（変数型, パスは文字列指定）	
VARIANT 型配列指定	Array(1.56,-1,-1)
（変数型, パスは数値指定）	

[<パス開始変位>] (<要素 1>,<要素 2>,..) [<付加軸>] (C3)

ex1. @P (1, 30.0)

文字列指定	"@P (1, 30.0)"
VARIANT 型配列指定 （変数型, パスは文字列指定）	Dim v(1) as Single v(0) = 1 : v(1) = 30.0 Dim vV as Variant vV = v() Array(vV," “,"@P")
VARIANT 型配列指定 （変数型, パスは数値指定）	Dim v(1) as Single v(0) = 1 : v(1) = 30.0 Dim vV as Variant vV = v() Array(vV, -1, -1)

その他の表記方法

ex1. V1,V2,V3

(CaoRobot::Rotate()の回転面)

文字列指定	"V1,V2,V3"
文字列型配列指定	Array("V1","V2","V3")
VARIANT 型配列指定 （変数型は文字列指定）	Array(Array(1,"V"),Array(2,"V"),Array(3,"V"))
VARIANT 型配列指定 （変数型は数値指定）	Array(Array(1,3),Array(2,3),Array(3,3))

ex2. APPROACH P,P70, 60, NEXT

(CaoRobot::Execute()の Approach コマンド アプローチ長パス指定なし)

第2引数:文字列指定 第3引数:文字列指定	.Execute "APPROACH", Array(1, "P70", "60", "NEXT")
第2引数:VARIANT 配列指定 第3引数:VARIANT 配列指定	.Execute "APPROACH", Array(1, Array(70, "P"), _ Array(60, "", ""), "NEXT")

ex3. APPROACH L,J(60.5,30.3,400,90),@100 70, NEXT

(CaoRobot::Execute()の Approach コマンド アプローチ長パス指定なし)

第2引数:文字列指定 第3引数:文字列指定	.Execute "APPROACH", Array(2, "J(60.5,30.3,400,90)", "@100 70", "NEXT")
第2引数:VARIANT 配列指定	Dim j(3) as Single Dim vJ as Variant j(0) = 60.5 : j(1) = 30.3 : j(2) = 400 : j(3) = 90

(即値指定. 変数型は文字列指定) 第3引数:VARIANT 配列指定 (変数型, パスは文字列指定)	vJ = j() ‘ = VT_R4 | VT_ARRAY .Execute "APPROACH", Array(2, Array(vJ, "J"), _ Array(70,"","@100"), "NEXT")
第2引数:VARIANT 配列指定 (即値指定. 変数型は文字列指定) 第3引数:VARIANT 配列指定 (変数型, パスは数値指定)	Dim j(3) as Single Dim vJ as Variant j(0) = 60.5 : j(1) = 30.3 : j(2) = 400 : j(3) = 90 vJ = j() ‘ = VT_R4 | VT_ARRAY .Execute "APPROACH", Array(2, Array(vJ, "J"), _ Array(70, -1, 100), "NEXT")

[注意事項]

即値を POSEDATA 型 VT_R4|VT_ARRAY 形式で直接指定した場合はデフォルトで P 型, @0 扱いになるため P 型以外のデータを VT_R4|VT_ARRAY 形式で直接扱うことは出来ません. このような場合は VT_VARIANT|VT_ARRAY 形式または VT_BSTR 形式を使用して明示的に扱うデータの変数型を指定するようにしてください.

次のようなコードは期待する動作にはならないので注意してください.

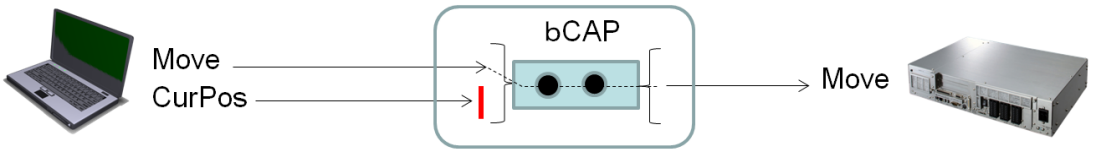
```
‘[PAC] MOVE P, J100
        Dim vJ as Variant
        vJ=CaoCtrl.Variables( “J100").Value    ‘VT_R4|VT_ARRAY
        Robot.Move 1, vJ                       ‘間違い!! = MOVE P, P(<j1>,<j2>,<j3>,…)
```

正しくは次のようなコードになります.

```
        Robot.Move 1, Array(vJ,"J")            ‘変数型指定= MOVE P, J(<j1>,<j2>,<j3>,…)
```

付録C. RC8 コントローラに対するコマンドの同時発行

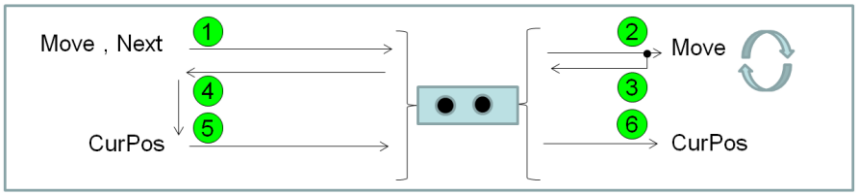
RC8 プロバイダによる実機接続では，接続単位（＝AddController 単位）で bCAP プロトコルによる通信がクライアント/サーバーの関係で行われます．このとき，クライアント(PC)からのコマンドがサーバー（RC8 コントローラ）によりシリアライズされるため,ロボット動作中に現在位置をモニタするような，二つ以上のコマンドを同時に発行する処理を実現する場合はクライアント側で以下の注意を払う必要があります．



接続単位でコマンドがシリアライズ

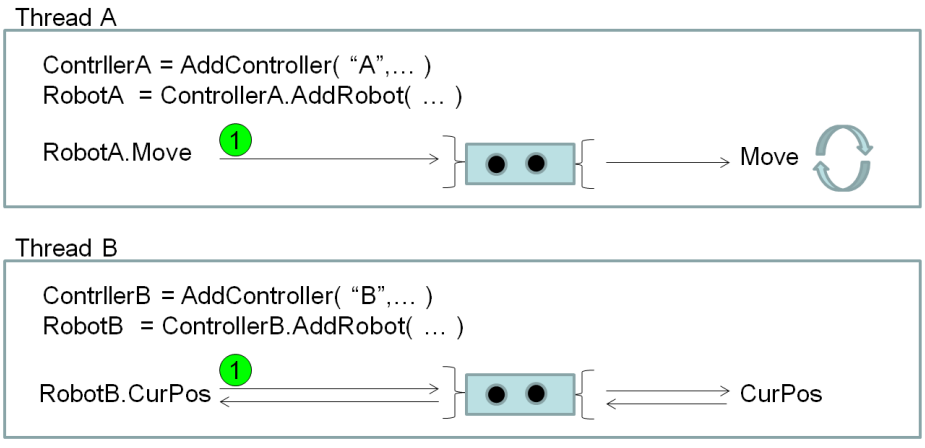
(1) シングルスレッドの場合

Next オプションで動作コマンドを非同期化し，動作完了待ちのループの中で現在位置モニタを挿入する．



(2) マルチスレッドの場合

動作実行と現在位置モニタ用に別々のスレッドを作成，スレッド単位で CaoWorkspace::AddController を実行し，別々の接続を確立させる．



上記例では ThreadA で Next オプションなしで動作コマンドを同期実行し，動作完了まで ThreadA がブロックされるが，現在位置モニタは別スレッド（ThreadB），別接続で実現されているため並列実行され，問題が生じません．ただし， ThreadA は GUI を制御するメインスレッドであることは好ましくありません．なぜなら動作コマンド実行中の間，メイン（GUI 制御）スレッドが占有され，結果として画面描画が阻害されるからです．こ

のように ThreadA はメインスレッドとは別に新規に作成する方が好ましいと考えられます.

付録D. 無停止教示点補正機能（外観検査軌道生成）

本プロバイダを使用可能にするには, RC8 の拡張機能画面にて別途「Non-stop motion calculator」ライセンスを入力する必要があります. 入力方法は, DENSO ROBOT USERS MANUALS の「機能拡張画面の表示, 追加/削除」を参照ください. 使用方法については,「ORiN2 RC8 プロバイダ"無停止教示点補正機能" オプションユーザーズ ガイド」を参照してください.

付録D.1. パラメータ

GenerateNonStopPath コマンドの<座標情報>パラメータと, <エリア情報>パラメータについて, 詳細を以下に示します.

<座標情報:VT_VARIANT | VT_ARRAY> =
<X:VT_R8>,
<Y:VT_R8>,
< Z:VT_R8>,
< RX:VT_R8>,
< RY:VT_R8>,
< RZ:VT_R8>,
< Fig:VT_R8>,
< J7:VT_R8>,
< J8:VT_R8>,
<動作速度:VT_R8>=
各通過点を通過する際の速度比率.
Move 時の動作オプション（SPEED）を 100 で割った値となります. (0.0～1.0),
<動作パターン:VT_I4> = (0:@P, 1:@0, 2:@E),
< Tool 番号 :VT_I4>

<エリア情報:VT_R4 | VT_ARRAY > =
<エリアサイズ X:VT_R8>,
<エリアサイズ Y:VT_R8>,
< エリアサイズ Z:VT_R8>,
< エリア Angle:VT_R8>,
< エリアサイズ J7:VT_R8>,
< エリアサイズ J8:VT_R8>

付録D.2. エラーコード

プロバイダ内で定義している GenerateNonStopPath コマンドで失敗した場合のエラーコードは 0x8150015E となります. また, コマンド独自のカスタムエラーコードの一覧と復帰処置を以下に示します. カスタムエラーコードは, 接続している RC に表示されるエラーの「詳細を表示」 → [オリジナルナンバ]を確認してください.

番号 (オリジナルナンバ)	内容	復帰処置
0x2000****	P2J 変換エラー (元データ)	エラーが発生した教示点はロボットが移動できない位置です. 教示点の座標を変更してください.
0x2100****	ソフトリミット範囲エラー (元データ)	エラーが発生した教示点はロボットが移動できない位置です. 教示点の座標を変更してください.
0x2200****	個別速度比の範囲エラー	エラーが発生した教示点の<座標情報(教示点)>引数の要素<動作速度>(付録 D.1 パラメータ 参照)の値を変更してください. (有効範囲:0.0～1.0)
0x2300****	P2J 変換エラー (補正中データ)	以下の1つ, または複数の復帰処置を実施してください. ・エラーが発生した教示点の座標を変更してください. ・エラーが発生した教示点の前にダミー点を設定してください. ・エラーが発生した教示点の後ろにダミー点を設定してください. ・付録D.5のパラメータを変更してください.
0x2400****	ソフトリミット範囲エラー (補正中データ)	以下の1つ, または複数の復帰処置を実施してください. ・エラーが発生した教示点の座標を変更してください. ・エラーが発生した教示点の前にダミー点を設定してください. ・エラーが発生した教示点の後ろにダミー点を設定してください.
0x2500****	補正演算収束不能	以下の1つ, または複数の復帰処置を実施

		してください. ・エラーが発生した教示点の座標を変更してください. ・エラーが発生した教示点の前にダミー点を設定してください. ・エラーが発生した教示点の後ろにダミー点を設定してください.
0x2600****	隣接する教示点間の距離(+姿勢)が近すぎる	以下の1つ, または複数の復帰処置を実施してください. ・エラーが発生した教示点の座標を変更してください. ・エラーが発生した教示点の前にダミー点を設定してください. ・エラーが発生した教示点の後ろにダミー点を設定してください.
0x2A00****	無限回転の付加軸には対応していません	有限回転の付加軸に変更してください.
0x2F000000	総速度比が範囲エラー	<総速度比>の引数を修正してください. (有効範囲:0.0～1.0)
0x2F010000	補正係数の範囲エラー	<補正係数>の引数を修正してください. (有効範囲:0.0～1.0)
0x2F030000	メモリ不足	コントローラを再起動後, 再度実行してください. 復帰しない場合は, 弊社サービス部門までご連絡ください.
0x2F0A0000	データサイズオーバー	<教示点数>の引数を修正してください. (有効範囲:0～200)

0x2000****～0x2800****のオリジナルナンバ
****に, エラーが発生した教示点番号－1の値が入ります.

0x2A00****のオリジナルナンバ
****に, 無限回転が設定された軸番号が入ります.

付録D.3. 制限事項

GenerateNonStopPath コマンドの実行には以下の制限があります.

・ 教示点数の上限 = 200 点

・ 6 軸ロボットのみ有効
・ エリアサイズの付加軸の値は，設置する付加軸によって，回転(degree)，直動(mm)の値をそれぞれ指定します
・ 付加軸の無限回転は対応不可

付録D.4. サンプルプログラム

以下に CaoScript で作成したサンプルプログラムを示します．教示座標は，VS-6577G-BA のロボットタイプを用いています．IP は各コントローラで設定した値を用いてください．サンプルプログラムでは以下の設定値を用いて接続しています．

IP: 10.6.235.72

Sample NonStopPath.vbs

```
' Generate NonStopPath
Const RC_ADDRESS = "10.6.235.72"

Sub Main
  Dim rc
  Dim vntTeachPos()
  Dim vntAreaInfo()
  Dim vntMovePos
  Dim vntParam
  Dim lIndex

  dbg.ClearLog

  set rc = cao.AddController("RC", "CaoProv.DENSO.RC8", "", "server=" & RC_ADDRESS)
  set rob = rc.AddRobot("Robot")

  ' GenerateNonStopPath Command Parameter (Pos, Area, Size, SpdRate, Coef)
  ' Pos : TeachPoint Data (x, y, z, rx, ry, rz, fig, J7, J8, SpdRate, attr, ToolNum)
  redim vntTeachPos(7)

  vntTeachPos(0) = Array(300.0, 100.0, 600.0, 180.0, 0.0, 180.0, 5, 0.0, 0.0, 100 * 0.01, 1, 0)
  vntTeachPos(1) = Array(300.0, 91.0, 600.0, 180.0, 0.0, -180.0, 5, 0.0, 0.0, 100 * 0.01, 0, 0)
  vntTeachPos(2) = Array(310.0, 30.0, 600.0, 180.0, 0.0, -180.0, 5, 0.0, 0.0, 100 * 0.01, 1, 0)
  vntTeachPos(3) = Array(315.5, 24.5, 600.0, 180.0, 0.0, -180.0, 5, 0.0, 0.0, 100 * 0.01, 0, 0)
  vntTeachPos(4) = Array(300.0, 10.0, 600.0, 180.0, 0.0, 173.0, 5, 0.0, 0.0, 100 * 0.01, 1, 0)
  vntTeachPos(5) = Array(300.0, 10.0, 600.0, 180.0, 0.0, 176.0, 5, 0.0, 0.0, 100 * 0.01, 0, 0)
  vntTeachPos(6) = Array(300.0, 10.0, 600.0, 180.0, 0.0, 171.0, 5, 0.0, 0.0, 100 * 0.01, 0, 0)
  vntTeachPos(7) = Array(300.0, 10.0, 600.0, 180.0, 0.0, -180.0, 5, 0.0, 0.0, 100 * 0.01, 1, 0)

  ' Area : Area Info (x, y, z, angle, J7, J8)
  redim vntAreaInfo(7)

  vntAreaInfo(0) = Array(4,4,4,4,0,0)
  vntAreaInfo(1) = Array(4,4,4,4,0,0)
  vntAreaInfo(2) = Array(4,4,4,4,0,0)
  vntAreaInfo(3) = Array(4,4,4,4,0,0)
  vntAreaInfo(4) = Array(4,4,4,4,0,0)
  vntAreaInfo(5) = Array(4,4,4,4,0,0)
  vntAreaInfo(6) = Array(4,4,4,4,0,0)
  vntAreaInfo(7) = Array(4,4,4,4,0,0)

  dbg.Output "Teach Points"
  for lIndex = 0 to Ubound(vntTeachPos)
```

```
            dbg.Output lIndex & ":" & dat.BstrFromVariant(vntTeachPos(lIndex))
    next

    ' ----------------------------------------
    ' Generate NonStopPath
    ' ----------------------------------------
    vntMovePos = rob.Execute("GenerateNonStopPath", Array(vntTeachPos, vntAreaInfo,
Ubound(vntTeachPos) + 1, 100.0 * 0.01, 0.7))

    dbg.Output "Move Points"
    for lIndex = 0 to Ubound(vntMovePos)
            dbg.Output lIndex & ":" & dat.BstrFromVariant(vntMovePos(lIndex))
    next
End Sub
```

付録D.5. 軌道補正失敗時(エラーコード[オリジナルナンバ] : 0x8150015E[0x2300****])の回避方法

GenerateNonStopPath コマンドに失敗する場合，補正中の教示点座標が，ロボットの各軸に対する最大補正角度の値を超過している可能性があります(エラーコード[オリジナルナンバ] : 0x8150015E[0x2300****](****は教示点番号)).

上記のエラーが発生した場合は，エラーの発生した教示点の座標を変更するか，以下のパラメータを調整してください.

RC8 の「アーム」-「使用条件」にて，以下のパラメータを調整してください.

パラメータ	概要	入力範囲
最適 FIG 探索オフセット値(J1)	1 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J2)	2 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J3)	3 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J4)	4 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J5)	5 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J6)	6 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J7)	7 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8
最適 FIG 探索オフセット値(J8)	8 軸最大補正角度のオフセット[deg]	-21474.8 ～ 21474.8