RC9 プロバイダ

デンソー ロボット RC9 対応

Version 1.0.0

ユーザーズ ガイド

May 11, 2021

【備考】

【改版履歴】

バージョン	日付	内容
1.0.0	2020-07-09	初版
	2020-11-09	付録 A. POSEDATA 型定義を追加
	2021-02-11	「3. AddController コマンドの引数」個所の誤記修正
	2021-05-11	「4. STO 状態の解除」の追加

【対応機器】

機種	バージョン	注意事項
RC9	1.0.0～	

【対応コマンド】

機種	バージョン	注意事項

目次

1. はじめに

ORiN2 SDK Ver 2.1.51 以上から RC9 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.
RC9 プロバイダの使用方法は、RC8 プロバイダと同等です。
本書は、RC9 プロバイダと RC8 プロバイダで、違いのある箇所のみ記述しています。

・起動権の設定方法が異なります。
・[AddController]コマンドの引数が異なります。
・モータ ON には STO 状態の解除が必要です。
・対応コマンドが異なります。

2. 起動権の設定方法

起動権の設定は，以下の手順で行います．

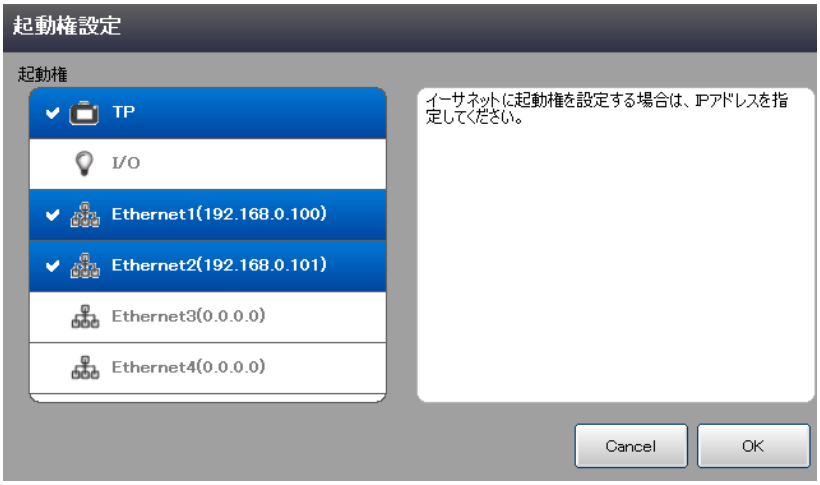
(1) ロボットコントローラを手動モードにします。

(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)]で起動権を許可するパソコンの IP アドレスを設定します。下記設定では TP と Ethernet1(192.168.0.100)と Ethernet2(192.168.0.101)が起動権を取得しています。

※RC9 の起動権は、RC8 にあった ANY の設定がなくなり、代わりに複数デバイスを選択できるようになっています。



[F6] ⇒ [F5] ⇒ [F1] ⇒



3. AddController コマンドの引数

AddController コマンドの引数が **RC8** と異なります。

コントローラへの接続

RC9

```
Set g_ctrl = g_eng.Workspaces(0).AddController("Controller", "caoProv.DENSO.RC9", "",
"Server=192.168.0.1")
```

RC8

```
Set g_ctrl = g_eng.Workspaces(0).AddController("Controller", "caoProv.DENSO.RC8", "",
"Server=192.168.0.1")
```

プロジェクトへの接続

RC9

```
Set g_ctrl = g_eng.Workspaces(0).AddController("Controller", "caoProv.DENSO.RC9", "",
"RCPJ=C:¥test¥$Current¥default.rcpj")
```

RC8

```
Set g_ctrl = g_eng.Workspaces(0).AddController("Controller", "caoProv.DENSO.RC8", "",
"WPJ=C:¥test¥test.WPJ")
```

4. STO 状態の解除

モータ ON するには STO 状態の解除が必要です. STO 状態及び解除方法については RC9 User Manuals の「STO 状態の解除(ID : 10038)」を参照してください。「CaoController::Execute("ResetStoState") コマンド」を使用して STO 状態を解除することも可能です。

5. コマンドリファレンス

5.1. コマンド一覧

表 4-1 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. 13
CaoController			
	AddFile	PacScript・システムファイル，フォルダに接続	P. 15
	AddRobot	ロボットに接続	P. 16
	AddTask	タスク(PacScript)に接続	P. 17
	AddVariable	ユーザ・システム変数に接続	P. 17
	get_Name	コントローラ名の取得	P. 18
	get_FileNames	ファイル名の一覧の取得	P. 18
	get_TaskNames	タスク(PacScript)名の一覧の取得	P. 19
	get_VariableNames	ユーザ・システム変数の一覧の取得	P. 19
	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. 19
CaoFile			
	AddFile	PacScript ファイルに接続	P. 38
	AddVariable	ファイルのシステム変数に接続	P. 38
	get_VariableNames	システム変数の一覧取得	P. 38
	get_FileNames	ファイル名一覧の取得	P. 39
	get_Size	ファイルのサイズの取得	P. 39
	get_Value	ファイルの内容の取得	P. 39
	put_Value	ファイルの内容の書き換え	P. 39
CaoRobot			
	Accelerate	内部加速度，内部減速度の設定	P. 39
	AddVariable	システム変数の接続	P. 41
	get_VariableNames	システム変数一覧の取得	P. 41
	Halt	ロボット非同期動作時の停止	P. 41
	Change	ロボットのツール座標系/ユーザ座標系の変更	P. 42
	Drive	サポートされていません.	P. 42
	Move	ロボットの動作	P. 43
	Rotate	軸回りの回転動作	P. 46
	Speed	内部移動速度の設定	P. 48
	Execute	ロボット特有の動作の実行	P. 49
CaoTask			
	AddVariable	システム変数の接続	P. 101
	get_VariableNames	システム変数の一覧取得	P. 101
	Start	PacScript プログラムの実行	P. 101

5.2. メソッド・プロパティ

5.2.1. CaoWorkspace::AddController メソッド

RC9 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.RC9".
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 RC9 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.
RCPJ={<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*.rcpj 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”RCPJ={*}”扱いになります. 例: “RCPJ={C:¥Test¥$Current¥default.rcpj}” “RCPJ={*}”
Message=<イベント通知>	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

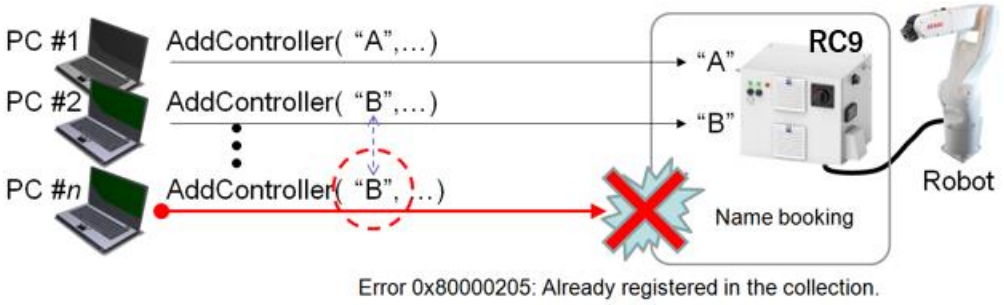
```
Private caoEng As CaoEngine      ' Engine オブジェクト
Private caoWs As CaoWorkspace    ' WorkSpace オブジェクト
Private caoCtrl As CaoController ' Controlle オブジェクト

Set caoEng = New CaoEngine
Set caoWS = caoEng.CaoWorkspaces.Item(0)
Set caoCtrl =
CaoWS.AddController( “rc9","CaoProv.DENSO.RC9"," “,"Server=192.168.0.1,Timeout=1000")
```

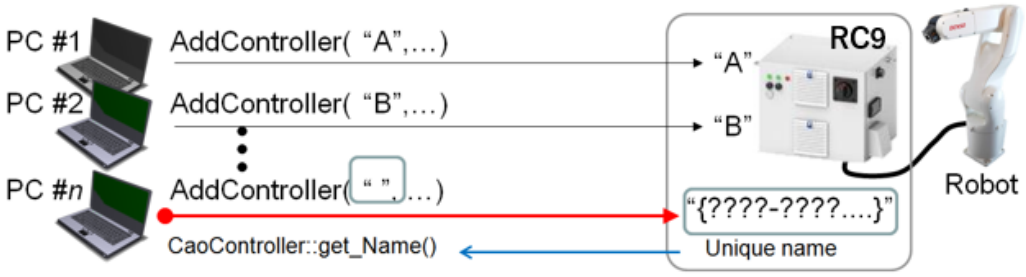
5.2.1.1. 実機に対する複数接続の注意点

RC9 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	*.PNS9	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.

書式 AddRobot(<bstrName:BSTR >)

bstrName : [in] ロボット名

使用例

```
Dim CaoRob as CaoRobot
Set Rob = caoCtrl.AddRobot("Robot0")
```

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．RC9 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 RC9 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.

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::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.RC9"," ","Server=192.168.0.1")

Debug.Print caoCtrl. Name
```

5.2.7. 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
```

5.2.8. 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
```

5.2.9. 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.

5.2.10. 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.

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.21
	GetErrorDescription	エラー文字列の取得	1.0.0	P.21
	GetCurErrorCount	現在発生しているエラー数の取得	1.0.0	P.35
	GetCurErrorInfo	現在発生しているエラー情報の取得	1.0.0	P.36
タスク制御				
	KillAll	全タスク停止	1.0.0	P.21
	SuspendAll	全タスク瞬時停止	1.0.0	P.23
	StepStopAll	全タスクステップ停止	1.0.0	P.24
	ContinueStartAll	全タスクコンティニュスタート	1.0.0	P.24
	KillAllTsr	実行中の全特権タスクを初期化停止	1.0.0	P.22
	RunAllTsr	モードで指定された特権タスクの起動	1.0.0	P.23
ログ				
	GetErrorLogCount	エラーログ数の取得	1.0.0	P.25
	GetErrorLog	エラーログの取得	1.0.0	P.25
	GetOprLogCount	操作ログ数の取得	1.0.0	P.26
	GetOprLog	操作ログの取得	1.0.0	P.27
変数アクセス				
	GetPublicValue	Public 変数読み込み	1.0.0	P.28
	SetPublicValue	Public 変数書き込み	1.0.0	P.30
その他				
	SysState	コントローラのステータスの取得	1.0.0	P.32
	SysInfo	コントローラのシステム情報を取得	1.0.0	P.32
	SetAllDummyIO	IO を疑似化	1.0.0	P.35
	StoState	STO 状態を取得	1.0.0	P.36

ResetStoState	STO 状態を解除	1.0.0	P.37
GetSlaveButtonState	マスタースレーブ時のボタン状態を取得	1.0.0	P.37
ClearSlaveButtonState	マスタースレーブ時のボタン状態クリア	1.0.0	P.37

5.2.10.1. 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"
```

5.2.10.2. 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 )
```

5.2.10.3. 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"
```

5.2.10.4. 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"
```

5.2.10.5. 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"
```

5.2.10.6. 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"
```

5.2.10.7. 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"
```

5.2.10.8. 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"
```

5.2.10.9. 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")
```

5.2.10.10. 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:16 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

[15]:呼び出し元が PC の場合の IP アドレス (VT_I4)

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
```

5.2.10.11. 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")
```

5.2.10.12. 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:13 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

[11]:操作内容 (VT_BSTR)

[12]:ログインユーザ ID (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
```

5.2.10.13. 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)
```

5.2.10.14. 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
```

5.2.10.15. 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")
```

5.2.10.16. 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	システム予約 0	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 4000:リスクアセッサ 5000 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
29	コントローラタイプ 0:RC9	整数型
30	セーフティタイプ	整数型

	0:標準 1:セーフティモーション	整数型

使用例

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

5.2.10.17. 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
```

5.2.10.18. 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")
```

5.2.10.19. 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)
```

5.2.10.20. CaoController::Execute("StoState") コマンド

STO 状態を返します.

書式　StoState( )

引数	:	なし
戻り値	:	[out] 状態<VT_BOOL>

使用例

```
Dim state as Variant
state = caoCtrl.Execute("StoState")
```

5.2.10.21. CaoController::Execute("ResetStoState") コマンド

STO 状態を解除します.

書式　ResetStoState( )

引数　：なし
戻り値　：なし

使用例

```
caoCtrl.Execute "ResetStoState"
```

5.2.10.22. CaoController::Execute("GetSlaveButtonState") コマンド

スレーブコントロール機能有効時に, マスターロボットからスレーブコントローラに通知されているボタン状態を, スレーブコントローラから取得できます.

書式　GetSlaveButtonState(<lBtnType>)

<lBtnType>　：状態を取得する対象を指定する(VT_I4). 下の表を参照.
戻り値　：[out] ボタン状態の配列. 要素は以下の 2 つ.
0:今までボタンが押された回数 (VT_I4)
1:現在ボタンが押されているか (VT_I4)
0 は押されていない, 1 は押されている

番号	ボタンの種類
0	COBOTTA ファンクションボタン
1	COBOTTA ハンドプラスボタン
2	COBOTTA ハンドマイナスボタン

使用例

```
Dim vntButtonState as Variant
vntButtonState = caoCtrl.Execute("GetSlaveButtonState", 0)
```

5.2.10.23. CaoController::Execute("ClearSlaveButtonState ") コマンド

スレーブコントロール機能有効時に, マスターロボットからスレーブコントローラに通知されているボタン状態をクリアします. このコマンドを実行すると, 今までボタンが押された回数のカウントが 0 にリセットされます.

書式　ClearSlaveButtonState (<lBtnType>)

<lBtnType>　：　クリアする対象を指定する(VT_I4). 上の表を参照.
戻り値　：　なし

使用例

```
caoCtrl.Execute "ClearSlaveButtonState"
```

5.2.11. 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
```

5.2.12. 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 5-12 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
```

5.2.13. 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.

5.2.14. 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.

5.2.15. 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
```

5.2.16. 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
```

5.2.17. 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.

5.2.18. 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.

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%
```

5.2.19. 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 5-10 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
```

5.2.20. 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．

5.2.21. CaoRobot::Halt メソッド

動作中のロボットを停止させます．

RC9 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．

5.2.22. 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.

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″
```

5.2.23. 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.

5.2.24. 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[1].
(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.

[1]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 5-5 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*>“
その他	MOVE P, P<*n1*> +(*x,y,z,rx,ry,rz*)	Move 1, DEV(“P<*n1*>“,"P(*x,y,z,rx,ry,rz*)")
	MOVE P, P<*n1*> +(*x,y,z,rx,ry,rz*)H	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>,…)"

5.2.25. 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.

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.

5.2.26. 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.

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.

5.2.27. 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 5-6 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.53
	TInv	同次変換型の逆行列を計算します	1.0.0	P.53
	TNorm	同次変換型の逆行列を計算します	1.0.0	P.54
	J2T	J 型から T 型へ変換	1.0.0	P.54
	T2J	T 型から J 型へ変換	1.0.0	P.55
	J2P	J 型から P 型へ変換	1.0.0	P.55
	P2J	P 型から J 型へ変換	1.0.0	P.55
	T2P	T 型から P 型へ変換	1.0.0	P.56
	P2T	P 型から T 型へ変換	1.0.0	P.57
	Dev	ベース座標系でのオフセットを計算します	1.0.0	P.57
	DevH	ツール座標系でのオフセットを計算します	1.0.0	P.58

	OutRange	可動範囲かを判定します	1.0.0	P.58
	MPS	mps 単位から SPEED コマンドの値を計算します	1.0.0	P.59
	RPM	rpm 単位から SPEED コマンドの値を計算します	1.0.0	P.59
	DPS	deg/s 単位から SPEED コマンドの値を計算します	1.0.0	P.60
位置取得				
	CurPos	現在位置を P 型で取得します	1.0.0	P.60
	DestPos	目標位置を P 型で取得します	1.0.0	P.61
	CurJnt	現在位置を J 型で取得します	1.0.0	P.62
	DestJnt	目標位置を J 型で取得します	1.0.0	P.62
	CurTrn	現在位置を T 型で取得します	1.0.0	P.64
	DestTrn	目標位置を T 型で取得します	1.0.0	P.66
	CurFig	現在の姿勢 Fig の値を取得します	1.0.0	P.67
	CurPosEx	現在位置をタイムスタンプ付きで P 型で取得します	1.0.0	P.61
	DestPosEx	目標位置をタイムスタンプ付きで P 型で取得します	1.0.0	P.62
	CurJntEx	現在位置をタイムスタンプ付きで J 型で取得します	1.0.0	P.64
	DestJntEx	目標位置をタイムスタンプ付きで J 型で取得します	1.0.0	P.64
	CurTrnEx	現在位置をタイムスタンプ付きで T 型で取得します	1.0.0	P.66
	DestTrnEx	目標位置をタイムスタンプ付きで T 型で取得します	1.0.0	P.67
速度				
	SpeedMode	最適速度制御機能の設定を変更します	1.0.0	P.94
	CurSpd	内部速度の設定値を返します	1.0.0	P.67
	CurAcc	内部加速度の設定値を返します	1.0.0	P.68
	CurDec	内部減速度の設定値を返します	1.0.0	P.69
	CurJSpd	内部軸速度の設定値を返します	1.0.0	P.69
	CurJAcc	内部軸加速度の設定値を返します	1.0.0	P.69
	CurJDec	内部軸減速度の設定値を返します	1.0.0	P.70
ログ				
	StartLog	制御ログの記録を開始します	1.0.0	P.71
	StopLog	制御ログの記録を停止します	1.0.0	P.71
	ClearLog	制御ログのデータをクリアします	1.0.0	P.72
	StartServoLog	サーボログの記録を開始します	1.0.0	P.97

	ClearServoLog	サーボログの記録を消去します	1.0.0	P.97
	StopServoLog	サーボログの記録を停止します	1.0.0	P.98
	GetCtrlLogMaxTime	制御ログの記録時間を取得します	1.0.0	P.98
	SetCtrlLogMaxTime	制御ログの記録時間を設定します	1.0.0	P.98
	GetCtrlLogInterval	制御ログの記録間隔を取得します	1.0.0	P.99
	SetCtrlLogInterval	制御ログの記録間隔を設定します	1.0.0	P.99
	GetLogCount	制御ログデータの個数を取得します	1.0.0	P.100
	GetLogRecord	制御ログデータの情報を取得します	1.0.0	P.100
ロボット動作				
	Motor	モータを ON/OFF します	1.0.0	P.72
	ExtSpeed	外部速度の設定を行います	1.0.0	P.73
	TakeArm	制御権の取得要求を行います	1.0.0	P.73
	GiveArm	制御権の解放要求を行います	1.0.0	P.74
	Draw	ワーク座標系指定で相対動作を行います	1.0.0	P.75
	Approach	ツール座標系指定で絶対動作を行います	1.0.0	P.76
	Depart	ツール座標系指定で相対動作を行います	1.0.0	P.77
	DriveEx	各軸の相対動作を行います	1.0.0	P.78
	DriveAEx	各軸の絶対動作を行います	1.0.0	P.79
	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.80
	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.80
	MotionSkip	実行中のロボットの動作を中断します	1.0.0	P.81
	MotionComplete	ロボット動作が完了したかを判断します	1.0.0	P.81
	MoveMasterPos	マスターロボットの各軸角度を目標としてスレーブコントローラに接続されているロボットを動作させます	1.0.0	P.101
ツール				
	CurTool	現在のツール番号を取得します	1.0.0	P.82
	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.82
	SetToolDef	ツール定義を設定します	1.0.0	P.83
ワーク				
	CurWork	現在のワーク番号を取得します	1.0.0	P.83
	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.83
	SetWorkDef	ワーク定義を設定します	1.0.0	P.84

	WorkAttribute	ワーク番号で指定したワーク属性を取得します	1.0.0	P.84
ベース				
	SetBaseDef	Base 定義を設定します	1.0.0	P.96
	GetBaseDef	Base 番号で指定した Base 定義を取得します	1.0.0	P.96
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.85
	SetAreaDef	エリアのパラメータを設定します	1.0.0	P.85
	SetArea	エリアチェックを有効化します	1.0.0	P.86
	ResetArea	エリアチェックを無効化します	1.0.0	P.86
	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.87
	GetAreaEnabled	エリアの有効/無効を取得します	1.0.0	P.87
	SetAreaEnabled	エリアの有効/無効を設定します	1.0.0	P.87
HandIO				
	SetHandIO	HandIO を設定します	1.0.0	P.96
	GetHandIO	HandIO を取得します	1.0.0	P.97
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	1.0.0	P.88
	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	1.0.0	P.89
	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	1.0.0	P.89
	HighPathAccuracy	高軌跡制御機能の有効/無効を切り替えます	1.0.0	P.92
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.90
	GetSrvJntData	指定軸のサーボ内部データを返します	1.0.0	P.91
	RobInfo	ロボットの情報を返します	1.0.0	P.95
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.92
	MotionTimeout	動作命令のタイムアウト設定値を変更します	1.0.0	P.93
	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.90

	PayLoad	内部負荷条件の設定値を変更します	1.0.0	P.94
	SingularAvoid	特異点回避機能を有効化，または無効化します	1.0.0	P.93
その他				
	GetRobotTypeName	ロボット型式を取得します	1.0.0	P.88
	GetPluralServoData	サーボデータを一括で取得します	1.0.0	P.99
	GenerateNonStopPath	無停止教示点補正機能オプションのコマンドです	1.0.0	P.94

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
```

5.2.27.1. 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
```

5.2.27.2. 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
```

5.2.27.3. 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
```

5.2.27.4. 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
```

5.2.27.5. 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
```

5.2.27.6. 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
```

5.2.27.7. 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
```

5.2.27.8. 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
```

5.2.27.9. 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
```

5.2.27.10. 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
```

5.2.27.11. 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
```

5.2.27.12. 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)" )
```

5.2.27.13. 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
```

5.2.27.14. 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
```

5.2.27.15. 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
```

5.2.27.16. 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
```

5.2.27.17. 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
```

5.2.27.18. 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
```

5.2.27.19. 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
```

5.2.27.20. 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
```

5.2.27.21. 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
```

5.2.27.22. 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
```

5.2.27.23. 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
```

5.2.27.24. 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
```

5.2.27.25. 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
```

5.2.27.26. 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
```

5.2.27.27. 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
```

5.2.27.28. 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
```

5.2.27.29. 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)
```

5.2.27.30. 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)
```

5.2.27.31. 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)
```

5.2.27.32. 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)
```

5.2.27.33. 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)
```

5.2.27.34. 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)
```

5.2.27.35. 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)
```

5.2.27.36. 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)
```

5.2.27.37. 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)
```

5.2.27.38. 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.

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"
```

5.2.27.39. 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"
```

5.2.27.40. 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"
```

5.2.27.41. 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
```

5.2.27.42. 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.

5.2.27.43. 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
```

5.2.27.44. 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’
```

5.2.27.45. 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.

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,"V0")
caoRob.Execute"Draw",  Array(2,"V(100, 100, 100)")
```

5.2.27.46. 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.

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")
```

5.2.27.47. 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.

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")
```

5.2.27.48. 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.

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")
```

5.2.27.49. 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.

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")
```

5.2.27.50. 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.

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" )
```

5.2.27.51. 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.

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
```

5.2.27.52. 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.

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)
```

5.2.27.53. 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
```

5.2.27.54. 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")
```

5.2.27.55. 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)
```

5.2.27.56. 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)")
```

5.2.27.57. 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")
```

5.2.27.58. 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)
```

5.2.27.59. 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)")
```

5.2.27.60. 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)
```

5.2.27.61. 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)
```

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

エリアのパラメータを設定します.

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)
```

5.2.27.63. 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
```

5.2.27.64. 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
```

5.2.27.65. 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)
```

5.2.27.66. 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
```

5.2.27.67. 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
```

5.2.27.68. 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" )
```

5.2.27.69. 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
```

5.2.27.70. 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
```

5.2.27.71. 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)
```

5.2.27.72. 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)
```

5.2.27.73. 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)
```

5.2.27.74. 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)
```

5.2.27.75. 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
```

5.2.27.76. 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
```

5.2.27.77. 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
```

5.2.27.78. 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
```

5.2.27.79. 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
```

5.2.27.80. 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)")
```

5.2.27.81. 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))
```

5.2.27.82. 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)
```

5.2.27.83. 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)” )
```

5.2.27.84. 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)
```

5.2.27.85. 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)
```

5.2.27.86. 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))
```

5.2.27.87. 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”
```

5.2.27.88. 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”
```

5.2.27.89. 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”
```

5.2.27.90. 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” )
```

5.2.27.91. 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
```

5.2.27.92. 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” )
```

5.2.27.93. 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
```

5.2.27.94. 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” )
```

5.2.27.95. 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″)
```

5.2.27.96. 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
```

5.2.27.97. CaoRobot::Execute("MoveMasterPos") コマンド

スレーブコントロール機能有効時に, マスターロボットの各軸角度を目標位置としてロボットが動作します.

PacScript 言語の MOVEMASTERPOS 命令に対応します.

書式 MoveMasterPos ()

引数 : なし

戻り値 : なし

使用例

```
caoRob.Execute  "MoveMasterPos"
```

5.2.28. 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 5-11 を参照して下さい.

5.2.29. 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.

5.2.30. 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)

5.2.31. 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.

5.2.32. 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 5-7 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. 103
プライオリティ			
	GetThreadPriority	優先度(プライオリティ)の取得	P. 103
	SetThreadPriority	優先度(プライオリティ)の設定	P. 104

5.2.32.1. 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")
```

5.2.32.2. 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

5.2.32.3. 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

5.2.33. 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.

5.2.34. 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.

5.3. 変数一覧

5.3.1. コントローラクラス

表 5-8 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 5-9 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	○	△[2]
@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	○	

[2]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.

@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	○	-
@ENABLE_SW	VT_BOOL	イネーブルスイッチの状態	○	-
@AUTO_ENABLE	VT_BOOL	自動イネーブルの状態	○	-
@MAKER_NAME	VT_BSTR	“DENSO CORPORATION”	○	-
@TYPE	VT_BSTR	“RC9 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	○	-

5.3.2. ロボットクラス

表 5-10 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	○	○
@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.	○	-
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 5-11 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	現在実行中のメインプログラムの行番号とファイル ID. [0] = 行番号 [1] = 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. ms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 5-12 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 5-13 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 1 以上のエラー発生時
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 AUTO:3	-

[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("RC9", "CaoProv.DENSO.RC9", "",
"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. 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[3] "[<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[4] <即値> = (<要素 1:VT_R4>,<要素 2:VT_R4>,...)[5]
- 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

[3] 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.
[4] <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.
[5] <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A.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")[6]
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))

[6] 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 （変数型, パスは数値指定）	Array(1.56,-1,-1)

[<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>,…)
```