

EPSON

EPSON RC+ 7.0

Ver.7.5

SPEL⁺ランゲージ リファレンス

Rev.9

JAM23YS6228F

翻訳版

EPSON RC+ 7.0 (Ver.7.5) SPEL+ ランダーズリファレンス Rev.9

EPSON RC+ 7.0 (Ver.7.5)

SPEL⁺ ランゲージリファレンス

Rev.9

©Seiko Epson Corporation 2012-2023

はじめに

このたびは当社のロボットシステムをお求めいただきましてありがとうございます。
本マニュアルは、EPSON RC+ 7.0 ソフトウェアを正しくお使いいただくために必要な事項を記載したものです。
システムをご使用になる前に、本マニュアルおよび関連マニュアルをお読みいただき、正しくお使いください。
お読みになった後は、いつでも取り出せる所に保管し、不明な点があったら再読してください。

当社は、厳密な試験や検査を行い、当社のロボットシステムの性能が、当社規格に満足していることを確認しております。マニュアルに記載されている使用条件を超えて、当社ロボットシステムを使用した場合は、製品の基本性能は発揮されませんのでご注意ください。

本書の内容は、当社が予見する範囲の、危険やトラブルについて記載しています。当社のロボットシステムを、安全に正しくお使いいただくため、本書に記載されている安全に関するご注意は、必ず守ってください。

商標

Microsoft, Windows, Windowsロゴ, Visual Basic, Visual C++は、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
Pentium は米国インテル社の商標です。
その他の社名、ブランド名、製品名は、各社の登録商標または商標です。

表記について

Microsoft® Windows® 8 operating system 日本語版
Microsoft® Windows® 10 operating system 日本語版
Microsoft® Windows® 11 operating system 日本語版
本取扱説明書では、上記オペレーティングシステムをそれぞれ、Windows 8, Windows 10, Windows 11と表記しています。また、Windows 8, Windows 10, Windows 11を総称して、Windowsと表記することがあります。

ご注意

本取扱説明書の一部、または全部を無断で複製や転載をすることはできません。
本書に記載の内容は、将来予告なく変更することがあります。
本書の内容について、誤りや、お気づきの点がありましたら、ご連絡くださいますようお願いいたします。

製造元

セイコーエプソン株式会社

お問い合わせ先

お問い合わせ先の詳細は、以下のマニュアル冒頭“販売元”に記載しています。

「ロボットシステム 安全マニュアル はじめにお読みください」

ご使用の前に

マニュアルのご使用の前に、知っておいていただきたいことを記載しています。

安全上の注意

ロボットおよび関連機器の運搬と設置は、有資格者が行ってください。また、必ず各国の法規と法令に従ってください。

ご使用になる前に、本マニュアル、ならびに関連マニュアルをよくお読みの上、正しくお使いください。

お読みになった後は、いつでも取り出せる所に保管し、不明な点があったら再読してください。

記号の意味

 警告	この表示を無視して誤った取り扱いをすると、人が死亡、または重傷を負う可能性が想定される内容を示しています。
 注意	この表示を無視して誤った取り扱いをすると、人が傷害を負う可能性が想定される内容および物的損害のみの発生が想定される内容を示しています。

SPEL+ コマンド一覧	1
システム管理に関する命令	1
ロボット制御に関する命令	2
トルクに関する命令	6
入出力に関する命令	7
ポイント制御に関する命令	9
座標変換に関する命令	10
プログラム制御に関する命令	11
プログラムの実行に関する命令	12
擬似命令	12
ファイル管理に関する命令	12
フィールドバスに関する命令	13
数値に関する命令	13
文字列に関する命令	14
論理演算に関する命令	15
変数に関する命令	15
セキュリティに関する命令	15
コンベヤトラッキングに関する命令	16
フォースセンシングに関する命令	17
DBに関する命令	17
PGに関する命令	17
衝突検出に関する命令	17
部品消耗管理に関する命令	18
シミュレーターに関する命令	18
ハンドに関する命令	19
安全機能に関する命令	19
SPEL+ ランゲージリファレンス	20
Appendix A: SPEL+ コマンド 使用条件一覧	875
Appendix B: 互換性に関する注意	887
B-1: EPSON RC+ 6.0 からの互換性に関する注意	887
B-2: EPSON RC+ 5.0 からの互換性に関する注意	897
B-3: EPSON RC+ Ver.4.* からの互換性に関する注意	908

Appendix C: EPSON RC+7.0 のコマンド	919
C-1: EPSON RC+ 4.0以降で追加されたコマンド一覧	919
C-2: EPSON RC+ 7.0のバージョンごとに追加されたコマンド一覧	922
C-3: EPSON RC+ 7.0のバージョンごとに削除されたコマンド一覧	928
Appendix D: 定数	929

SPEL+ コマンド一覧

SPEL+ コマンド一覧です。

システム管理に関する命令

Reset	コントローラーを初期状態にリセット
SysConfig	システム設定パラメーターを表示
SysErr	最新のエラーステータス、またはワーニングステータスを返す
Date	日付の表示
Time	時刻の表示
Date\$	日付を文字列で返す関数
Time\$	時刻を文字列で返す関数
Hour	コントローラーの積算通電時間の表示
Stat	コントローラーの状態ビットを返す
CtrlInfo	コントローラーの情報を返す
RobotInfo	ロボットの情報を返す
RobotInfo\$	ロボットのテキスト情報を返す
TaskInfo	タスク情報を返す
TaskInfo\$	タスクのテキスト情報を返す
DispDev	現在のディスプレイデバイスを設定
EStopOn	非常停止状態を返す
CtrlDev	現在の制御デバイスの番号を返す
Cls	Run ウィンドウ, オペレーターウィンドウ, コマンドウィンドウのテキスト領域をクリア、TP のプリントパネルをクリア
Toff	LCD に実行行の表示をオフ
Ton	LCD に実行行の表示をオン
SafetyOn	安全扉の状態を返す
Eval	コマンドウィンドウのステートメントを実行し、エラー状態を返す
ShutDown	EPSON RC+ をシャットダウンし、Windows をシャットダウンまたは再起動
TeachOn	ティーチモードの状態を返す
WindowsStatus	Windows の起動状態を返す

 ロボット制御に関する命令

AIO_TrackingSet	距離追従機能の設定
AIO_TrackingStart	距離追従機能の開始
AIO_TrackingEnd	距離追従機能の終了
AIO_TrackingOn 関数	距離追従機能の状態を返す関数
AtHome	現在のロボット姿勢が Home 姿勢か否かを返す関数
Calib	現在のアーム姿勢のパルス値を、Calpls により設定されているパルス値に置き換える
CalPls	キャリブレーションに使用する位置姿勢パルス値の設定と表示、または設定値を返す関数
Hofs	エンコーダー原点からソフトウェア原点への補正パルスの設定と表示
JointAccuracy	関節精度補正の補正值の設定と表示
HofsJointAccuracy	関節精度補正を変更せずに、エンコーダー原点からソフトウェア原点への補正パルスを設定
MCal	インクリメンタルエンコーダーロボットの原点復帰 (メカ原点の検出)
MCalComplete	MCal の状態を返す
MCordr	MCal 命令による原点復帰時の関節の動作順序の指定と表示、または設定値を返す関数
Power	パワーモードを High または Low にし、現在のモードを表示
MHour 関数	ロボットのモーターの積算励磁時間を返す関数
Motor	ロボットの全軸のモーターオンまたはオフ
SFree	指定した関節をフリージョイント状態にします
SLock	指定された関節のフリージョイント状態を解除します
SyncRobots	動作予約中のロボットの動作を開始
Jump	目標座標にゲートモーションで移動する動作 (PTP 動作)
Jump3	目標座標に 3 次元ゲート動作で移動する動作 (PTP 動作+CP 動作)
Jump3CP	目標座標に 3 次元ゲート動作で移動する動作 (CP 動作)
JumpTLZ	目標座標に 3 次元ゲート動作で移動する動作 (PTP 動作+CP 動作)
Arch	Jump 命令でのアーチパラメーターの設定と表示、および設定値を返す関数
LimZ	Jump コマンドのときの第 3 関節の高さ (Z 座標値)の初期値を設定、および設定値を返す関数
LimZMargin	LimZ 設定値より高い位置で動作開始したときのエラー検出のマージンを設定、および設定値を返す関数
Sense	Jump 命令で Sense 指定したときに、目標座標上方で停止させるための条件の設定と表示
JS	Jump Sense 実行後、Sense 入力が成立したのかどうかを返す関数
JT	直前の Jump 動作の結果を返す関数
Go	現在位置から指定位置まで PTP 動作で動かす
Pass	指定点の近傍を停止しないで通過する PTP 動作
Pulse	各関節のパルス値で指定されたポイントまで、アームを PTP 制御で動かす
BGo	選択したローカル座標系でのオフセット PTP 動作

BMove	ローカル座標系でのオフセット直線補間動作
TGo	ツール座標系上でのオフセット PTP 動作
TMove	ツール座標系上でのオフセット直線補間動作
Till	動作命令で Till 指定したときに、動作途中で停止し、処理を終了させる条件の設定と表示
TillOn	Till の状態を返す
!...!	動作中に I/O などの入出力処理を並列して行う機能
Speed	PTP 動作速度の設定と表示、および設定値を返す関数
Accel	PTP 動作の加減速度の設定と表示、および設定値を返す関数
SpeedFactor	PTP 動作速度の設定と表示、および設定値を返す関数
Inertia	ロボットの負荷イナーシャと偏心量を設定
Weight	PTP 動作時の、速度/加減速度を補正するためのパラメーター設定と表示
Arc	XY 平面における円弧補間動作 (CP 動作)
Arc3	3 次元における円弧補間動作 (CP 動作)
Move	直線補間動作 (CP 動作)
Curve	自由曲線による CP 制御のためのデータとポイントを作成
CVMove	Curve 命令で定義した自由曲線 CP 動作を実行
SpeedS	CP 動作速度の設定と表示、および設定値を返す関数
AccelS	CP 動作の加減速度の設定と表示、および設定を返す関数
SpeedR	ツール姿勢変化の速度の設定と表示、および設定値を返す関数
AccelR	ツール姿勢変化の加減速度の設定と表示、および設定値を返す関数
AccelMax	Accel で設定可能な加減速度の最大値を返す関数
Brake	現在のロボットの指定した関節のブレーキのオンオフ
Home	ホーム位置 (待機姿勢) への移動 (PTP 動作)
HomeClr	ホーム位置の定義をクリア
HomeDef	ホーム位置の定義を返す
HomeSet	ホーム位置 (待機位置) のパルスの設定と表示
Hordr	Home 実行時の各関節の動作順序の指定と表示、および設定値を返す関数
InPos	ロボットの位置決め状態を返す関数
CurPos	ロボットの現在の動作目標位置を返す
TCPSpeed	算出された現在のツール先端速度を返す
Pallet	パレットの定義と定義パレットの表示
PalletClr	パレットの定義をクリア
Fine	目標位置の位置決め終了判断範囲の設定と表示 (単位: pulse)
FineDist	目標位置の位置決め終了判断範囲の設定と表示 (単位: mm)
FineStatus 関数	Fine を使用しているか FineDist を使用しているかを整数値で返す

QP	クイックポーズ機能の設定/解除と、現在の設定の表示、および設定値を返す関数
QPDcelR	CP 動作時のツール姿勢変化について、クイックポーズ減速度を設定
QPDcelS	CP 動作時のクイックポーズ減速度を設定
CP	パスモーションを設定
Box	進入検出エリアの設定と表示
BoxClr	設定された Box をクリア
BoxDef	進入検出エリアが設定されているかどうかを返す
Plane	進入検出平面の設定と表示
PlaneClr	進入検出平面をクリア (未定義)
PlaneDef	進入検出平面の設定状態を返す
InsideBox	進入検出エリアの検出状態を返す
InsidePlane	進入検出平面の検出状態を返す
GetRobotInsideBox	進入検出エリアに進入しているロボットを返す
GetRobotInsidePlane	進入検出平面に進入しているロボットを返す
Find	動作命令中に座標を保存するための条件の設定と表示
FindPos	動作命令実行中に Find で保存した座標を返す
PosFound	Find 命令時実行の状態を返す
WaitPos	パスモーションが有効な状態でも、次のステートメントを実行する前にロボットが減速停止するのを待つ
Robot	ロボットの選択 (Robot 関数でロボット番号を返す)
RobotModel\$	ロボットの型名を返す
RobotName\$	ロボット名を返す
RobotSerial\$	ロボットのシリアル番号を返す
RobotType	ロボットタイプを返す
TargetOK	現在位置から目標座標への PTP (point to point) 動作が可能かどうかを返す
ROTOK 関数	目標座標への動作命令時に、 ROT 修飾パラメーターが付加可能かどうかを返す
JRange	パルス値による指定関節の許容動作エリアの設定、および設定値を返す関数
Range	パルス値による許容動作エリアの設定と表示
XYLim	XY 平面許容動作エリアの設定と表示、および設定値を返す関数
XYLimClr	設定された XYLim をクリア
XYLimDef	XYLim を設定したかどうかを返す
XYLimMode	XYLim の監視方法の設定と表示、および設定値を返す関数
XY	指定された座標値をポイントデータとして返す
Dist	2 つのロボット座標間の距離を返す
DiffToolOrientation 関数	ツール座標系の各座標軸のなす角度を返す

DiffPoint 関数	指定された 2 つのポイントの差分を返す
PTPBoost	PTP 動作微小移動時の設定と表示、および設定値を返す関数
PTPBoostOK	現在位置から目標座標への PTP (point to point) 動作が微小移動であるかどうかを返す
PTPTime	PTP 動作命令の推定所要時間を返す、PTP 動作は行なわない
CX	指定したポイントデータの X 座標値の設定と設定値を返す関数
CY	指定したポイントデータの Y 座標値の設定と設定値を返す関数
CZ	指定したポイントデータの Z 座標値の設定と設定値を返す関数
CU	指定したポイントデータの U 座標値の設定と設定値を返す関数
CV	指定したポイントデータの V 座標値の設定と設定値を返す関数
CW	指定したポイントデータの W 座標値の設定と設定値を返す関数
CR	指定したポイントデータの R 座標値の設定と設定値を返す関数
CS	指定したポイントデータの S 座標値の設定と設定値を返す関数
CT	指定したポイントデータの T 座標値の設定と設定値を返す関数
PIs	各関節のパルス値を返す関数
AgI	指定関節の角度または位置を返す関数
PAgI	指定した座標値から関節位置 (回転関節の角度, 直動関節の位置) を計算して返す関数
JA	関節の角度から計算したロボット座標を返す
AgItoPIs	ロボットの各関節の角度をパルスに変換
DegToRad	角度をラジアンに変換
RadToDeg	ラジアンを角度に変換
Joint	現在のロボットの位置を、関節の座標系で表示
JTran	原点復帰を必要としない 1 関節のみの PTP 動作を行う
PTran	現在位置から指定したパルス量を、1 関節のみ PTP 動作
RealPIs	指定した関節のパルス値を返す
RealPos	指定したロボットの現在の位置を返す
RealAccel 関数	OLAccel で自動調整した Accel の値を返す関数
PPIs	指定した座標値から関節パルスを計算して返す関数
LJM 関数	指定ポイントに対して参照ポイントから見て最も関節移動量が少なくなるように姿勢フラグを変換したポイントデータを返す
AutoLJM	自動 LJM を設定
AutoLJM 関数	自動 LJM の状態を返す関数
AutoOrientationFlag	N6-A1000**の姿勢フラグを変更
AutoOrientationFlag 関数	AutoOrientationFlag の状態を返す関数
AvoidSingularity	特異姿勢回避機能を設定
AvoidSingularity 関数	特異姿勢回避機能の状態を返す関数
SingularityAngle	特異姿勢回避機能の特異姿勢近傍角度を設定
SingularityAngle 関数	特異姿勢回避機能の特異姿勢近傍角度を返す関数
SingularitySpeed	特異姿勢回避機能の特異姿勢近傍速度を設定
SingularitySpeed 関数	特異姿勢回避機能の特異姿勢近傍速度を返す関数
SingularityDist	特異姿勢通過機能に必要な特異点近傍距離を設定
SingularityDist 関数	特異姿勢通過機能に必要な特異点近傍距離を返す関数

AbortMotion	動作命令を中断し動作を実行していたタスクをエラーにする
Align 関数	ロボットの姿勢をローカル座標系の最も近い座標軸に整列するよう変換したポイントデータを返す関数
AlignECP 関数	ロボットの姿勢を ECP 座標系の最も近い座標軸に整列するよう変換したポイントデータを返す関数
SoftCP	SoftCP 動作モードの設定と表示
SoftCP 関数	SoftCP 動作モードの状態を返す関数
Here	現在位置をロボット座標として教示
Where	ロボットの現在位置データを表示
PerformMode	ロボットの動作モードを設定
PerformMode 関数	ロボットの動作モード番号を返す
VSD	スカラロボットの可変速度 CP 動作機能を設定する
VSD 関数	スカラロボットの可変速度 CP 動作機能の設定を返す
CP_Offset	CP On 時の後続する動作命令の開始オフセット時間を設定
CP_Offset 関数	CP On 時の後続する動作命令の開始オフセット時間を返す
AvgSpeedClear	関節の平均速度をクリアし、初期化
AvgSpeed	関節の平均速度を表示
AvgSpeed 関数	関節の平均速度を返す関数
PeakSpeedClear	関節のピーク速度をクリアし、初期化
PeakSpeed	関節のピーク速度を表示
PeakSpeed 関数	関節のピーク速度を返す関数

トルクに関する命令

TC	トルク制御モードの設定および現在のモードを表示する関数
TCSpeed	トルク制御中の速度制限値を設定する関数
TCLim	トルク制御モードのための各関節のトルク制限値を設定する関数
RealTorque	指定した関節の現在のトルク指令値を返す関数
ATCLR	関節の実効トルクをクリアし、初期化
ATRQ	指定した関節の実効トルク値を表示
PTCLR	関節のピークトルクをクリアし、初期化
PTRQ	指定した関節のピークトルク値を表示
OLAccel	過負荷率に応じた加減速度の自動調整を設定
OLRate	指定関節の過負荷率を表示
LimitTorque	ハイパワーモード時のトルク上限値を設定、および設定値を返す関数
LimitTorque 関数	LimitTorque 命令の設定値を返す関数
LimitTorqueLP	ローパワーモード時のトルク上限値を設定、および設定値を返す関数
LimitTorqueLP 関数	LimitTorqueLP 命令の設定値を返す関数
LimitTorqueStop	ハイパワーモードでトルク上限に達した場合に、ロボットを停止するかどうかを設定、および設定値を返す関数

LimitTorqueStop 関数	LimitTorqueStop 命令の設定値を返す関数
LimitTorqueStopLP	ローパワーモードでトルク上限に達した場合に、ロボットを停止するかどうかを設定、および設定値を返す関数
LimitTorqueStopLP 関数	LimitTorqueStopLP 命令の設定値を返す関数

入出力に関する命令

On	出力ビットをオンします。
Off	出力ビットをオフします。
Oport	出力ビットの状態を読み込み
Sw	I/O 入力またはメモリーI/O の状態を読み込み、I/O で使用します。
In	1 バイト (8 ビット) の入力データを読み込み、I/O で使用します。
InW	1 ワード (16 ビット) の入力データを読み込み、I/O で使用します。
InBCD	入力ポートの入力状態を BCD (バイナリーコード化された 10 進数) フォーマットで読み込み
Out	1 バイト (8 ビット) の出力データを出力、I/O で使用します。
OutW	1 ワード (16 ビット) の出力データを出力、I/O で使用します。
OpBCD	出力ポートに BCD フォーマットで 1 バイトのデータを出力
MemOn	ビット番号で指定されたメモリーI/O をオンする命令
MemOff	ビット番号で指定されたメモリーI/O をオフする命令
MemSw	指定されたメモリーI/O ビットの状態を返す関数
MemIn	メモリーI/O の状態をバイト単位で返す関数
MemOut	メモリーI/O の状態をバイト単位で変更する命令
MemInW	メモリーI/O ポートの状態をワード単位で返す、ワードポートは 16 個のメモリーI/O ビットで構成
MemOutW	メモリーI/O ポートの状態を 16 ビット同時にワード単位で設定
Wait	イベント条件待機
TMOut	Wait 命令実行時のタイムアウト時間の設定
TW	Wait 命令の条件式が成立したかどうかを返す関数
Input	ディスプレイデバイスから入力を受け取り、変数に保存
InReal	2 ワード (32 ビット) の入力データを 32 ビット浮動小数点データ (IEEE754 準拠) として読み込みます。I/O で使用します。
Print	出力画面、コマンド画面、オペレーターウィンドウへのデータ表示
Line Input	1 行入力データを読み、そのデータを文字列変数に代入
Input #	ファイル、通信ポート、データベース、またはデバイスから、文字列か数値をデータ入力し、変数に保存
Print #	指定されたファイル、通信ポート、データベース、デバイスにデータを出力
Line Input #	ファイル、通信ポート、データベース、デバイスから、1 行データを読み込み
Lof	指定した RS-232C ポート、または TCP/IPport のバッファの受信データの行数を返す
SetIn	仮想 I/O の入力ポート (8 ビット) を設定
SetInW	仮想 I/O の入力ポート (16 ビット) を設定

SetSw	仮想 I/O の入力を設定
IOLabel\$	指定した入出力のビット, バイト, ワードの I/O ラベルを返す
IONumber	指定した I/O ラベルの I/O ポート番号を返す
IODef	指定した I/O ラベルが定義されているかを返す
OpenCom	RS-232C ポートをオープン
OpenCom 関数	OpenCom を実施しているタスク番号を取得する関数
CloseCom	OpenCom でオープンした RS-232C ポートをクローズ
SetCom	RS-232C ポートのパラメーターを設定または表示
ChkCom	通信ポートの受信バッファ内の文字数を返す
OpenNet	TCP/IP ネットワークポートをオープン
OpenNet 関数	OpenNet を実施しているタスク番号を取得する関数
OutReal	実数値の出力データを 32 ビット浮動小数点データ (IEEE754 準拠) として出力ポート 2 ワード (32 ビット) に出力します。I/O で使用します。
CloseNet	OpenNet でオープンした TCP/IP ポートをクローズ
SetNet	TCP/IP ポートのパラメーターを設定
ChkNet	ネットワークポートの受信バッファ内の文字数を返す
WaitNet	TCP/IP ポートの接続が確立するまで待つ
Read	ファイル、または通信ポートから指定した文字数を読み込む
ReadBin	ファイル、または通信ポートからバイナリーデータを読む
Write	ファイル、または通信ポートに文字列を書き込む、行末ターミネーターは付加しない
WriteBin	ファイル、または通信ポートへバイナリーデータを書き出す
InputDialog	入力ダイアログボックスを表示し、入力した内容を返す
MsgBox	ダイアログボックスのメッセージの表示
RunDialog	SPEL+ のプログラムから EPSON RC+ 画面を起動
LatchEnable	R-I/O 入力によるロボット位置のラッチ機能を有効/無効にする
LatchState 関数	R-I/O によるロボット位置のラッチ状態を返す
LatchPos 関数	R-I/O 入力信号でラッチしたロボットの位置を返す
SetLatch	R-I/O 入力によるロボット位置のラッチ機能を設定
AIO_In 関数	アナログ I/O 入力チャンネルからアナログ値を読み込む
AIO_InW 関数	アナログ I/O 入力チャンネルから 1 ワードの入力データを読み込む
AIO_Out	アナログ I/O 出力チャンネルにアナログ値を出力
AIO_Out 関数	アナログ I/O 出力チャンネルの出力状態を返す
AIO_OutW	アナログ I/O 出力チャンネルに 1 ワードのデータを出力
AIO_OutW 関数	アナログ I/O 出力チャンネルの出力状態を 1 ワードで返す
AIO_Set	アナログ I/O 出力チャンネルに速度情報を出力する
AIO_Set 関数	オプションのアナログ I/O 出力チャンネルに設定されているロボット速度出力の設定情報を返す

ポイント制御に関する命令

ClearPoints	メモリー上の位置データを消去
LoadPoints	ポイントファイルを、ロボットのポイントメモリーエリアに読込
SavePoints	メインメモリーにあるポイントデータを、カレントロボットのディスクファイルに保存
ImportPoints	現在選択中のロボットのプロジェクトにポイントファイルをインポート
ExportPoints	ポイントファイルを PC の指定パスにエクスポート
P#	ポイントデータを定義
PDef	指定したポイントデータが定義されているかを返す
PDel	指定したポイントデータを削除
PLabel	指定したポイントデータのラベルを設定
PLabel\$	指定したポイント番号に定義されたポイントラベルを返す
PNumber	ポイントラベルに対応するポイント番号を返す
PList	現在のロボットのメモリー上のポイントデータを表示
PLocal	指定したポイントデータのローカル属性を設定
PDescription	指定したポイントデータのコメントを設定
PDescription\$	指定したポイント番号に定義されたポイントのコメントを返す
WorkQue_Add	指定したワークキューにワークキューデータ (ポイントデータとユーザーデータ)を追加
WorkQue_AutoRemove	指定したワークキューに自動削除機能を設定
WorkQue_AutoRemove 関数	ワークキューに設定されている自動削除機能の状態を返す
WorkQue_Get 関数	指定したワークキューから、ポイントデータを返す
WorkQue_Len 関数	指定したワークキューに登録された有効なワークキューデータの数を返す
WorkQue_List	指定したワークキューのワークキューデータ一覧 (ポイントデータとユーザーデータ)を表示
WorkQue_Reject	指定したワークキューにポイントデータの 2 重登録防止のための最小距離を設定、表示
WorkQue_Reject 関数	指定したワークキューに設定されている 2 重登録防止距離を返す
WorkQue_Remove	指定したワークキューからワークキューデータ (ポイントデータとユーザーデータ)を削除
WorkQue_Sort	指定したワークキューに Sort 方法を設定、表示
WorkQue_Sort 関数	指定したワークキューに設定されている Sort 方法を返す
WorkQue_UserData	指定したワークキューに登録されているユーザーデータ (実数)の再設定、表示
WorkQue_UserData 関数	指定したワークキューに登録されているユーザーデータ (実数)を返す

座標変換に関する命令

AreaCorrection 関数	補正エリアを用いて補正をしたポイントを返す
AreaCorrectionClr	補正エリアをクリア
AreaCorrectionDef	補正エリアの設定を返す
AreaCorrectionInv 関数	補正済みのポイントをもとに戻す
AreaCorrectionOffset 関数	補正済みのポイントから相対移動したポイントを返す
AreaCorrectionSet	補正エリアの設定と表示
Arm	アームの選択と、現在選択されているアーム番号の表示または設定値を返す関数
ArmSet	増設アームの設定と表示
ArmDef	アームの設定を返す
ArmClr	アームの設定をクリア
ArmCalib	アーム長補正の有効、または無効の設定と表示
ArmCalibClr	アーム長補正の設定をクリア
ArmCalibDef	アーム長補正の設定を返す
ArmCalibSet	アーム長補正の設定と表示
Tool	ツールの選択、または選択されているツール番号の表示、または設定値を返す関数
TLSet	ツール座標系の設定と表示
TLDef	ツール座標系の設定を返す
TLClr	ツール座標系の設定をクリア
ECP	ECP の選択、または選択されている ECP 番号の表示、または設定値を返す関数
ECPSet	外部制御点 (ECP) の定義と表示
ECPDef	ECP 設定を返す
ECPClr	ECP 設定をクリア
Base	ベース座標系の定義と表示
Local	ローカル座標系データの定義と表示
LocalDef	ローカル座標系の設定を返す
LocalClr	ローカル座標系をクリア (未定義)
Elbow	ポイントの肘姿勢の設定と設定を返す関数
Hand	ポイントのハンド(腕系)姿勢の設定と設定を返す関数
Wrist	ポイントの手首姿勢の設定と設定を返す関数
J4Flag	ポイントの J4Flag 値の設定と設定値を返す関数
J6Flag	ポイントの J6Flag 値の設定と設定値を返す関数
J1Flag	ポイントの J1Flag 値の設定と設定値を返す関数
J2Flag	ポイントの J2Flag 値の設定と設定値を返す関数
J1Angle	ポイントの J1Angle 値の設定と設定値を返す関数
J4Angle	ポイントの J4Angle 値の設定と設定値を返す関数
VxCalib	Vision Guide 以外のビジョンシステムのキャリブレーションデータを作成

VxTrans	ピクセル座標からロボット座標への座標変換を行い、変換したポイントデータを返す関数
VxCalInfo	Vision Guide 以外のビジョンシステムのキャリブレーションの終了状態、キャリブレーションデータを返す関数
VxCalDelete	Vision Guide 以外のビジョンシステムのキャリブレーションデータを削除
VxCalSave	Vision Guide 以外のビジョンシステムのキャリブレーションデータをファイルへ保存
VxCalLoad	Vision Guide 以外のビジョンシステムのキャリブレーションデータをファイルから読み込み

プログラム制御に関する命令

Function	ファンクションの定義
For...Next	For...Next 間にある一連のステートメントを繰り返し実行
GoSub	サブルーチンの実行
Return	サブルーチンからプログラムを戻す
GoTo	プログラム制御を指定のラベルに移す
Call	ファンクションのサブルーチンとしての呼び出し
If...Then...Else...EndIf	プログラムの条件分岐
Else	If...Then...Else 形式で用い、条件式が満たされなければ、Else に続くステートメントが実行、Else は省略可能
Select ... Send	比較値にしたがって、実行するステートメントを選択
Do...Loop	ループの先頭行もしくは最終行で条件判断を行い、条件が一致または不一致の場合 Do...Loop 間を繰り返す
Declare	DLL (ダイナミックリンクライブラリー)で定義されている外部ファンクションを呼び出す
Trap	割り込み、および割り込み発生時の処理の定義
OnErr	エラー処理ルーチンの位置を定義
Era	エラーが発生した関節の番号を返す関数
Err\$	エラーが発生した関数の名前を返す
Erl	エラーが発生した行番号を返す関数
Err	エラーコードを返す関数
Ert	エラーが発生したタスクの番号を返す関数
Errb	エラーが発生したロボットの番号を返す関数
ErrMsg\$	エラーコードに対応したエラーメッセージを返す関数
Signal	WaitSig 命令を実行中のタスクに信号を送る
SyncLock	相互排他ロックを用いて複数のタスクを同期
SyncUnlock	あらかじめ SyncLock でロックされた、シグナル番号のロックを解除
WaitSig	他のタスクからの Signal 命令による同期シグナルを待つ
ErrorOn	コントローラーのエラー状態を返す
Error	ユーザー定義エラーを発生
EResume	エラー処理ルーチンの終了後に、プログラムの実行を再開
PauseOn	一時停止状態 (Pause 状態)を返す

Exit ループまたはファンクションを強制終了します。

プログラムの実行に関する命令

Xqt	ファンクション名で指定したタスクを実行
Pause	一時停止可能なすべてのタスクを一時停止
Cont	一時停止中のすべてのタスクを再実行
Halt	実行中のタスクを一時停止
Quit	タスクを終了
Resume	一時停止中のタスクを継続実行
MyTask	現在のプログラムのタスク番号を返す
TaskDone	タスクの終了を確認する関数
TaskState	タスクの現在の状態を取得する関数
TaskWait	指定のタスクの終了を待つ
Restart	実行中のタスクをすべて中断し、実行していた最後のプログラムグループを再実行
Recover	安全扉が開いた時点の位置へ復帰動作を実行し、状態を返す
RecoverPos	安全扉が開いた時点の位置を返す
StartMain	バックグラウンドタスクからメインファンクションを実行

擬似命令

#define	識別子を指定文字列に置換するための定義
#ifdef ... #endif	条件付コンパイル
#ifndef ... #endif	条件付コンパイル
#include	指定したインクルードファイルの挿入
#undef	#define ステートメントで定義された識別子をクリア

ファイル管理に関する命令

ChDir	指定ドライブのカレントディレクトリの変更
ChDisk	ファイル操作する対象ディスクの設定
MkDir	サブディレクトリの作成
RmDir	サブディレクトリの削除
RenDir	ディレクトリ名の変更
FileDateTime\$	ファイルの日付と時刻を返す
FileExists	ファイルが存在するかどうかのチェック
FileLen	ファイルサイズを返します
FolderExists	フォルダーが存在するかどうかのチェック
FreeFile	現在使われていないファイル番号を返し、予約する
Del	ファイルの削除
Copy	ファイルのコピー
Rename	ファイル名の変更

AOpen	アペンド (付加)のためにファイルを開く
BOpen	ファイルをバイナリーアクセスモードで開く
ROpen	読み込むファイルを開く
Uopen	読み書きするファイルを開く
WOpen	書き込むファイルを開く
Input #	ファイル, 通信ポート, データベース, デバイスから、文字列か数値をデータ入力し、変数に保存
Print #	指定されたファイル, 通信ポート, データベース, デバイスにデータを出力
Line Input #	ファイル, 通信ポート, データベース, デバイスから、1 行データを読み込む
Read	ファイル、または通信ポートから指定した文字数を読み込む
ReadBin	ファイル、または通信ポートからバイナリーデータを読む
Write	ファイル、または通信ポートに文字列を書き込む、行末ターミネーターは付加しない
WriteBin	ファイル、または通信ポートへバイナリーデータを書き出す
Seek	カレントファイルポインターを設定
Close	ファイルを閉じる
Eof	ファイルの EOF (オープンしたファイルのポインターが終端にある)を返す
ChDrive	カレントドライブの変更
CurDir\$	現在のカレントディレクトリーを文字列で返す
CurDrive\$	現在のドライブ名を文字列で返す
CurDisk\$	現在のディスク名を文字列で返す
Flush	バッファをファイルへ書き込む

フィールドバスに関する命令

FbusIO_GetBusStatus	指定したフィールドバスの状態を返す
FbusIO_GetDeviceStatus	指定したフィールドバスデバイスの状態を返す
FbusIO_SendMsg	フィールドバス I/O デバイスに、メッセージを送り、回答を返す

数値に関する命令

Ctr	カウンターのカウント値を返す関数
CTReset	カウンターの定義とリセット
ElapsedTime	タクトタイム計測関数
ResetElapsedTime	タクトタイム計測用タイマーのリセット, スタート
Tmr	タイマー関数
TmReset	タイマーのリセット, スタート
Sin	指定角度のサインを返す関数
Cos	指定角度のコサインを返す関数
Tan	指定角度のタンジェントを返す関数
Acos	指定数値のアークコサインを返す関数
Asin	指定数値のアークサインを返す関数
Atan	指定数値のアークタンジェントを返す関数

Atan2	座標原点と指定点を結ぶ直角の角度を返す関数
Sqr	平方根を返す関数
Abs	絶対値を返す関数
Sgn	数値の符号を返す関数
Int	数値の小数点以下を切り捨てた値を返す関数
BClr	指定した数値の 1 ビットをクリアし、その値を返す
BSet	指定した数値の 1 ビットをセットし、その結果を返す
BTst	数値の 1 ビット状態を返す
BClr64	指定した数値の 1 ビットをクリアし、その値を返す
BSet64	指定した数値の 1 ビットをセットし、その結果を返す
BTst64	数値の 1 ビット状態を返す
Fix	実数値から整数部分を取り出す
Hex	16 進数表現の数値を文字化して返す
Randomize	乱数系列の初期化
Redim	ランタイム時に配列の最大要素番号の変更
Rnd	乱数を返す
UBound	指定した配列に設定できる要素番号の最大値を返す
文字列に関する命令	
Asc	文字列の先頭文字のキャラクターコードを 10 進数で返す
Chr\$	キャラクターコードに関する文字を返す
Left\$	文字列の左から、指定数の文字列を取り出す
Mid\$	指定した文字列から、開始位置より、指定した文字数の文字を取り出す
Right\$	文字列の末尾から、指定数の文字列を取り出す
Len	文字列の長さ (文字数) を返す
LSet\$	指定された文字列の最後にスペースを加え、指定の長さにした文字列を返す
RSet\$	文字列の先頭にスペースを追加し、指定の長さにした文字列を返す
Space\$	指定した数のスペース文字列を返す
Str\$	数値を文字列に変換
Val	数字からなる文字列を数値に変換
LCase\$	指定した文字列を小文字で返す
UCase\$	指定した文字列を大文字で返す
LTrim\$	文字列の先頭からのスペースを削除して返す
RTrim\$	文字列の最後からのスペースを削除して返す
Trim\$	文字列の先頭と最後からスペースを削除して返す
ParseStr	文字列をトークンの配列に構文解析
FmtStr	数字、あるいは、日付、時間の表記をフォーマット
FmtStr\$	数字、あるいは、日付、時間の表記をフォーマット

InStr	文字列からの文字列検索をし、見つかった場所を返す
Tab\$	指定した数だけのタブ文字列を返す

論理演算に関する命令

And	論理積演算
Or	論理和演算
LShift	数値データの左論理シフト
LShift64	数値データの左論理シフト
Mod	整数の剰余演算
Not	否定演算
RShift	数値データの右論理シフト
RShift64	数値データの右論理シフト
Xor	排他的論理和演算
Mask	Wait ステートメントでビットワイズ AND 操作を実行

変数に関する命令

Boolean	Boolean 型変数を宣言
Byte	Byte 型変数を宣言
Double	Double 型変数を宣言
Global	グローバル変数を宣言
Int32	4 バイト整数型変数を宣言
Integer	2 バイト整数型変数を宣言
Long	Long 型変数を宣言
Int64	8 バイト整数型変数を宣言
Real	実数型変数を宣言
Short	2 バイト整数型変数を宣言
String	文字列型変数を宣言
UByte	符号なし整数型変数を宣言
UInt32	符号なし 4 バイト整数型変数を宣言
UShort	符号なし 2 バイト整数型変数を宣言
UInt64	符号なし 8 バイト整数型変数を宣言

セキュリティーに関する命令

GetCurrentUser\$	カレントの EPSON RC+ ユーザーのログイン ID を返す
Login	実行時、他のユーザーとして EPSON RC+ にログイン

コンベヤートラッキングに関する命令

Cnv_AbortTrack	コンベヤークューポイントへの動作コマンドを中断
Cnv_Accel 関数	コンベヤ追従までの加速度、減速度設定値を返す
Cnv_Accel	コンベヤ追従までの加速度、減速度設定値を設定
Cnv_AccelLim	コンベヤ追従後の加速度、減速度の制限値を設定
Cnv_AccelLim 関数	コンベヤ追従後の加速度、減速度の制限値を返す
Cnv_Adjust	コンベヤに対する追従遅れ補正値を取得する動作を実施するかを設定
Cnv_AdjustClear	コンベヤに対する追従遅れ補正値を消去
Cnv_AdjustGet 関数	コンベヤに対する追従遅れ補正値を返す
Cnv_AdjustSet	コンベヤに対する追従遅れ補正値を設定
Cnv_Downstream 関数	コンベヤの下流限設定値を返す
Cnv_Downstream	コンベヤの下流限設定値を設定
Cnv_Fine 関数	指定コンベヤの追従完了判断範囲の設定を返す
Cnv_Fine	指定コンベヤに対し、コンベヤの追従完了判断範囲の設定、表示
Cnv_Flag 関数	追従中止線に対する追従状態を返す
Cnv_Mode 関数	コンベヤの設定モード設定値を返す
Cnv_Mode	コンベヤの設定モード設定値を設定
Cnv_Name\$関数	指定したコンベヤの名称を返す
Cnv_Number 関数	指定したコンベヤ名称のコンベヤ番号を返す
Cnv_OffsetAngle	コンベヤキューデータのオフセット値を設定
Cnv_OffsetAngle 関数	コンベヤキューデータのオフセット値を返す
Cnv_Point 関数	センサー座標値をコンベヤ座標値に変換して返す
Cnv_PosErr 関数	現在のトラッキング位置とターゲットとの位置偏差を返す
Cnv_PosErrOffset	現在のトラッキング位置とターゲットとの位置偏差を補正する値を設定
Cnv_Pulse 関数	コンベヤの現在位置のパルスを返す
Cnv_QueueAdd	コンベヤキューデータにポイントデータを追加
Cnv_QueueGet 関数	指定したコンベヤキューデータからポイントデータを返す
Cnv_QueueLen 関数	指定したコンベヤキューのデータ数を返す
Cnv_QueueList	指定したコンベヤキューデータの一覧を表示
Cnv_QueueMove	上流コンベヤのキューデータを下流コンベヤのキューに移動
Cnv_QueueReject	コンベヤの2重登録防止のための最小距離を設定、表示
Cnv_QueueReject 関数	コンベヤのキューの2重登録防止距離を返す
Cnv_QueueRemove	コンベヤキューデータからキューデータを削除
Cnv_QueueUserData	キューエントリーに関連したユーザーデータの表示、設定
Cnv_QueueUserData 関数	キューエントリーに関連したユーザーデータを返す
Cnv_RobotConveyor 関数	追従中のコンベヤ番号を返す
Cnv_Speed 関数	コンベヤの動作速度を返す
Cnv_Trigger	次の Cnv_QueueAdd ステートメントのために、コンベヤの現在位置をラッチ
Cnv_Upstream 関数	コンベヤの上流限設定値を返す
Cnv_Upstream	コンベヤの上流限設定値を設定

フォースセンシングに関する命令

Force_Calibrate	現在のフォースセンサーの全軸に対してゼロオフセットを設定
Force_ClearTrigger	現在のフォースセンサーのトリガー条件をすべてクリア
Force_GetForces	すべてのフォースセンサー軸のフォース、トルクを配列で返す
Force_GetForce 関数	指定する軸のフォースを返す
Force_Sensor	現在のタスクについて、使用するフォースセンサーを設定
Force_Sensor 関数	現在のタスクについて、使用するフォースセンサーを返す
Force_SetTrigger	Till コマンドのためのフォーストリガーを設定

DB に関する命令

CloseDB	OpenDB コマンドでオープンしたデータベースをクローズし、データベース番号を解放
DeleteDB	オープンしているデータベース内のテーブルからデータを削除する
OpenDB	データベース、Excel ブックをオープン
SelectDB	オープンしているデータベース内のテーブルからデータを検索する 関数
UpdateDB	オープンしているデータベース内のテーブルのデータを更新する

PG に関する命令

PG_FastStop	連続回転中のパルス出力軸を急停止する
PG_LSpeed	パルス出力軸の加速開始時のパルス速度および減速終了時のパルス速度を設定
PG_Scan	パルス出力軸の連続回転動作を開始
PG_SlowStop	連続回転中のパルス出力軸を減速停止

衝突検出に関する命令

CollisionDetect	衝突検出機能の有効/無効を設定
CollisionDetect 関数	CollisionDetect 命令の設定値を返す関数

部品消耗管理に関する命令

HealthCalcPeriod	部品消耗管理の演算期間を設定する
HealthCalcPeriod 関数	部品消耗管理の演算期間を返す関数
HealthCtrlAlarmOn 関数	コントローラーの部品の部品消耗アラーム状態を返す関数
HealthCtrlInfo	コントローラーの部品の交換推奨時期までの残り月数を表示する
HealthCtrlInfo 関数	コントローラーの部品の交換推奨時期までの残り月数を返す関数
HealthCtrlRateOffset	コントローラーの部品の部品消耗率のオフセット値を設定する
HealthCtrlReset	コントローラーの部品の部品消耗率を初期化する
HealthCtrlWarningEnable	コントローラーの部品の部品消耗アラーム通知の有効無効を設定する
HealthCtrlWarningEnable 関数	コントローラーの部品の部品消耗アラーム通知の有効無効状態を返す関数
HealthRateCtrlInfo 関数	コントローラーの部品の部品消耗率を返す関数
HealthRateRBInfo 関数	ロボットの部品の部品消耗率を返す関数
HealthRBAAlarmOn 関数	ロボットの部品の部品消耗アラーム状態を返す関数
HealthRBAAnalysis	ロボットの部品の部品消耗に関する分析結果 (交換推奨時期までの残り月数)を表示する
HealthRBAAnalysis 関数	ロボットの部品の部品消耗に関する分析結果 (交換推奨時期までの残り月数)を返す関数
HealthRBDistance	指定した関節の駆動量を表示する
HealthRBDistance 関数	指定した関節の駆動量を返す関数
HealthRBInfo	ロボットの部品の交換推奨時期までの残り月数を表示する
HealthRBInfo 関数	ロボットの部品の交換推奨時期までの残り月数を返す関数
HealthRBRateOffset	ロボットの部品の部品消耗率のオフセット値を設定する
HealthRBReset	ロボットの部品の部品消耗率を初期化する
HealthRBSpeed	指定した関節の平均速度を表示する
HealthRBSpeed 関数	指定した関節の平均速度を返す関数
HealthRBStart	ロボットの部品の部品消耗に関する分析を開始する
HealthRBStop	ロボットの部品の部品消耗に関する分析を終了する
HealthRBTRQ	指定した関節のトルク値を表示する
HealthRBTRQ 関数	指定した関節のトルク値を返す関数
HealthRBWarningEnable	ロボットの部品の部品消耗アラーム通知の有効無効を設定する
HealthRBWarningEnable 関数	ロボットの部品の部品消耗アラーム通知の有効無効状態を返す関数

シミュレーターに関する命令

SimSet	シミュレーターのオブジェクトの設定, 操作、ロボット動作を設定する
SimGet	シミュレーターのオブジェクトの設定値を取得する

ハンドに関する命令 (詳細は、Hand 機能マニュアルを参照してください。)	
Hand_On	グリッパー：ハンドの把持動作を実行 電動ドライバー：ハンドのねじ締め動作を実行
Hand_On 関数	グリッパー：ハンドが把持状態の場合、“True”を返す関数 電動ドライバー：ハンドがねじ締め完了状態の場合、“True”を返す関数
Hand_Off	グリッパー：ハンドの解放動作を実行 電動ドライバー：ハンドのねじ緩め動作を実行
Hand_Off 関数	グリッパー：ハンドが解放状態の場合、“True”を返す関数 電動ドライバー：ハンドがねじ緩め完了状態の場合、“True”を返す関数
Hand_TW 関数	直前の Hand_On コマンド、Hand_Off コマンドがタイムアウトした場合、“True”を返す関数
Hand_Def 関数	ハンドが定義されている場合、“True”を返す関数
Hand_Type 関数	ハンドのタイプ番号を返す関数
Hand_Label\$ 関数	ハンドのラベルを返す関数
Hand_Number 関数	ハンド番号を返す関数
安全機能に関する命令 (詳細は、安全機能マニュアルを参照してください。)	
SF_GetParam 関数	安全機能パラメーターの情報を返す
SF_GetParam\$関数	安全機能パラメーターのテキスト情報を返す
SF_GetStatus 関数	安全機能の状態ビットを返す
SF_LimitSpeedS	SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能について速度調整値を設定、表示する
SF_LimitSpeedS 関数	SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能の速度調整値を返す
SF_LimitSpeedSEnable	SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能について On/Off の設定、設定状態を表示する
SF_LimitSpeedSEnable 関数	SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能の状態を返す
SF_PeakSpeedS	速度監視点のピーク速度値を表示する
SF_PeakSpeedS 関数	速度監視点のピーク速度値を返す
SF_PeakSpeedSClear	速度監視点のピーク速度値をクリアし、初期化する
SF_RealSpeedS	速度監視点の現在速度を表示する
SF_RealSpeedS 関数	速度監視点の現在速度を返す

SPEL+ ランゲージリファレンス

この章では、下記の項目から SPEL+ のコマンドを説明しています。

書式	書式は、そのコマンドを使うときのフォーマットを示します。コマンドによっては、複数の書式が“解説”欄との関連を示す参照番号とともに示されています。
パラメーター	そのコマンドのパラメーターについての説明です。
結果	そのコマンドが返す値についての説明です。
戻り値	その関数が返す値についての説明です。
解説	コマンドの機能について、詳細を説明します。
注意	そのコマンドについて、重要事項が説明されています。
参照	そのコマンドに関連するその他のコマンドのリストです。関連するコマンドを参照したいときは、目次からページを探すことができます。
使用例	そのコマンドを使った例が示されます。

演算子

SPEL+ 言語で使用する演算子を示します。

演算子	書式例	説明
+	A+B	加算
-	A-B	減算
*	A*B	乗算
/	A/B	除算
**	A**B	べき乗
=	A=B	A と B は等しい
>	A>B	A が B より大きい
<	A<B	A が B より小さい
>=	A>=B	A が B より大きい等しい
<=	A<=B	A が B より小さい等しい
<>	A<>B	A と B が等しくない
And	A And B	論理積
Mod	A Mod B	整数の剰余
Not	Not A	否定
Or	A Or B	論理和
Xor	A Xor B	排他的論理和

演算子の優先順位

演算子は、プログラム中で以下の順位で処理されます。

優先順位	演算子	書式例	説明
1	()	(A+B)	カッコ
2	**	A**B	べき乗
3	*	A*B	乗算
	/	A/B	除算
4	Mod	A Mod B	整数の剰余
5	+	A+B	加算
	-	A-B	減算
6	=	A=B	A と B は等しい
	<>	A<>B	A と B が等しくない
	<	A<B	A が B より小さい
	>	A>B	A が B より大きい
	<=	A<=B	A が B より小さい等しい
	>=	A>=B	A が B より大きい等しい
7	Not	Not A	否定
8	And	A And B	論理積
9	Or	A Or B	論理和
10	Xor	A Xor B	排他的論理和

!...! 並列処理

動作中に I/O などの入出力処理を並列して行います。

書式

動作命令 !並列処理ステートメント!

パラメーター

動作命令

次のいずれかの並列処理可能な命令を記述します。

: Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move,
BGo, BMove, TGo, TMove

並列処理ステートメント

動作中に並列処理可能な I/O ステートメントを記述します。(下表参照)

解説

動作開始と同時に、“!”マークではさまれたステートメントの実行を開始します。例えば、これにより I/O は、ロボットの動きが止まるまで実行を待つのではなく、まだアームが動作中に、実行できることとなります。アームが動作中に I/O に対して実行を促すステートメントが用意されています。(下表“Dn”参照)

使用できるすべての並列処理ステートメントは、下表にまとめてあります。表中のステートメントは、いずれも単独で使うこともできます。また、いくつかを組み合わせ、1つの動作命令中に複数の I/O ステートメントを実行することもできます。

Dn	次の並列処理ステートメントが実行される前に移動量の%を指定し、動作命令と同期をとるために使用します。“n”は、0 から 100 までの整数で指定される%値で、その動作のどこから並列処理ステートメントが開始するか、開始位置を指定します。つまり、動作が全移動量の n%に達したとき Dn 以降のステートメントが実行されます。 Jump コマンドと組み合わせて使う場合、移動量の%値には、第 3 関節 (上下軸)の垂直移動動作が含まれていません。Jump 動作のときは第 3 関節 (上下軸)を除いた平行移動動作に対してのみ同期がとられます。第 3 関節の垂直移動動作が完了したところでステートメントを実行するためには、ステートメントの最初に D0 (ゼロ)を入れてください。 Jump3 コマンドと組み合わせて使う場合、移動量の%値には、退避、接近動作が含まれていません。Jump3 動作のときは、スパン動作に対してのみ同期がとられます。退避動作が完了したところでステートメントを実行するためには、ステートメントの最初に D0 (ゼロ)を入れてください。 “Dn”は、1 回の並列処理ステートメント中、最高 16 回まで使うことができます。
On / Off n	出力ビット番号“n”をオン、またはオフします。
MemOn / MemOff n	メモリーI/O ビット番号“n”をオン、またはオフします。
Out p,d OpBCD p,q OutW p,d	出力ポート“p”にデータ“d”を出力します。
MemOut p, d MemOutW p,d	メモリーI/O ポート“p”にデータ“d”を出力します。
Signal s	シンクロ信号を發します。
WaitSig s	信号“s”を待ってから、次のステートメントを処理します。

Wait t	“ t ”秒間待ち、次の並列処理ステートメントを実行します。
Wait Sw(n) = j	入力ビット“ n ”が“ j ”で定義される条件 (On または Off) に一致するまで、次の並列処理ステートメントの実行を待ちます。
Wait MemSw(n) = j	メモリー I/O ビット“ n ”が“ j ”で定義される条件 (On または Off) に一致するまで、次の並列処理ステートメントの実行を待ちます。
Wait その他の条件	上記 2 パターン以外の Wait も可能です。詳細は Wait を参照してください。
Print	ディスプレイデバイスにデータを出力します。
Print #	指定されたコミュニケーションポートにデータを出力します。
外部ファンクション	Declare ステートメントで宣言した外部ファンクションを実行します。
Hand_On n Hand_Off n	ハンド番号“ n ”の Hand_On/Hand_Off 動作を実行します。

注意

I/O コマンドがすべて終了する前に動作が終わった場合

特定の動作命令に対する動作が終了しても、すべての並列処理ステートメントの実行が終わっていない場合、次のプログラムは、それらがすべて終了してから実行されます。このような状況は、多くの I/O コマンドを並列処理しなくてはならない短距離移動の動作時に、特に想定されるものです。

アームを停止する Till ステートメントで、途中で動作が終了した場合

もし、移動の途中でアームを停止する Till ステートメントが使われた場合、動作は 100% 完了したとみなされ D100 までが実行されます。すべての並列処理ステートメントの実行が終わるまで、動作命令の次のステートメントには移りません。

AbortMotion ステートメントや Trap により、途中で動作が終了した場合

動作が終了した以降の D ステートメントは実行されません。

100% に近い“ n ”値を設定した場合のパスモーションの減速

パスモーション中に高い“ n ”値を設定した場合、ロボットは実行中の動きを終わらせるために減速することがあります。通常 CP On は、減速開始と同時に次の動作命令の加速を始めます。しかし、減速中に D n の指定があると、その命令を終了するまでは、次の加速を始めることができません。減速を避けるためには、処理ステートメントを動作コマンドのあとにもってきます。例えば、下記の例では、On 1 ステートメントの位置を、Jump P1 動作中の並列処理から、Jump P1 の後に移しています。

```
CP On
Jump P1 !D96; On 1!
Go P2
```

```
CP On
Jump P1
On 1
Go P2
```

Jump ステートメントと並列処理

Jump ステートメントとともに使われる並列処理ステートメントの実行は、上昇移動の終了後に始まり、下降移動の開始前に終わります。

Jump3 ステートメントとともに使われる並列処理ステートメントの実行は、退避移動の終了後に始まり、接近移動の開始前に終わります。

Here ステートメントと並列処理

Here ステートメントと並列処理を 1 つの動作命令内で混在させることはできません。

```
Go Here :Z(0) ! D10; MemOn 1 !
```

上記のような使い方はできません。

```
P999 = Here
```

```
Go P999 Here :Z(0) ! D10; MemOn 1 !
```

上記のようなプログラムに変更してください。

参照

Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move, BGo, BMove, TGo, TMove

!...! 並列処理使用例

下記は、並列処理機能を動作命令とともに使う例です。

並列処理を Jump コマンドとともに使います。第 3 関節が上昇移動を終わり、第 1, 2, 4 関節が動きだした段階で、出力ビット 1 がオンします。出力ビット 1 は、Jump 動作の 50%が完了した段階で、再びオフします。

```
Function test
  Jump P1 !D0; On 1; D50; Off 1!
Fend
```

並列処理を Move コマンドとともに使います。P1 までの移動の 10%を完了した段階で、出力ビット 5 をオンし、0.5 秒後に出力ビット 5 をオフします。

```
Function test2
  Move P1 !D10; On 5; Wait 0.5; Off 5!
Fend
```


#define

識別子を指定文字列に置換するために定義します。

書式

#define 識別子 [(パラメーター [, パラメーター])] 置換文字列

パラメーター

識別子 文字列パラメーターの省略のキーワードです。識別子の規則は次のとおりです。

- 英字で始まり、英数字とアンダースコア(_)を使用して作成します。
- 識別子には、スペースやタブは使えません。

パラメーター 置換文字列に使われる、1 つ、または複数の変数を指定し、マクロのようにダイナミックな定義メカニズムを提供します。**#define** コマンドでは、8 個までのパラメーターが使えます。それぞれのパラメーターをカンマで区切り、パラメーターリストを括弧の中に入れてください。

置換文字列 プログラムがコンパイルされたときに識別子と置き換えられる置換文字列です。置換文字列に関する規則は次のとおりです。

- 置換文字列では、スペースやタブが使えます。
- ステートメントとともに使われる識別子は置換文字列として使えません。
- コメント記号“**/***”を使用すると、その後に続く文字はコメントとして判断され、置換文字列には含まれません。
- 置換文字列は省略できます。その場合、指定した識別子は、何もないもの、またはヌル文字列に置換されます。これにより、その識別子はプログラムから削除されます。

解説

#define 命令により、指定された識別子はプログラム内で置換されます。指定した識別子が見つかるたびに、置換文字列に置き換えられてコンパイルします。しかし、ソースコード自体は、置換文字列ではなく、識別子を含んだ形で残されます。それは、コード自体を読むとき、識別子のままの方が、文字列に置換されてしまったコードを読むよりはるかに読みやすいからです。

定義された識別子は、**#ifdef** や **#ifndef** コマンドと組み合わせて、コンパイルするかしないかの条件として使用できます。

パラメーターが指定されれば、識別子はマクロとして使うことができます。

注意

#define を変数宣言やラベル置換に使うと、エラーになります

変数宣言のために **#define** 命令を使うと、エラーになりますので注意してください。

参照

#ifdef, **#ifndef**

#define 使用例

'デバッグモードの場合は次の行を非コメント化

```
' #define DEBUG
```

```
Input #1, A$  
#ifdef DEBUG  
    Print "A$ = ", A$  
#endif  
Print "The End"
```

```
#define SHOWVAL(x) Print "var = ", x
```

```
Integer a
```

```
a = 25
```

```
SHOWVAL(a)
```

#ifdef...#else...#endif

条件付コンパイルを行います。

書式

```
#ifdef 識別子
...条件付コンパイルのためのソースコード
[#else
...偽の条件のためのソースコード]
#endif
```

パラメーター

識別子 #ifdef と #else または #endif の間のソースコードをコンパイルする、ユーザー定義のキーワードです。これにより、この識別子はコンパイルの条件として機能します。

解説

#ifdef...#else...#endif は、選択されたソースコードを条件付コンパイルします。コンパイルをするかしないかの条件は、識別子によって決定されます。最初に、#ifdef は、指定された識別子が現在#define によって定義されているかチェックします。#else ステートメントは省略可能です。

<定義されている場合> #else ステートメントが使われていれば、#ifdef と #else の間のステートメントがコンパイルされます。#else ステートメントが使われていなければ、#ifdef と #endif 間のステートメントがコンパイルされます。

<定義されていない場合> #else ステートメントが使われていれば #else と #endif の間のステートメントがコンパイルされます。#else ステートメントが使われていなければ、#ifdef と #endif の間のステートメントはコンパイルされず、スキップします。

参照

#define, #ifndef

#ifdef 使用例

下記は#ifdefを使ったプログラムの一例です。下記の例では、#define DEBUG 擬似命令の定義の有無によって、変数 A\$ の値を出力するかしないか制御できます。もし、#define DEBUG 擬似命令が、ソース中、ここで例として記載している部分より以前に使われていれば、Print A\$ の行はコンパイルされ、プログラムの実行時に出力されます。しかし、文字列“The End”は、#define DEBUG 擬似命令にかかわらず出力されます。

```
'デバッグモードの場合は次の行を非コメント化
' #define DEBUG

Input #1, A$
#ifdef DEBUG
    Print "A$ = ", A$
#endif
Print "The End"
```

#ifndef...#endif

条件付コンパイルを行います。

書式

```
#ifndef 識別子
.. 条件付コンパイルのためのソースコード
[#else
... 真の条件のためのソースコード]
#endif
```

パラメーター

識別子 ユーザー定義のキーワードです。定義されていないときには、**#ifndef**から**#else**、または**#endif**までのソースコードがコンパイルされます。この識別子はコンパイルを行うための条件として機能します。

解説

この命令は、「もし定義されていなければ」命令とも呼ばれ、**#ifdef**とは反対の条件をチェックします。**#ifndef...#else...#endif**は、選択したソースコードの条件付コンパイルを行います。**#else**ステートメントは省略可能です。

<定義されている場合>

#elseステートメントが使われていれば、**#else**と**#endif**の間のステートメントがコンパイルされます。**#else**ステートメントが使われていなければ、**#ifndef**と**#endif**の間のステートメントはコンパイルされません。

<定義されていない場合>

#elseステートメントが使われていれば、**#else**と**#endif**の間のステートメントは、コンパイルされません。**#else**ステートメントが使われていなければ、**#ifndef**と**#endif**の間のステートメントはコンパイルされます。

注意

#ifdef と**#ifndef** の違い

#ifdefは、識別子が定義されているとき、指定されたソースコードをコンパイルします。一方、**#ifndef**は、識別子が定義されていないとき、指定されたソースコードをコンパイルします。

参照

#define, **#ifdef**

#ifndef 使用例

下記は**#ifndef**を使ったプログラムの一例です。下記の例では、**#define NODELAY** 擬似命令の定義の有無によって、変数 **A\$**の値を出力するかしないか制御できます。もし、**#define NODELAY** 擬似命令が、ソース中、ここで例として記載している部分より以前に使われていれば、コンパイルのとき、**Wait 1**の行は、このプログラムのソースのほかの部分と一緒にコンパイルされません。もし、**#define NODELAY** 擬似命令がここで例として記載している部分より以前に使われていなければ、(つまり **NODELAY** は定義されていなければ)**Wait 1**の行は、コンパイルされて、後にプログラムで実行されます。文字列“The End”は、**#define NODELAY** 擬似命令にかかわらず、出力されます。

```
'遅延する場合は次の行をコメント化
#define NODELAY 1

Input #1, A$
#ifndef NODELAY
    Wait 1
#endif
Print "The End"
```

#include

#include ステートメントが使われた場所に、指定したインクルードファイルを挿入します。

書式

```
#include "インクルードファイル名.INC "
```

パラメーター

インクルードファイル名	インクルードファイル名は、カレントプロジェクトで有効となっているインクルードファイル名を指定してください。拡張子は、すべて“.inc”です。インクルードファイル名は、現在のファイルに挿入されるファイル名を指定します。

解説

#include は、指定したインクルードファイルの内容を、#include ステートメントが使われた場所に挿入します。

インクルードファイルには、#define ステートメントとグローバル変数の宣言を入れてください。

#include ステートメントは、ファンクション定義の外で使用してください。

インクルードファイルの中に他のインクルードファイルを記述することができます。例えば、FILE1 の中で、FILE2 をインクルードし、FILE2 の中で FILE3 をインクルードすることができます。これをファイルのネスティングといいます。

参照

#define, #ifdef, #ifndef

#include 使用例

Include File (Defs.inc)

```
#define DEBUG 1
#define MAX_PART_COUNT 20
```

Program File (main.prg)

```
#include "defs.inc"

Function main
  Integer i

  Integer Parts (MAX_PART_COUNT)

Fend
```

#undef

#define ステートメントで定義された識別子を未定義にします。

書式

#undef 識別子名

パラメーター

識別子名 #define ステートメントで使⽤したキーワードを指定します。

参照

#define, #ifdef, #ifndef

AbortMotion

動作命令を中断し動作を実行していたタスクをエラーにします。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

AbortMotion { ロボット番号 | All }

パラメーター

ロボット番号	動作を中断するロボット番号を指定します。
All	すべてのロボットの動作を中断します。

解説

AbortMotion を実行したときのロボットの状態により、次のようになります。
いずれの場合も処理を継続するには、OnErr でエラーをフックして、エラー処理を記述する必要があります。

エラー2999 は、定数 ERROR_DOINGMOTION を使用することができます。

エラー2998 は、定数 ERROR_NOMOTION を使用することができます。

継続実行 (Cont)を実行する前に、2 回以上 AbortMotion を実行しないようにプログラムを作成してください。

ロボットが動作命令を実行中の場合

ロボットはアーム動作を直ちに一時停止 (クイックポーズ)し、残りの動作を破棄します。
ロボットに対して動作命令を実行していたタスクでエラー2999 (ERROR_DOINGMOTION)が発生します。
次の動作命令は、一時停止した位置から直接次の目標位置へ動作します。

ロボットが一時停止 (クイックポーズ)中の場合

AbortMotion を実行した時点では何も発生しませんが、内部的には残りの動作が破棄されます。
継続実行 (Cont)を指示した時点で、ロボットに対して動作命令を実行していたタスクで、エラー2999 (ERROR_DOINGMOTION)が発生します。
次の動作命令は、一時停止していた位置から直接次の目標位置へ動作します。

ロボットが WaitRecover 状態 (安全扉開状態)の場合

AbortMotion を実行した時点では何も発生しませんが、内部的には残りの動作が破棄されます。
その先の動作は Recover コマンドのフラグにより選択可能です。

“Recover ロボット番号, WithMove” を実行するとロボットの励磁が回復し、復帰動作を実行します。
継続実行 (Cont)を指示した時点で、ロボットに対して動作命令を実行していたタスクでエラー2999 (ERROR_DOINGMOTION)が発生します。
次の動作命令は、復帰動作が完了した位置から直接次の目標位置へ動作します。

“Recover ロボット番号, WithoutMove” を実行するとロボットの励磁が回復します。
継続実行 (Cont)を指示した時点で、ロボットに対して動作命令を実行していたタスクでエラー2999 (ERROR_DOINGMOTION)が発生します。
次の動作命令は、励磁が回復した位置から直接次の目標位置へ動作します。(復帰動作がありません。)

ロボットが動作命令以外の命令を実行中の場合

直前にロボットに対して動作命令を実行していたタスクでエラー2998 (ERROR_NOMOTION)が発生します。タスクが Wait や Input コマンドによる待機中の場合、即座に中断しエラー2998 になります。CP On の設定で動作命令を実行し、動作命令からプログラムが抜けている場合は、ロボットが動作中でもエラー2998 が発生します。

指定したロボットが、プログラム (タスク)から動作していない場合

エラーとなります。

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

参照

OnErr, Recover, Till

AbortMotion 使用例

メモリーI/O の 0 番が On すると AbortMotion が実行され、Home 位置へロボットが移動します。

```
Function main
  Motor On
  Xqt sub, NoEmgAbort
  OnErr GoTo errhandle

  Go P0
  Wait Sw(1)
  Go P1

  Quit sub
  Exit Function

errstart:
  Home
  Quit sub
  Exit Function

errhandle:
  Print Err
  If Err = ERROR_DOINGMOTION Then
    Print "ロボット動作中"           `Go P0 や Go P1 を実行中
    EResume errstart
  ElseIf Err = ERROR_NOMOTION Then
    Print "ロボット非動作中"         `Wait Sw(1) を実行中
    EResume errstart
  EndIf

  Print "Error Stop"                 `その他のエラー発生
  Quit All
Fend

Function sub
  MemOff 0
  Wait MemSw(0)
  AbortMotion 1
  MemOff 0
Fend
```


Abs 関数

絶対値を返す関数です。

書式

Abs(数値)

パラメーター

数値 式または直接数値で指定します。

戻り値

指定した数値の絶対値を返します。

解説

絶対値とは符号なしの数値のことです。例として、Abs (-1)と Abs (1)では両方とも 1 を返します。

参照

Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str\$, Tan, Val

Abs 関数使用例

次の例はコマンドウィンドウから Print 命令を使って実行します。

```
> print abs (1)
1
> print abs (-1)
1
> print abs (-3.54)
3.54
>
```

Accel

Go, Jump, Pulse などの PTP 動作の加減速度の設定と表示をします。

書式

(1) Accel 加速設定値, 減速設定値 [, 退避加速設定値, 退避減速設定値, 接近加速設定値, 接近減速設定値]

パラメーター

加速設定値	最大加速度に対する割合を 1 以上の整数で指定します。(単位: %)
減速設定値	最大減速度に対する割合を 1 以上の整数で指定します。(単位: %)
退避加速設定値	Jump 時における退避加速度を 1 以上の整数で指定します。 省略可能です。Jump コマンド時のみ設定可能です。
退避減速設定値	Jump 時における退避減速度を 1 以上の整数で指定します。 省略可能です。Jump コマンド時のみ設定可能です。
接近加速設定値	Jump 時における接近加速度を 1 以上の整数で指定します。 省略可能です。Jump コマンド時のみ設定可能です。
接近減速設定値	Jump 時における接近減速度を 1 以上の整数で指定します。 省略可能です。Jump コマンド時のみ設定可能です。

結果

パラメーターが省略されたときは、現在の Accel パラメーターが返されます。

解説

Accel はあらゆる PTP 動作 (Go, Jump, Pulse などの命令によって起こる動作) の加減速度を設定します。

Accel が設定する加減速度パラメーターは、1 以上の整数値で設定します。この数値は、最大加速度 (または減速度) に対する割合を示します。通常 100 が最大値となりますが、100 を越える設定が可能なロボットがあります。AccelMax 関数は Accel 設定できる最大値を返します。

Accel は、新たに加減速度を設定するときや、単に現在の設定値を出力するときなどに使います。新たに加速度や減速度を設定して Accel を使う場合、最初の 2 つのパラメーター (加速設定値と減速設定値) が必要です。

退避加速設定値, 退避減速設定値, 接近加速設定値, 接近減速設定値の 4 パラメーターは、Jump 命令のみ有効で、省略可能です。これらのパラメーターは、Jump 動作開始時の退避動作と Jump 動作終了時の接近動作の、それぞれの加速設定値と減速設定値を指定します。

下記のいずれかの場合、Accel 値は、初期化されます。

コントローラー電源 オン Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

注意

ローパワーモード (Power Low)で Accel コマンドを実行するとき

ローパワーモード (Power Low) 時に Accel を実行するときは、新しい値は保存されますが、現在値は低く制限されます。TEACH モードが OFF で、パワーが High に設定されているとき、Accel は有効になります。

Accel と AccelS の違い

Accel 命令は、直線や円弧動作の加減速度を設定するものではありません。一方、AccelS 命令は、直線や円弧動作について加減速度を設定します。

100 を越える Accel 設定

Accel 設定は通常 100 が最大値となりますが、100 以上の値が設定可能な機種があります。通常の使用においては Accel 設定値 100 が加減速度と位置決め時の振動とのバランスをとった最適な設定になっています。しかし、動作条件によっては、位置決め時の振動を小さくすることより、加減速度を上げてサイクルタイムの向上を優先したい場合があります。そのような場合には、Accel 設定値に 100 以上の値を設定してください。ただし、動作条件によっては 100 以上の値を設定した場合でもサイクルタイムに変化がみられないことがあります。

参照

AccelR, AccelS, Go, Jump, Jump3, Power, Pulse, Speed, TGo

Accel 使用例

下記は、Accel と Speed を使った簡単な動作プログラム例です。Accel と Speed には、あらかじめ定義された変数が使われています。

<例 1>

```
Function acctest
  Integer slow, accslow, decslow, fast, accfast, decfast

  slow = 20      '低速度の設定
  fast = 100     '高速度の設定
  accslow = 20   '低加速度の設定
  decslow = 20   '低減速度の設定
  accfast = 100  '高加速度の設定
  decfast = 100  '高減速度の設定

  Accel accslow, decslow
  Speed slow
  Jump pick
  On gripper
  Accel accfast, decfast
  Speed fast
  Jump place
  .
  .
  .
End
```

<例 2>

Jump 命令時、パーツをていねいにあつかうように第 3 関節の下降減速度を遅くする例です。このような例では、Accel 設定時に第 3 関節下降減速設定値パラメーターを低く設定する必要があります。

```
>Accel 100,100,100,100,100,35

>Accel
    100      100
    100      100
    100      35
>
```

Accel 関数

現在の加減速度の設定値を返す関数です。

書式

Accel(設定値番号)

パラメーター

設定値番号	整数値で下記のそれぞれの値を指定します。
	1: 加速設定値
	2: 減速設定値
	3: Jump 動作時の退避加速設定値
	4: Jump 動作時の退避減速設定値
	5: Jump 動作時の接近加速設定値
	6: Jump 動作時の接近減速設定値

戻り値

1 以上の整数 (%)を返します。

参照

Accel

Accel 関数使用例

プログラム中に Accel 関数を使う例です。

```
Integer currAccel, currDecel

' 現在の加減速度を取得
currAccel = Accel (1)
currDecel = Accel (2)
Accel 50, 50
SRVJump pick
' 前の設定をリストア
Accel currAccel, currDecel
```

AccelMax 関数

Accel で設定可能な加減速度の最大値を返す関数です。

書式

AccelMax(最大値番号)

パラメーター

最大値番号 整数値で下記のそれぞれの値を指定します。

- 1: 加速度最大値
- 2: 減速度最大値
- 3: Jump 動作時の退避加速度最大値
- 4: Jump 動作時の退避減速度最大値
- 5: Jump 動作時の接近加速度最大値
- 6: Jump 動作時の接近減速度最大値

戻り値

1 以上の整数 (%)を返します。

参照

Accel

AccelMax 関数使用例

プログラム中に AccelMax 関数を使う例です。

```
' 最大の加減速度を表示  
Print AccelMax(1), AccelMax(2)
```

AccelR

CP 動作のツール姿勢変化について、加減速度の設定と表示をします。

書式

- (1) AccelR 加速設定値 [, 減速設定値]
- (2) AccelR

パラメーター

- 加速設定値 加速設定値を実数値 (0.1~5000)で指定します。(単位: deg/sec²)
- 減速設定値 減速設定値を実数値 (0.1~5000)で指定します。(単位: deg/sec²)

パラメーターの有効設定値

	加速設定値/減速設定値
VT シリーズ	0.1~1000
C シリーズ, N シリーズ T シリーズ, G シリーズ, GX シリーズ RS シリーズ, LS-B シリーズ	0.1~5000

(単位: deg/sec²)

結果

パラメーターを省略すると、現在の AccelR 設定値を表示します。

解説

AccelR は、Move, Arc, Arc3, BMove, TMove, Jump3CP において、ROT 修飾パラメーターを使っている時のみ有効となります。

下記のいずれかの場合、AccelR 値は、初期化されます。

コントローラー電源 オン Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

参照

Arc, Arc3, BMove, Jump3CP, Power, SpeedR, TMove

AccelR 使用例

AccelR 360, 200

AccelR 関数

指定したツール姿勢変化の加減速度の設定値を返します。

書式

AccelR(設定値番号)

パラメーター

設定値番号 下記のいずれかの値を、式または数値で指定します。
1: 加速度設定値
2: 減速度設定値

戻り値

加速、または減速設定値を実数値 (単位: deg/sec²)で返します。

参照

AccelR

AccelR 関数使用例

```
Real currAccelR, currDecelR
```

```
' 現在の加減速度を取得
```

```
currAccelR = AccelR(1)
```

```
currDecelR = AccelR(2)
```

AccelS

ロボットの直線動作や CP 動作について加減速度の設定を行います。
(Move, Arc, Arc3, Jump3, CVMove を参照してください。)

書式

- (1) AccelS 加速設定値 [,減速設定値] [, 退避加速設定値, 退避減速設定値, 接近加速設定値, 接近減速設定値]
(2) AccelS

パラメーター

加速設定値	直線動作または CP 動作時の加速度を <u>実数値</u> で指定します。(単位: mm/sec ²) 減速設定値を省略すると、加速設定値が、加速時と減速時に適用されます。
減速設定値	直線動作または CP 動作時の減速度を <u>実数値</u> で指定します。(単位: mm/sec ²) 省略可能です。
退避加速設定値	Jump3 時と Jump3CP 時における退避動作の退避加速度を <u>実数値</u> で指定します。 (単位: mm/sec ²) 省略可能です。
退避減速設定値	Jump3 時と Jump3CP 時における退避動作の退避減速度を <u>実数値</u> で指定します。 (単位: mm/sec ²) 省略可能です。
接近加速設定値	Jump3 時と Jump3CP 時における接近動作の接近加速度を <u>実数値</u> で指定します。 (単位: mm/sec ²) 省略可能です。
接近減速設定値	Jump3 時と Jump3CP 時における接近動作の接近減速度を <u>実数値</u> で指定します。 (単位: mm/sec ²) 省略可能です。

パラメーターの有効設定値 (単位: mm/sec²)

	加速設定値/減速設定値 退避加速設定値/退避減速設定値 接近加速設定値/接近減速設定値
N2	0.1~5000
LS20-B, T シリーズ, VT シリーズ	0.1~10000
C4-*901**	0.1~15000
C4-*601**, C8-*1401**, G シリーズ, GX シリーズ, RS シリーズ, LS3-B, LS6-B, LS10-B C8-*701**W, C8-*901**W, N6, C12	0.1~25000
C8-*701**, C8-*701**R, C8-*901**, C8-*901**R	0.1~35000

結果

パラメーターを省略すると、加速値と減速値を表示します。

加速値と減速値を表示する場合、加速設定値、減速設定値、退避加速設定値、退避減速設定値、接近加速設定値、接近減速設定値それぞれに対して、現在設定されているハンド質量に応じて補正された加速値や減速値も表示されます。

解説

AccelS は、直線および円弧補間を含む、あらゆる円弧補間動作について加速と減速を設定します。Move や Arc 命令による動作も含みます。

下記のいずれかの場合、AccelS 値は、初期化され加減速度は遅くなります。

コントローラー電源 オン
Motor On 実行
SFree, SLock, Brake 実行
Reset, Reset Error 実行
停止ボタン, Quit All 実行などによるタスクの終了

注意

ローパワーモード (Power Low)で AccelS コマンドを実行するとき

ローパワーモード (Power Low)時に AccelS を実行するときは、新しい値は保存されますが、現在値は低く制限されます。

TEACH モードが OFF で、パワーが High に設定されているとき、AccelS は有効になります。

Accel と AccelS の違い

AccelS 命令は、PTP 動作 (Go や Jump 命令の動作)についての加減速度を設定するものではありません。一方、Accel 命令は、PTP 動作についての加減速度を設定します。

上限値

スカルロボット (RS シリーズ含む)の AccelS の上限値は、Weight 設定とスプラインユニットの位置で変化します。詳細は、マニピュレーターマニュアル (CP 動作時の ACCELS の設定)を参照してください。

垂直 6 軸型ロボットの AccelS の上限値は、Weight の設定で変化します。詳細は、マニピュレーターマニュアル (仕様表)を参照してください。

参照

Accel, Arc, Arc3, Jump3, Jump3CP, Power, Move, TMove, SpeedS

AccelS 使用例

下記は、あらかじめ定義された変数を使って、Move 命令の AccelS と SpeedS を設定する簡単な動作プログラム例です。

```
Function acctest
  Integer slow, accslow, fast, accfast

  slow = 20          '低速度の設定
  fast = 100         '高速度の設定
  accslow = 200      '低加速度の設定
  accfast = 5000     '高加速度の設定
  AccelS accslow
  SpeedS slow
  Move P1
  On 1
  AccelS accfast
  SpeedS fast
  Jump P2
  .
  .
  .
Fend
```

AccelS 関数

CP 動作の加減速度の設定値を返す関数です。

書式

AccelS(設定値番号)

パラメーター

設定値番号	下記のいずれかの値を、整数値または式で指定します。
1:	加速設定値
2:	減速設定値
3:	Jump3 時と Jump3CP 時の退避加速設定値
4:	Jump3 時と Jump3CP 時の退避減速設定値
5:	Jump3 時と Jump3CP 時の接近加速設定値
6:	Jump3 時と Jump3CP 時の接近減速設定値
7:	ハンド質量によって補正された加速値
8:	ハンド質量によって補正された減速値
9:	ハンド質量によって補正された Jump3 時と Jump3CP 時の退避加速値
10:	ハンド質量によって補正された Jump3 時と Jump3CP 時の退避減速値
11:	ハンド質量によって補正された Jump3 時と Jump3CP 時の接近加速値
12:	ハンド質量によって補正された Jump3 時と Jump3CP 時の接近減速値

戻り値

加速設定値、または減速設定値 (0~5000 の実数値、単位: mm/s²)を返します。

参照

AccelS, Arc3, SpeedS, Jump3, Jump3CP

AccelS 関数使用例

```
Real savAccelS
savAccelS = AccelS(1)
```

Acos 関数

指定数値のアーコサインを返します。

書式

Acos(数値)

パラメーター

数値 角度のコサインを実数値で指定します。

戻り値

指定した数値のアーコサイン値を実数値 (単位ラジアン) で返します。

解説

Acos は、指定された数値のアーコサインを返します。指定する数値は、 -1 ~ 1 の範囲です。戻り値の範囲は、 $0 \sim \pi$ です。数値が -1 より小さい、または 1 より大きい場合は、エラーが発生します。

ラジアン値を、度に変換するためには、RadToDeg 関数を使います。

参照

Abs, Asin, Atan, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Acos 関数使用例

```
Function acostest
    Double x

    x = Cos(DegToRad(30))
    Print "Acos of ", x, " is ", Acos(x)
End
```

Agl 関数

指定された回転関節の角度、または直動関節の位置を返す関数です。

書式

Agl(関節番号)

パラメーター

関節番号 関節番号を整数値で指定します。範囲は、1 からロボットの関節の数までです。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

指定された回転関節の角度、または直動関節の位置が返されます。

解説

Agl 関数は、指定された回転関節の角度、または直動関節の位置を得るために使われます。

指定された関節が回転関節の場合、指定された関節の角度 0 からの回転角度を“度”単位の実数値で返します。

指定された関節が直動関節の場合は、その関節の位置 0 からの移動量を“mm”単位の実数値で返します。

Arm ステートメントにより増設アームを設定し、その増設アームを選択している場合、Agl 関数はその増設アームの、標準アームのパルス 0 位置からの角度、または位置を返します。

参照

PAgl, PIs, PPIs

Agl 関数使用例

下記は、コマンドウィンドウから Print 命令を使う例です。

```
> print agl(1), agl(2)
17.234 85.355
```

AgIToPls 関数

ロボットの各関節の角度をパルスに変換します。

書式

AgIToPls(関節位置 1, 関節位置 2, 関節位置 3, 関節位置 4 [, 関節位置 5, 関節位置 6] [, 関節位置 7] [, 関節位置 8, 関節位置 9])

パラメーター

関節位置 1 - 6	関節の角度を実数値で指定します。
関節位置 7	第 7 関節の角度を実数値で指定します。 ジョイント型 7 軸ロボットのみを使用します。
関節位置 8	付加軸 S 関節の角度を実数値で指定します。
関節位置 9	付加軸 T 関節の角度を実数値で指定します。

戻り値

各関節位置から計算した関節パルスを返します。

解説

関節角度から関節パルスに変換するには AgIToPls 関数を使います。

注意

AgIToPls 関数で指定した関節位置で実現されるロボットの形態が特異姿勢である場合、AgIToPls の結果をポイント変数に代入すると、指定した関節位置の一部の情報が失われてしまいます。その結果、代入したポイント変数を用いて動作すると、AgIToPls で指定した関節位置とは異なる関節位置にロボットが動作します。下記の例では、P1 は関節位置 (0, 0, 0, 0, 90) に動作します。

```
P1 = AgIToPls (0, 0, 0, 90, 0, 0)
Go P1
```

同様に、AgIToPls を CP 動作命令の引数に直接指定したとき、指定した関節位置とは異なる関節位置に動作します。下記の例では、関節位置 (0, 0, 0, 0, 90) に動作します。

```
Move AgIToPls (0, 0, 0, 90, 0, 0)
```

PTP 動作命令の引数として AgIToPls 関数を用いた場合は、このような特異姿勢の問題はありません。

参照

AgI, JA, Pls

AgIToPls 関数使用例

```
Go AgIToPls (0, 0, 0, 90, 0, 0)
```

AIO_In 関数

オプションのアナログ I/O 入力チャンネルからアナログ値を読み込みます。

書式

AIO_In(チャンネル番号)

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。

戻り値

チャンネル番号で指定したアナログ I/O チャンネルのアナログ入力値を実数で返します。戻り値の範囲はアナログ IO 基板の入力レンジ設定によります。

解説

In 関数

参照

AIO_InW 関数, AIO_Out, AIO_OutW, AIO_Out 関数, AIO_OutW 関数, AIO_Set, Wait

AIO_In 関数使用例

```
Function main
  Real var1
  var1 = AIO_In(2)  'アナログ入力チャンネル 2 の入力状態を取得
  If var1 > 5.0 Then
    Go P1
    Go P2
    '他の動作コマンドをここで実行
    '
    '
  Else
    Print "Error in initialization!"
    Print "Sensory Inputs not ready for cycle start"
    Print "Please check analog inputs 2."
  EndIf
Fend
```

AIO_InW 関数

オプションのアナログ I/O 入力チャンネルからアナログ値を読み込みます。

書式

AIO_InW(チャンネル番号)

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。

戻り値

指定したアナログ I/O チャンネルの入力状態(0~65535 の Long 型整数)を返します。

各入力チャンネルの入力電圧(電流)と戻り値の対応は、アナログ I/O 基板の入力レンジ設定により下記の通りとなります。

入力データ		入力レンジ設定				
16 進数	10 進数	±10.24(V)	±5.12(V)	0-5.12(V)	0-10.24(V)	0-24(mA)
0xFFFF	65535	10.23969	5.11984	5.12000	10.24000	24.00000
0x8001	32769	0.00031	0.00016	2.56008	5.12016	12.00037
0x8000	32768	0.00000	0.00000	2.56000	5.12000	12.00000
0x0000	0	-10.24000	-5.12000	0.00000	0.00000	0.00000

参照

AIO_In 関数, AIO_Out, AIO_OutW, AIO_Out 関数, AIO_OutW 関数, AIO_Set, Wait

AIO_In 関数使用例

```
Long word0

word0 = AIO_InW(1)
```

AIO_Out

オプションのアナログ I/O 出力チャンネルからアナログ値を出力します。

書式

AIO_Out チャンネル番号, 出力データ [, Forced]

パラメーター

チャンネル番号	アナログ I/O のチャンネル番号を指定します。
出力データ	出力したい電圧[V]、または電流値[mA]を表す Real 型実数を、式または数値で指定します。
Forced	省略可能です。通常は省略します。

解説

チャンネル番号で指定したアナログ出力ポートへ指定の電圧[V]、または電流[mA]を表す Real 値を出力します。アナログ出力ポートの電圧出力レンジおよび電圧、電流出力選択は、ボード上のスイッチで設定します。アナログ IO ボードの出力レンジ設定の範囲外の値を指定した場合は、範囲を超えない境界値(最大, 最小値)を出力します。

指定したチャンネルでロボットの速度情報を出力している場合、AIO_Out コマンドはエラーになります。速度情報出力を停止してから AIO_Out を実行してください。

注意

Forced フラグ

NoPause タスク, NoEmgAbort タスク (Xqt 実行時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)から、非常停止中や安全扉開時にアナログ I/O 出力を行う場合にこのフラグを指定します。非常停止中や安全扉開時にアナログ I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

AIO_In 関数, AIO_OutW, AIO_Out 関数, AIO_OutW 関数, AIO_Set

AIO_Out 使用例

アナログ I/O チャンネル#1 から 7.0[V]を出力します。

```
AIO_Out 1, 7.0
```


AIO_Out 関数

オプションのアナログ I/O 出力チャンネルで出力しているアナログ値を実数値で返します。

書式

AIO_Out(チャンネル番号)

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。

戻り値

指定したアナログ I/O チャンネルの電圧, 電流出力状態を実数値で返します。電圧出力時の単位は[V]、電流出力時の単位は[mA]です。

指定したチャンネルで、ロボットの速度情報を出力している場合も、本関数が実行できます。

参照

AIO_In 関数, AIO_Out, AIO_OutW, AIO_OutW 関数, AIO_Set, Wait

AIO_Out 関数使用例

```
Real rdata01  
  
rdata01 = AIO_Out(1)
```

AIO_OutW

オプションのアナログ I/O 出力チャンネルから 16 ビットアナログ値を出力します。

書式

AIO_OutW チャンネル番号, 出力データ [, Forced]

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。
出力データ 出力データ (0 から 65535 の整数)を、式または数値で指定します。
Forced 省略可能です。通常は省略します。

解説

チャンネル番号で指定したアナログ I/O チャンネルに出力します。
 出力データは 0 から 65535 の整数を、式または数値で指定します。
 出力電圧(電流)は、ボード上のスイッチで設定される出力レンジ設定により下記の通りとなります。

出力データ		出力レンジ設定					
16 進数	10 進数	±10(V)	±5(V)	0-5(V)	0-10(V)	4-20(mA)	0-20(mA)
0xFFFF	65535	9.99970	4.99985	5.00000	10.00000	20.00000	20.00000
0x8001	32769	0.00031	0.00015	2.50008	5.00015	12.00024	10.00031
0x8000	32768	0.00000	0.00000	2.50000	5.00000	12.00000	10.00000
0x0000	0	-10.00000	-5.00000	0.00000	0.00000	4.00000	0.00000

注意

Forced フラグ

NoPause タスク, NoEmgAbort タスク (Xqt 実行時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時にアナログ I/O 出力を行う場合にこのフラグを指定します。
 非常停止中や安全扉開時にアナログ I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

AIO_In 関数, AIO_Out, AIO_Out 関数, AIO_OutW 関数, AIO_Set, Wait

AIO_OutW 使用例

```
AIO_OutW 1, &H8000
```

AIO_OutW 関数

オプションのアナログ I/O 出力チャンネルで出力しているアナログ値を 0~65535 の Long 型整数で返します。

書式

AIO_OutW(チャンネル番号)

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。

戻り値

指定したアナログ I/O チャンネルの出力状態を 0~65535 の Long 型整数で返します。
各出力チャンネルの出力電圧(電流)と戻り値の対応は、アナログ I/O 基板の出力レンジ設定により下記の通りとなります。

出力データ		出力レンジ設定					
16 進数	10 進数	±10(V)	±5(V)	0-5(V)	0-10(V)	4-20(mA)	0-20(mA)
0xFFFF	65535	9.99970	4.99985	5.00000	10.00000	20.00000	20.00000
0x8001	32769	0.00031	0.00015	2.50008	5.00015	12.00024	10.00031
0x8000	32768	0.00000	0.00000	2.50000	5.00000	12.00000	10.00000
0x0000	0	-10.00000	-5.00000	0.00000	0.00000	4.00000	0.00000

指定したチャンネルで、ロボットの速度情報を出力している場合も、本関数が実行できます。

参照

AIO_In 関数, AIO_Out, AIO_OutW, AIO_Out 関数, AIO_Set, Wait

AIO_OutW 関数使用例

```
Long word0
word0 = AIO_OutW(1)
```

AIO_Set

オプションのアナログ I/O 出力チャンネルにロボットの速度情報を出力する。

書式

- (1) AIO_Set チャンネル番号, On, {RefTCPSpeed | RealTCPSpeed | RefECPSpeed | RealECPSpeed }, 最大出力時速度 [, 最小出力時速度] [, Cnv , コンベヤー番号]
- (2) AIO_Set チャンネル番号, Off
- (3) AIO_Set [チャンネル番号]

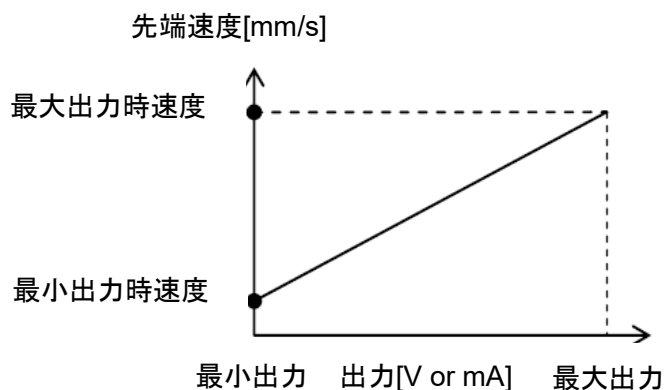
パラメーター

チャンネル番号	アナログ I/O のチャンネル番号を指定します。
On	出力データ (0 から 65535 の整数)を、式または数値で指定します。
Off	速度情報のアナログ出力を終了し、出力 0 に初期化します。
RefTCPSpeed	現在選択されている TCP の指令速度を出力します。
RealTCPSpeed	現在選択されている TCP の実速度を出力します。
RefECPSpeed	現在選択されている ECP の指令速度を出力します。
RealECPSpeed	現在選択されている ECP の実速度を出力します。
最大出力時速度	出力レンジの最大値を出力するときの速度を表す Real 型実数値(単位[mm/s])を、式または数値で指定します。
最小出力時速度	出力レンジの最小値を出力するときの速度を表す Real 型実数値(単位[mm/s])を、式または数値で指定します。省略可能です。省略時は“0”[0mm/s]です。
Cnv	コンベヤーとの相対 TCP 速度を出力します。コンベヤー番号と一緒に指定します。
コンベヤー番号	相対 TCP 速度の計算に使用するコンベヤー番号を指定します。

解説

チャンネル番号で指定したアナログ I/O チャンネルに、選択中のロボットの TCP(ツールセンターポイント) または ECP(外部制御点)の速度をアナログ電圧、電流でリアルタイム出力します。アナログ電圧、電流の選択、および出力レンジの設定はアナログ I/O 基板上のスイッチおよびジャンパーにて設定します。

出力レンジの最小、最大値に対応するロボットの速度は、指定された最小出力時速度、最大出力時速度に応じて下図に示した線形補間によって決まります。



指令速度を指定(RefTCPSpeed、または RefECPSpeed)した場合は、ロボットに与える指令値に基づく理想的な速度波形を出力します。

実速度を指定(RealTCPSpeed、または RealECPSpeed)した場合は、ロボットの実際の動き(各関節のエンコーダー値)に基づいて算出した速度波形を出力します。

TCP を指定(RefTCPSpeed または RealTCPSpeed)した場合は、現在選択されているツール(デフォルト Tool 0)の中心点の速度を出力します。

ECP を指定(RefECPSpeed、または RealECPSpeed)した場合は、現在選択されている外部制御点(ECP)の速度を出力します。ここで、ECP の選択されていない場合(ECP = 0 のとき)は、常に最小出力を出力します。

ECP と Cnv は同時に指定できません。

コンベヤー番号は、設定されているマニピュレーターでコンベヤーキャリブレーションをしたコンベヤー番号のみ指定できます。

チャンネル番号のみ指定した場合は、指定したアナログ I/O チャンネルの出力設定情報を表示します。すべての引数を省略した時は、すべてのアナログ I/O チャンネルの出力設定情報を表示します。

参照

AIO_In 関数, AIO_Out, AIO_Out 関数, AIO_Out, AIO_OutW 関数, AIO_Set, Wait

AIO_Set 使用例

アナログ出力チャンネル 1 にロボット 1、ツール 1 の TCP の実速度出力を設定します。
ロボット動作中の速度をアナログ出力した後、速度出力設定を解除します。

```
Robot 1
Tool 1
Motor On
Power High
SpeedS 2000
AccelS 5000
AIO_Set 1, On, RealTCPSpeed, 2000.0, 0.0
Move P1
AIO_Set 1, Off
```

AIO_Set 関数

オプションのアナログ I/O 出力チャンネルに設定されているロボット速度出力の設定情報を返す。

書式

AIO_Set(チャンネル番号、インデックス)

パラメーター

チャンネル番号 アナログ I/O のチャンネル番号を指定します。
 インデックス 取得する設定情報のインデックスを整数値で指定します。

戻り値

AIO_Set 関数で取得可能な情報は以下の通りです。

インデックス	情報
1	On(1) / Off(0)
2	RefTCPSpeed(0)/ RealTCPSpeed(1)/ RefECPSpeed(2)/ RealECPSpeed(3)
3	最大出力時速度 [mm/s]
4	最小出力時速度 [mm/s]
5	コンベヤー番号 未設定(0)/コンベヤー番号 (1~16)

参照

AIO_In 関数, AIO_Out, AIO_OutW, AIO_Out 関数, AIO_OutW 関数, AIO_Set, Wait

AIO_Set 関数使用例

```
Print "Analog Ch#1 speed output is: ", AIO_Set(1, 1)
```

AIO_TrackingSet

距離追従機能の設定を行います。

書式

- (1) AIO_TrackingSet チャンネル番号, 計測値と距離の変換係数, 距離 0mm の測定値, 追従可能範囲の下限値, 追従可能範囲の上限値 [, 追従可能範囲外での動作 [, 距離追従を行う軸]]
- (2) AIO_TrackingSet チャンネル番号

パラメーター

チャンネル番号 使用する距離センサーが接続されているアナログ I/O のチャンネル番号を 1～8 の整数値で指定します。

計測値と距離の変換係数 距離センサーの計測値(V, mA)を、距離(mm)に変換する係数です。0 をのぞく-500~500 の実数で指定します。(単位: mm/V, mm/mA)

距離 0mm の測定値 距離(変位計の場合: 変位量)が、0mm のときの電圧、または電流値を、下記の範囲の実数で指定します。(単位: V, mA)

アナログ I/O 基板の入力レンジ設定の範囲内の値を設定します。

入力レンジ設定	最小値	最大値
±10.24 V	-10.24 V	10.24 V
±5.12 V	-5.12 V	5.12 V
0-5.12 V	0 V	5.12 V
0-10.24 V	0 V	10.24 V
0-24 mA	0 mA	24 mA

追従可能範囲の下限値 追従可能範囲の下限値は、距離追従機能実行時に許容する変位量の下限値です。下限値は-300~300 の実数で指定します。(単位: mm)
距離センサーの測定可能範囲の下限値より大きな値を指定してください。追従可能範囲の下限値は、追従可能範囲の上限値より小さい値を指定します。

追従可能範囲の上限値 追従可能範囲の上限値は、距離追従機能実行時に許容する変位量の上限値です。上限値は-300~300 の実数で指定します。(単位: mm)
距離センサーの測定可能範囲の上限値より小さな値を指定してください。追従可能範囲の上限値は、追従可能範囲の下限値より大きい値を指定します。

追従可能範囲外での動作 追従可能範囲(上記、下限値から上限値の間)外の場合に、ロボットの動作を停止するか、継続するかを 0~1 の整数で指定します。
省略可能です。省略された場合は、“0”が設定されます。
定数は以下の通りです。

定数	値	内容
AIOTRACK_ERRSTOP	0	追従可能範囲外でロボットがエラー停止します。
AIOTRACK_CONTINUE	1	追従可能範囲外でロボットの動作を継続します。

距離追従を行う軸

距離追従を行う軸を 0~5 の整数で指定します。使用する距離センサーの計測方向と一致する軸を指定してください。

省略可能です。省略された場合は、“2”が設定されます。

定数は以下の通りです。

定数	値	内容
AIOTRACK_TOOL_X	0	Tool 座標 X 軸
AIOTRACK_TOOL_Y	1	Tool 座標 Y 軸
AIOTRACK_TOOL_Z	2	Tool 座標 Z 軸
AIOTRACK_ECP_X	3	ECP 座標 X 軸
AIOTRACK_ECP_Y	4	ECP 座標 Y 軸
AIOTRACK_ECP_Z	5	ECP 座標 Z 軸

3~5は、外部制御点動作(ECP)オプションが有効である場合のみ指定可能です。

結果

書式 2 の場合は、現在の設定値をコンソールに表示します。

以下は、上記パラメーター名とコンソールで表示するパラメーター名の対応表です。

パラメーター名	コンソール表示名
計測値と距離の変換係数	ScaleFactor
距離 0mm の測定値	RefVoltage
追従可能範囲の下限值	ThresholdMin
追従可能範囲の上限値	ThresholdMax
追従可能範囲外での動作	OutOfRangeMode
距離追従を行う軸	TrackingAxis

表示例は以下の通りです。

例 1: チャンネル番号 1 が設定されている場合

Ch1:

ScaleFactor 1.000[V/mm or mA/mm]
 RefVoltage 0.000 [V or mA]
 ThresholdMin -10.000[mm]
 ThresholdMax 10.000[mm]
 OutOfRangeMode AIOTRACK_ERRSTOP
 TrackingAxis AIOTRACK_TOOL_Z

例 2: チャンネル番号 1 が設定されていない場合

Ch1: Undefined

解説

AIO_TrackingSet は、距離追従機能のパラメーターを設定します。設定するパラメーターは、使用する距離センサーや本機能を実施する環境によって決まります。コントローラー電源投入後、AIO_TrackingStart 実行前に必ず AIO_TrackingSet を実行する必要があります。設定したパラメーターは、ロボットコントローラーの電源 OFF、または再起動まで値を保持します。

以下は、パラメーターの詳細な説明です。

計測値と距離の変換係数

+1V あたり+2mm の変位を表す距離センサーの場合、変換係数は 2 です。このとき、+2mm は距離が長くなる方向への変位です。変位計では、距離が短くなる方向の変位に対して電圧が正に設定されている場合があります。その場合は、変換係数は負の値となります。

距離 0mm の測定値

距離センサー、特に変位計は、商品ごとに距離 0mm のときの電圧、または電流値が異なります。また、ユーザーの設定で、距離 0mm のときの電圧、または電流値を任意に設定できるものもあります。使用している距離センサーの設定に応じて値を指定してください。距離(または変位量)が、0mm のときに距離センサーの出力電圧が 0V の場合、本パラメーターは“0”です。

追従可能範囲の上,下限値

アプリケーションで許容されるばらつきに応じて、上下限值を設定します。

設定する値は、必ず距離センサーの測定可能範囲内の値に設定してください。距離センサーの測定可能範囲は、各センサーやユーザーの設定により異なります。距離追従機能実施前に、必ず確認してください。本パラメーターが、距離センサーの測定可能範囲外に設定された場合は、距離追従機能が正しく動作できず、ロボットが意図しない動作をする場合があります。

追従可能範囲外での動作

以下の図は、Tool の Z 方向に対して距離追従機能を実施したときに、“追従可能範囲外での動作”パラメーターの設定を“0”にした場合と、“1”にした場合でのロボットの移動軌跡を示しています。

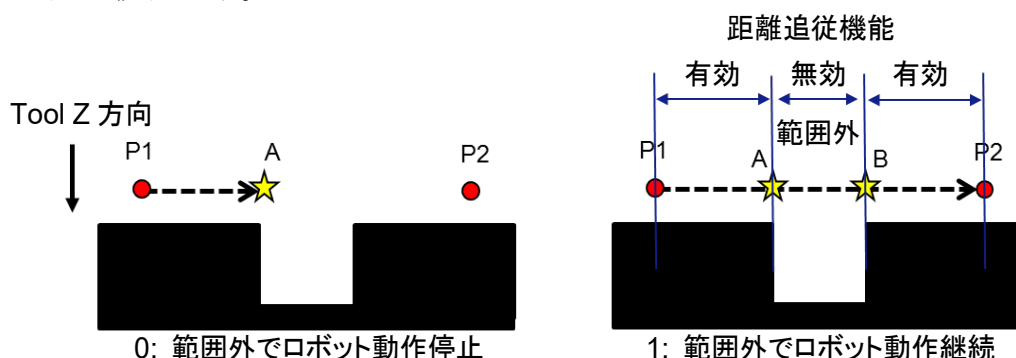
P1: 距離追従開始位置

P2: 目標位置

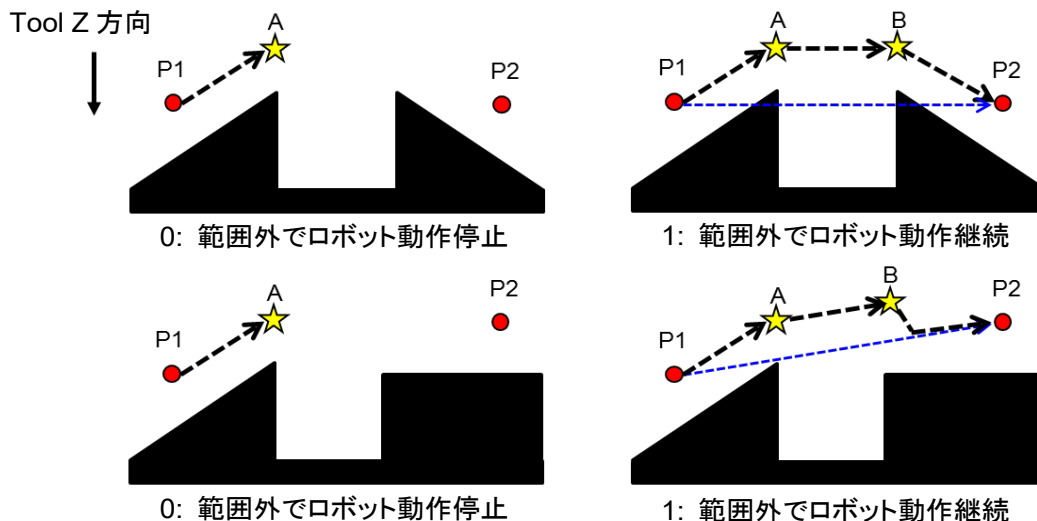
A 点で計測範囲外になり、B 点で計測範囲内に戻ると対象物を図示しています。

距離追従機能は、機能を開始する位置 P1 での Tool Z 方向の測定値(変位量)を基準値として、常に測定値が基準値になるようにロボットを制御します。そのため、P1 から P2 へ移動したとき、P1 から点 A までは測定値が一定になるようにロボットが移動します。

点 A に到達したとき、“0”が設定されている場合は、ロボットは A 点でエラー停止します。“1”が設定されている場合は、点 A から P2 に向かって移動を継続します。ただし、追従可能範囲外のため、追従しません。点 B からは追従可能範囲内に入るため、P1 から点 A までと同様に測定値が一定になるようにロボットが移動します。



“1”を設定した場合の追従可能範囲外でのロボット動作は、開始位置(P1)から、目標位置(P2)に向かって CP 動作する軌道を移動します。以下の図では、追従可能範囲外である点 A-B 間の軌道が、P1-P2 間の軌道と平行になります。点 B に到達後は、追従可能範囲内となるため、計測値を基準値にするロボットの制御が働き、急にロボットが移動する場合があります。



注意

- 各パラメーターが正しく設定されていない場合は、AIO_TrackingStartが実行されたとき、ロボットが意図しない動作をする場合があります。

使用する機器や実施環境に応じて適切に設定してください。

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

参照

AIO_TrackingStart, AIO_TrackingEnd, AIO_TrackingOn 関数

AIO_TrackingSet 使用例

P1 を動作開始位置とし、P2 を動作終了位置として、距離追従機能を使用してロボットが移動するプログラムの例です。



注意

- 使用例で設定しているパラメーターは参考値です。

設定したパラメーターや動作環境によっては作業が成功しない場合や、振動的な動作をする場合があります。

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

Function Main

```
Motor On
Power High
SpeedS 30
AccelS 300,300
```

Go P1

```
AIO_TrackingSet 1,1,0,-5,5,0,2
```

```
AIO_TrackingStart 1,5,5,5
```

Move P2

```
AIO_TrackingEnd
```

```
Motor Off
```

Fend

‘ 開始位置 P1 に移動

‘ 距離追従機能の設定

‘ 距離追従機能の開始

‘ P2 まで距離追従機能実行しながら移動

‘ 距離追従機能の終了

AIO_TrackingStart

距離追従機能を開始します。

書式

AIO_TrackingStart チャンネル番号,ProportionalGain [,IntegralGain [,DifferentialGain]]

パラメーター

チャンネル番号	使用する距離センサーが接続されているアナログ I/O のチャンネル番号を 1～8 の整数値で指定します。
ProportionalGain	距離追従機能の比例ゲインを、0 をのぞく 50 以下の正の実数で指定します。ロボット移動速度やワークの形状などにより最適値が変わるため、使用環境に応じて設定が必要です。
IntegralGain	距離追従機能の積分ゲインを、100 以下の正の実数で指定します。省略可能です。省略された場合は、“0”が設定されます。距離追従精度を上げる場合は、積分ゲインを調整してください。
DifferentialGain	距離追従機能の微分ゲインを、100 以下の正の実数で指定します。省略可能です。省略された場合は、“0”が設定されます。距離追従精度を上げる場合は、微分ゲインを調整してください。

解説

距離追従機能は、アナログ I/O に接続された距離センサーの計測値を使って、ロボットとワークの距離を一定に保持するようにロボットを制御する機能です。

制御するロボットの軸方向は、AIO_TrackingSet のパラメーター“距離追従を行う軸”で指定した 1 つの軸です。保持する距離を“基準値”とすると、本コマンドが実行されたとき、距離センサーで計測した距離が基準値となります。

距離追従機能は、AIO_TrackingStart 実行により開始され、AIO_TrackingEnd 実行で終了します。AIO_TrackingEnd を実行するまでは、距離追従機能が動作し続けます。距離追従機能を使用しない場合は、直ちに AIO_TrackingEnd を実行し、距離追従機能を終了してください。

AIO_TrackingSet が実行される前に、AIO_TrackingStart を実行した場合はエラーになります。必ず AIO_TrackingSet 実行後、AIO_TrackingStart を実行してください。

距離追従機能が使えるロボットタイプは、水平多関節型(RS シリーズを含む)と垂直 6 軸型(N シリーズを含む)です。

距離追従機能実行中に、ロボットは動作できますが、CP 動作に限定されます。PTP 動作はできません。

距離追従機能実行中に特異点近傍を通過した場合は、エラーになります。

以下のコマンドは距離追従機能実行中に実行できません。

MOTOR OFF になるコマンド	Motor off, SFree
PTP 動作コマンド	BGo, Go, JTran, Jump, Jump3, Jump3CP, JumpTLZ, Pass, Ptran, Pulse, TGo
力覚制御実行コマンド	FCKeep, FC 付き動作命令, FS#.Reset, FS.Reboot
トルク制御実行コマンド	TC
コンベヤートラッキング実行コマンド	動作命令 + Cnv_QueueGet
VRT 実行コマンド	VRT, VRT_CPMotion
設定変更コマンド	AIO_TrackingSet, Arm, ArmSet, Base, Calib, CalPls, ECP, ECPSet, Hofs, Inertia, MCal, Power, TLSet, Tool, Weight (AIO_TrackingSet, ArmSet, ECPSet, TLSet は、使用中の番号に関する変更のみエラーになります。)
その他	Brake, Here, Home, VCal, WaitPos

ProportionalGain, IntegralGain, DifferentialGain の設定

ProportionalGain は、設定する値を大きくするほど、ロボットが速く追従します。ただし、値が大き過ぎるとロボットが速く動作して、エラーが発生することがあります。

IntegralGain と DifferentialGain は、省略可能です。補正精度を上げる場合は、設定が必要です。

適切な値を設定しないと、ロボットが高速で動作したり、発振したりする場合があります。

各ゲインの設定方法は、下記マニュアルを参照してください。

EPSON RC+ ユーザーズガイド 19. 距離追従機能



注意

- ProportionalGain, IntegralGain, DifferentialGain は、大きすぎる値を設定した場合、ロボットが意図しない動作をする場合があります。

各パラメーターは、段階的に大きな値に変更してください。1度に大きな値に変更すると、ロボットが意図しない動作をすることがあり、大変危険です。

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

参照

AIO_TrackingSet, AIO_TrackingEnd, AIO_TrackingOn 関数

AIO_TrackingStart 使用例

P1 を動作開始位置とし、P2 を経由して P3 を動作終了位置として、距離追従機能を使用してロボットが移動するプログラムの例です。



注意

- 使用例で設定しているパラメーターは参考値です。

設定したパラメーターや動作環境によっては作業が成功しない場合や、振動的な動作をする場合があります。

また、異常な動作をした場合には、すぐに非常停止ボタンを押してください。

Function Main

Motor On

Power High

SpeedS 30

AccelS 300,300

Go P1

AIO_TrackingSet 1,1,0,-5,5,0,2

AIO_TrackingStart 1,1,0,0

Move P2

Move P3

AIO_TrackingEnd

Motor Off

Fend

‘ 開始位置 P1 に移動

‘ 距離追従機能の設定

‘ 距離追従機能の開始

‘ P2 まで距離追従機能実行しながら移動

‘ P3 まで距離追従機能実行しながら移動

‘ 距離追従機能の終了

AIO_TrackingEnd

距離追従機能を終了します。

書式

AIO_TrackingEnd

解説

AIO_TrackingStart で開始した距離追従機能を終了します。

参照

AIO_TrackingSet, AIO_TrackingStart, AIO_TrackingOn 関数

AIO_TrackingEnd 使用例

P1 を動作開始位置とし、P2 を経由して P3 を動作終了位置として、距離追従機能を使用してロボットが移動するプログラムの例です。



注意

- 使用例で設定しているパラメーターは参考値です。

設定したパラメーターや動作環境によっては作業が成功しない場合や、振動的な動作をする場合があります。

また、異常な動作をした場合には、すぐに非常停止ボタンを押してください。

Function Main

Integer ChNo

Motor On

Power High

Speeds 30

Accels 300,300

ChNo=1

Go P1

AIO_TrackingSet ChNo,10,0,-3,3,0,2

AIO_TrackingStart ChNo,1,0,0

Move P2

Move P3

AIO_TrackingEnd

Motor Off

Fend

‘ 開始位置 P1 に移動

‘ 距離追従機能の設定

‘ 距離追従機能の開始

‘ P2 まで距離追従機能実行しながら移動

‘ P3 まで距離追従機能実行しながら移動

‘ 距離追従機能の終了

AIO_TrackingOn 関数

指定したロボットが距離追従機能を実行しているか否かを返します。

書式

AIO_TrackingOn(ロボット番号)

パラメーター

ロボット番号 状態を取得したいロボットの番号を式、または数値で指定します。

戻り値

距離追従機能実行中は、True(-1)、停止中は、False(0)を返します。

参照

AIO_TrackingSet, AIO_TrackingStart, AIO_TrackingEnd

AIO_TrackingOn 使用例

```
Function Main
    Integer i
    i = AIO_TrackingOn(1)
    print i
End
```

コマンドウィンドウでの使用例

```
>print AIO_TrackingOn(1)
0
```

Align 関数

指定ポイントにおけるロボットのツール座標系の姿勢 (U, V, W) を、指定したローカル座標系の座標軸のうち最も近い座標軸や指定した座標軸に整列するように変換したポイントデータを返します。

書式

(1) Align (ポイント指定[, ローカル座標系番号[, 座標軸指定]])

パラメーター

ポイント指定	対象とするポイントデータを指定します。
ローカル座標系番号	姿勢を整列するための基準にしたいローカル座標系番号を指定します。省略するとベース座標系が指定されます。
座標軸指定	整列したい座標軸を指定します。省略すると最も近い座標軸に整列します。

定数	値
COORD_X_PLUS	1: +X 軸
COORD_Y_PLUS	2: +Y 軸
COORD_Z_PLUS	3: +Z 軸
COORD_X_MINUS	4: -X 軸
COORD_Y_MINUS	5: -Y 軸
COORD_Z_MINUS	6: -Z 軸

解説

垂直 6 軸型ロボット (N シリーズを含む) では、ポイントデータによって定義されたツール座標系の位置 (原点) を固定したまま、姿勢だけを変化させて特定の座標系に整列させたいことがあります。Align 関数は指定したポイントデータの姿勢データ (U, V, W 値) を指定したローカル座標系の座標軸のうち最も近い座標軸に整列したり、指定した座標軸に整列したりするように変換します。

垂直 6 軸型 (N シリーズを含む) 以外のロボットでは、指定ポイントをそのまま返します。

参照

AlignECP 関数, LJM 関数

Align 関数使用例

```
Move Align(P0) ROT

P1 = Align(P0, 1)
Move P1 ROT

P2 = Align(P0, 1, 3)
Move P2 ROT
```


AlignECP 関数

指定ポイントにおけるロボットのツール座標系の姿勢 (U, V, W) を、指定した ECP 座標系の座標軸のうち最も近い座標軸に整列するように変換したポイントデータを返します。

書式

(1) AlignECP (ポイント指定, ECP 座標系番号)

パラメーター

ポイント指定	対象とするポイントデータを指定します。
ECP 座標系番号	姿勢を整列するための基準にしたい ECP 座標系番号を指定します。

解説

垂直 6 軸型ロボット (N シリーズを含む) では、ポイントデータによって定義されたツール座標系の位置 (原点) を固定したまま、姿勢だけを変化させて特定の座標系に整列させたいことがあります。AlignECP 関数は指定したポイントデータの姿勢データ (U, V, W 値) を指定した ECP 座標系の座標軸のうち最も近い座標軸に整列するように変換します。

垂直 6 軸型 (N シリーズを含む) 以外のロボットでは、指定ポイントをそのまま返します。

参照

Align 関数, LJM 関数

AlignECP 関数使用例

```
Move AlignECP (P0) ROT
P1 = AlignECP (P0, 1)
Move P1 ROT
```

And 演算子

2つの値の And 演算 (論理またはビット)を行います。

書式

result =値 1 And 値 2

パラメーター

値 1, 値 2 論理 And 演算では、論理値を返す値を指定します。ビット And 演算では、整数式を指定します。

result 論理 And 演算では、論理値を返します。ビット And 演算では、整数を返します。

解説

論理 And 演算は2つ以上の値を結合し、Boolean 型の結果を出します。次の表は And 演算のパターンを示しています。

値 1	値 2	result
True	True	True
True	False	False
False	True	False
False	False	False

ビット And 演算は、2つの数値をビット単位で比較し、次の表にしたがって対応するビットを **result** に出します。

値 1 のビット	値 2 のビット	result
0	0	0
0	1	0
1	0	0
1	1	1

参照

LShift, Mask, Not, Or, RShift, Xor

And 演算子使用例

```
Function LogicalAnd(x As Integer, y As Integer)

    If x = 1 And y = 2 Then
        Print "The values are correct"
    EndIf
Fend

Function BitWiseAnd()

    If (Stat(0) And &H8000000) = &H8000000 Then
        Print "The enable switch is open"
    EndIf
Fend

>print 15 and 7
7
>
```

AOpen

ファイルをアペンドモード (追加書き込み) で開きます。

書式

AOpen ファイル名 As #ファイル番号

・
・

Close #ファイル番号

パラメーター

ファイル名	パスを含んだファイル名の文字列を指定します。 ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。 詳細は、ChDisk を参照してください。
ファイル番号	30~63 の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号でオープンします。このステートメントは指定したファイルにアペンド (追加書き込み) するときに使います。指定したファイルが存在しない場合、新規でファイルを作成します。

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号は使えません。ファイル番号は、ファイル操作命令 (Print#, Write, Flush, Close) で使用します。

Close ステートメントでファイルを閉じ、ファイル番号を解放します。

複数のタスクで同一の番号を使用しないように、FreeFile 関数でファイル番号を取得してください。

注意

ネットワークパスが使用できます。

ファイルへの書き込みはバッファリングされます。

バッファリングされたデータは Flush ステートメントで書き込むことができます。Close ステートメントでファイルを閉じた場合も書き込まれます。

参照

Close, Print #, BOpen, ROpen, UOpen, WOpen, FreeFile, Flush

AOpen 使用例

```
Integer fileNum, i
fileNum = FreeFile
WOpen "TEST.DAT " As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum
::::
FileNum = FreeFile
AOpen "TEST.DAT" As #FileNum
For i = 101 to 200
    Print #FileNum, i
Next i
Close #FileNum
```

Arc, Arc3

Arc は XY 平面で、アームを現在位置から指定位置まで円弧補間動作で動かします。
 Arc3 は 3 次元で、アームを現在位置から指定位置まで円弧補間動作で動かします。
 この 2 つのコマンドは、ロボットが、水平多関節型 (RS シリーズを含む) でも垂直 6 軸型 (N シリーズを含む) でも使うことができます。

書式

- (1) Arc 経由座標, 目標座標 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]
 (2) Arc3 経由座標, 目標座標 [ROT] [ECP] [CP] [Till | Find] [!並列処理!] [SYNC]

パラメーター

経由座標	ポイントデータ、もしくは XY 関数で指定します。アームが現在位置から目標座標に移動するまでの軌道で通過しなくてはならないポイントです。
目標座標	ポイントデータ、もしくは XY 関数で指定します。アームが円弧動作で移動する到達地点、目標位置です。
ROT	ツール姿勢変化を優先させて、動作の速度、加減速度を決めます。省略可能です。
ECP	外部制御点動作を指定します。省略可能です。(ECP オプション使用時のみ有効)
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
! 並列処理 !	Arc ステートメントには並列処理ステートメントが使えます。省略可能です。(詳細については“並列処理”を参照してください。)
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

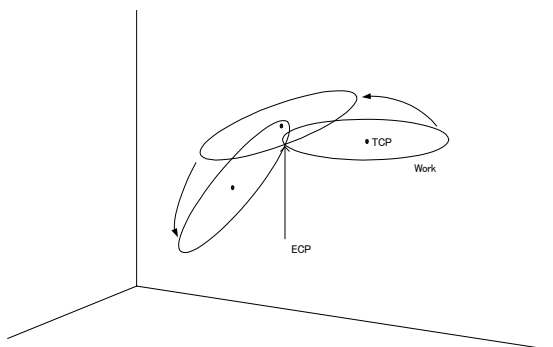
Arc と Arc3 は、アームを現在位置から目標座標まで、経由座標を通過して円弧補間動作で動かすときに使われます。与えられた 3 点 (現在地, 経由座標, 目標座標) から自動的に円弧補間軌道を演算し、目標座標に至るまで、その軌道に沿ってアームを動かします。
 また、水平多関節型ロボットの場合は、現在地から目標座標の点へ U 座標が動きますが、垂直 6 軸型ロボットの場合は、現在地から目標座標に向かう最短の回転姿勢で U, V, W 座標が動きます。
 使用する場合は、実際の動きを事前に確認してください。

Arc と Arc3 の速度と加減速度は、それぞれ SpeedS と AccelS の設定値を使用します。速度と加減速度の関係については、“注意”の“Arc, Arc3 を CP とともに用いる”を参照してください。ただし、ROT 修飾パラメーターを指定しているときの速度と加減速度はそれぞれ SpeedR と AccelR の設定値を使用します。この場合、SpeedS と AccelS の設定値は無効となります。

通常、移動距離が 0 で、姿勢関節のみが動作するような場合にはエラーとなります。ROT 修飾パラメーターを付けてツール姿勢変化の加減速を優先することで、エラーなく動作が可能になります。ROT 修飾パラメーターを付けたとき、姿勢変化がなく、移動距離が“0”でない場合には、エラーとなります。

また、移動距離に対してツール姿勢変化速度が大きすぎる場合や、指定された回転速度がマニピュレーターの限度を超えている場合もエラーとなります。その場合は指定速度を落とすか、または ROT 修飾パラメーターを付けて姿勢変化の加減速度を優先するようにしてください。

ECPを使っている場合 (Arc3 のみ)、指定した ECP 番号に対応する外部制御点で、ワークが円弧の軌道に沿って動きます。このとき先端関節の中心は、円弧軌道をたどりません。



Arc 動作の速度と加速度の設定

Arc と Arc3 命令に対する速度と加減速度の設定は、それぞれ SpeedS と AccelS により行います。SpeedS で速度 (単位: mm/sec)、AccelS で加減速度 (単位: mm/sec²) を指定します。

注意

Arc 命令は水平面においてのみ有効です

Arc 命令によって描かれる軌道は XY 平面上での真円弧になります。Z 方向や姿勢については、現在のポイントと目標座標値を補間します。Arc3 では 3 次元空間における円弧軌道を指定することができます。

Arc 命令に対する範囲確認

Arc と Arc3 ステートメントは、Arc 動作に先立って軌道範囲確認の演算を行うことができません。したがって、目的位置が動作エリア内でも、軌道がエリア外にでると停止することがあります。そのとき衝撃が生じ、アームに障害を与える可能性がありますので、あらかじめプログラムを低速で走らせて軌道を確認しておく必要があります。

Arc 動作の設定にあたって

Arc コマンドによる円弧動作は、現在位置から開始されるため、Arc と Arc3 の実行に先立って、あらかじめ Go や Jump その他の関連動作コマンドを使って、ロボットのアームを適切な位置に動かしておくことが必要な場合もあります。

Arc, Arc3 を CP とともに用いる

CP パラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。CP 指定なしの Arc 命令, Arc3 命令では、アームは必ず減速して、指定された目標座標に停止します。

起こりやすいエラー

ハンド(腕系)の属性を変更する

Arc 命令を使用する場合は、各ポイントのハンドの属性に注意してください。円弧補間動作の間にハンドの向きが変更されると (例えば、右腕から左腕、あるいはその逆の変更など)、エラーになります。アームの属性値 (L 左腕, R 右腕) は、実際の現在位置、経由座標や目標座標と一致していなければいけません。

可動範囲外にアームを動かそうとすると

もし指定された円弧動作が、アームを可動範囲外に動かそうとしていると、エラーになります。

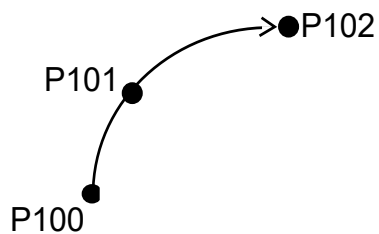
参照

! 並列処理 !, AccelS, Move, SpeedS

Arc, Arc3 使用例

下記は、図のような軌道を描くプログラムの例です。P100 から動作を開始し、P101 を経由して、P102 に到着する円弧動作軌道をとります。

```
Function ArcTest  
  Go P100  
  Arc P101, P102  
Fend
```

**ヒント**

初めて Arc 命令を使うときには、可動範囲の中にあるロボットの側のポイントを使って、簡単な円弧で試してみることをお勧めします。そのとき、実際に描かれる円弧軌道を想定してみてください。そしてアームが通常の可動範囲外を移動してしまうポイントをティーチングしないようにしてください。

Arch

Jump, Jump3, Jump3CP 命令でのアーチパラメーターの設定と表示をします。

書式

- (1) Arch アーチ番号, 退避距離, 接近距離
- (2) Arch アーチ番号
- (3) Arch

パラメーター

アーチ番号	アーチ番号を 0~6 の整数で指定します。有効値は 0~6 の整数で、次ページの Arch テーブルのように有効な値は全部で 7 つあります。
退避距離	Jump 命令で水平動作前の退避距離 (出発点からの垂直距離)を指定します。 (単位: mm) Jump3, Jump3CP 命令で、スパン動作前の退避距離を指定します。(単位: mm)
接近距離	Jump 命令で水平移動が完全に終了した段階での、接近距離 (目的点からの垂直距離)を指定します。(単位: mm) Jump3, Jump3CP 命令で、スパン動作が完全に終了した段階での、接近距離を指定します。(単位: mm)

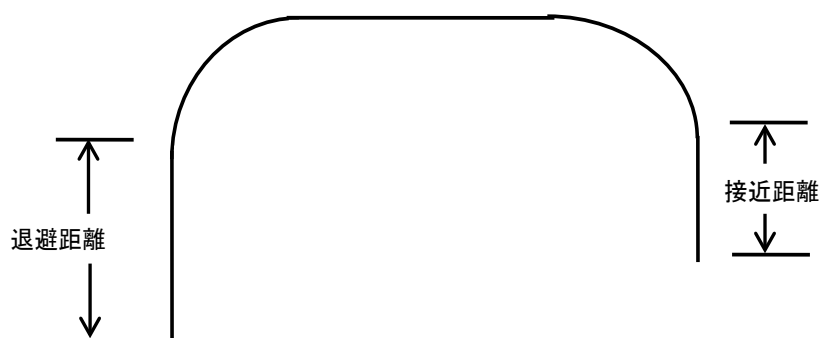
結果

すべてのパラメーターを省略すると、Arch テーブルの内容すべてが表示されます。
アーチ番号だけを指定すると、指定アーチ番号の Arch テーブル内容が表示されます。

解説

Arch 命令で、Jump 動作命令で必要な Arch テーブルの値を定義します。Arch 動作は、Jump の修飾語である、アーチ番号に対応したパラメーターで実行されます。(Arch 命令を理解するためには、まず Jump ステートメントをよくお読みください。)

Arch の設定により、Jump C[アーチ番号]を使うと Z 方向の角を丸めることができます。(使用例を参照)
Arch テーブルでは、水平方向移動を始める前の垂直方向の移動距離 (退避距離)と、水平方向移動が終了した後の、目標座標までの垂直方向の移動距離(接近距離)を設定します。(下図を参照してください。)



ユーザー定義の Arch テーブルの値は、0~6 までの整数で、全部で 7 つあります。8 番目の設定値 (Arch 7) は、デフォルト値で、実際にはアーチモーションではなく、ゲートモーション (次図参照) が設定されています。デフォルト Arch 値 (8 番目の設定値) を使って **Jump** 命令を実行すると、アームは次図のように動作します。

- 1) まず、第 3 関節だけが、LimZ コマンドによって設定された Z 座標値 (最大 Z 値) まで、移動します。
- 2) 次に、目標座標までアームが水平移動し、最終的な X, Y, U 位置まで行きます。
- 3) 最後に第 3 関節だけが動き、アームを最終的な第 3 関節座標位置 (Z 座標値) まで下降させ、**Jump** 命令動作が完了します。

ゲートモーション (Arch 7 の Jump 動作)



Arch テーブル デフォルト値

Arch 番号	退避距離	接近距離
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

注意

ゲートモーションになるその他の場合

実際の垂直移動距離より大きい値が、垂直上昇距離、垂直下降距離に設定されると、アーチモーションにはならず、ゲートモーションになります。

Arch 値は保存されます。

Arch テーブルの値は、ユーザーが変更しない限り、保持されます。

アーチ使用時の重要事項

アーチモーションは、動作の合成を軌跡制御上で行うため、実際の軌跡を保証できるものではありません。動作速度やアームの動き方によってその軌跡は変化します。実際の軌跡は、作業で使用する実速度と姿勢で確認してください。

- 同じ位置で、同じ C[アーチ番号]を持つ **Jump** 命令 (または **Jump3** 命令)を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- 低速動作時より高速動作の方が、合成なしの退避移動量が増え、合成なしの接近移動量が減る傾向にあります。期待した移動距離が出ない場合は、速度や減速度を下げるか、接近距離を長めに設定してください。
- 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。アームの動き方による軌跡の変化はさまざまですが、一般的な水平多関節型ロボットを例にとれば、第 1 アームを大きく動かすほど垂直上昇量が増え、垂直下降量が減る傾向が強くなります。期待した垂直下降距離が出ない場合は、速度や減速度を下げるか、下降距離を長めに設定してください。

参照

Jump, Jump3, Jump3CP

Arch 使用例

下記は、コマンドウィンドウから行う Arch 値の設定例です。

```
> arch 0, 15, 15
> arch 1, 25, 50
> jump p1 c1
> arch
arch0 =      15.000      15.000
arch1 =      25.000      50.000
arch2 =      50.000      50.000
arch3 =      60.000      60.000
arch4 =      70.000      70.000
arch5 =      80.000      80.000
arch6 =      90.000      90.000
>
```

Arch 関数

アーチの設定を返します。

書式

Arch(アーチ番号, パラメーター番号)

パラメーター

アーチ番号 0~6 の整数を指定します。

パラメーター番号 1: 退避距離
 2: 接近距離

戻り値

パラメーター番号で指定した距離を返します。

参照

Arch

Arch 関数使用例

```
Double archValues(6, 1)
Integer i

'現在のアーチの設定を保存
For i = 0 to 6
    archValues(i, 0) = Arch(i, 1)
    archValues(i, 1) = Arch(i, 2)
Next i
```

AreaCorrection 関数

補正エリアを用いて補正をしたポイントを返す関数

書式

AreaCorrection(ポイント指定, エリア番号)

パラメーター

ポイント指定	補正の対象とするポイントデータを指定します。
エリア番号	エリア番号(1~8 の整数)を式または数値で指定します。

解説

あらかじめ定義した補正エリアをもとに、補正をした結果のポイントを返します。補正前のポイントと同じローカル座標系で定義された座標です。この関数を動作命令(Go、または Jump 命令など)とともに用いることにより、ロボットを指定位置に動かすことができます。本コマンドを使用することにより、指定ポイントの位置精度が向上します。ポイント指定では、図面上での位置を入力してください。

補正は位置に対してのみ適用されます。付加軸, UVW 座標値, 姿勢フラグに関しては、補正が適用されません。入力されたポイントデータの値がそのまま出力されます。

設定されていない補正エリアを指定した場合は、エラーになります。

注意

ティーチング済みのポイント

ティーチング済みのポイントデータに対して AreaCorrection 関数を適用しないでください。合わせこんだティーチング位置に対して補正されてしまい、位置がずれてしまいます。

補正エリアから遠い場合

AreaCorrectionSet で設定した補正エリアから遠い場合、補正の効果が減少します。補正エリアは動作点を取り囲むように基準点を設定してください。

補正の種類に平面を選択している場合、補正エリアに選択した平面から垂直方向に距離のあるポイントでは、補正の効果が減少します。適切な高さに補正エリアを設定するか、基準点を設けられる場合は、補正の種類に空間を指定してください。

補正エリアを設定した姿勢フラグと異なる場合

AreaCorrectionSet で設定した基準点と姿勢フラグが異なる場合、エラーが発生します。

補正エリアを設定した姿勢(U, V, W)と異なる場合

スカルロボット(RS シリーズを含む)の場合、補正できます。

垂直 6 軸型ロボット(N シリーズを含む)では、補正前のポイントにおけるツール座標系 Z 軸と、補正エリアの基準点のポイントにおけるツール座標系 Z 軸が一致する場合、補正できます。一致しない場合は、補正対象外となり、エラーが発生します。ツール座標系 Z 軸の角度は DiffToolOrientation 関数の軸番号に COORD_Z_PLUS を指定すると取得できます。

参照

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef 関数, AreaCorrectionInv, AreaCorrectionOffset 関数, DiffToolOrientation 関数

AreaCorrection 使用例

Function sample

' P(1:4) 基準点

P1 = XY(-100, 200, -20, 0)

P2 = XY(100, 200, -20, 0)

P3 = XY(-100, 400, -20, 0)

P4 = XY(100, 400, -20, 0)

' P(11:14) 実際は P(1:4)をティーチングしたポイントを用いる

P11 = XY(-100, 200.5, -20, 0)

P12 = XY(100.3, 200.1, -20, 0)

P13 = XY(-100.4, 400.8, -20, 0)

P14 = XY(100.2, 400.4, -20, 0)

' 補正エリアを設定

AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE

P999 = **AreaCorrection**(P1, 1) ' P999 は補正済みのポイント

Print Dist(P11, P999)

P999 = **AreaCorrection**(XY(0, 300, -20, 0), 1) ' エリア内の点を補正

Print P999

Fend

[出力結果]

0

X: 0.100 Y: 300.450 Z: -20.000 U: 0.000 /R /0

AreaCorrectionClr

補正エリアをクリアします。

書式

AreaCorrectionClr エリア番号

パラメーター

エリア番号 エリア番号(1~8 の整数)を式または数値で指定します。

結果

エリア番号に対応する補正エリアをクリアします。
ロボット動作中は本命令を実行できません。停止した状態で使用してください。

参照

AreaCorrectionSet, AreaCorrectionDef 関数, AreaCorrectionInv, AreaCorrectionOffset 関数

AreaCorrectionClr 使用例

AreaCorrectionClr 1

AreaCorrectionDef 関数

指定した補正エリアが設定されているかを返します。

書式

AreaCorrectionDef(エリア番号)

パラメーター

エリア番号 エリア番号(1~8 の整数)を式または数値で指定します。

戻り値

補正エリアが設定されている場合は “True”、それ以外は “False” を返します。

参照

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionInv, AreaCorrectionOffset 関数

AreaCorrectionDef 関数 使用例

```
Function DisplayAreaCorrectionDef(areaNum As Integer)

    If AreaCorrectionDef(areaNum) = False Then
        Print "Area", areaNum, " is not defined"
    Else
        Print "Area Definition:"
        AreaCorrectionSet areaNum
    EndIf
End
```

AreaCorrectionInv 関数

補正済みのポイントをもとに戻す関数

書式

AreaCorrectionInv(ポイント指定, エリア番号)

パラメーター

ポイント指定	補正の対象とするポイントデータを指定します。
エリア番号	エリア番号(1~8 の整数)を式または数値で指定します。

解説

AreaCorrection 関数で補正済みのポイントに対して、補正前のポイントデータを返します。
 実際にティーチングをして作成したポイント、または補正済みのポイントに対して AreaCorrectionInv を適用すると、補正前のポイントデータが得られます。
 設定されていない補正エリアを指定した場合は、エラーになります。

参照

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef 関数, AreaCorrectionOffset 関数

AreaCorrectionInv 使用例

```
Function AreaCorrectionTest
  ' P(1:4) 基準点
  P1 = XY(-100, 200, -20, 0)
  P2 = XY(100, 200, -20, 0)
  P3 = XY(-100, 400, -20, 0)
  P4 = XY(100, 400, -20, 0)
  ' P(11:14) 実際は P(1:4)をティーチングしたポイントを用いる
  P11 = XY(-100, 200.5, -20, 0)
  P12 = XY(100.3, 200.1, -20, 0)
  P13 = XY(-100.4, 400.8, -20, 0)
  P14 = XY(100.2, 400.4, -20, 0)
  ' 補正エリアを設定
  AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
  P888 = AreaCorrection(P1, 1) ' P888 は補正済みのポイント
  P999 = AreaCorrectionInv(P888, 1) ' P999 は変換前のポイント
  Print Dist(P11, P888)
  Print Dist(P1, P999)
End
```

[出力結果]

```
0
0
```

AreaCorrectionOffset 関数

補正済みのポイントから相対移動したポイントを返す関数

書式

AreaCorrectionOffset(ポイント指定, 相対移動量指定, エリア番号[, 相対関係選択])

パラメーター

ポイント指定	相対移動の基準となる位置のポイントデータを指定します。	
相対移動量指定	相対移動量をポイントデータで指定します。	
エリア番号	エリア番号(1~8 の整数)を式または数値で指定します。	
相対関係選択	どの座標系を基準として相対移動するかを表します。省略した場合は、ローカル座標系基準になります。	
	ローカル座標系基準で動かす場合は、ポイント指定が定義されている座標系を基準として相対移動した座標になります。ツール座標系基準で動かす場合は、ポイント指定位置を基準として相対移動した座標になります。	
相対関係選択	定数	値
ローカル座標基準	AC_LOCAL	0
ツール座標基準	AC_TOOL	1

解説

AreaCorrection 関数で補正済みのポイントに対して、相対移動したポイントを返します。指定したポイントと同じローカル座標系で定義された座標になります。この関数を動作命令(Go または Jump 命令など)と同時に使用することにより、ロボットを指定位置に動かすことができます。

Here 関数と合わせて使うことにより、BGo, TGo と同様の動作を行うことができます。補正エリアの範囲内にある場合、相対移動量がより正確になります。

設定されていない補正エリアを指定した場合はエラーになります。

注意

姿勢の相対移動を行った場合

姿勢の相対移動を行った場合、相対移動後の姿勢が補正対象外になり、エラーが発生することがあります。

参照

AreaCorrectionSet, AreaCorrectionClr, AreaCorrectionDef 関数, AreaCorrectionInv, Here

AreaCorrectionOffset 使用例

```
' 補正エリア 1 が定義済みとする
' BGo XY(50, 0, 0, 0) と同様
Go AreaCorrectionOffset (Here, XY(50, 0, 0, 0), 1)
' TGo XY(50, 0, 0, 0) と同様
Go AreaCorrectionOffset (Here, XY(50, 0, 0, 0), 1, AC_TOOL)
```


AreaCorrectionSet

補正エリアの設定と表示を行います。

書式

- (1) AreaCorrectionSet エリア番号, 基準連続ポイント, 教示連続ポイント, 補正の種類
- (2) AreaCorrectionSet エリア番号
- (3) AreaCorrectionSet

パラメーター

エリア番号	エリア番号(1~8の整数)を式または数値で指定します。		
基準連続ポイント	基準点としたいポイントデータの連番を、始点と終点の2つのポイント番号をコロンで結合して P(1:4)のように指定します。 補正の効果を最大にするために、基準点は補正したいポイントを取り囲むように選択してください。		
教示連続ポイント	基準連続ポイントに対応する教示したポイントデータを、始点と終点の2つのポイント番号をコロンで結合して P(1:4)のように指定します。ポイントデータの並び順は、基準連続ポイントに対応するように設定してください。		
補正の種類	補正の種類を表す整数値です。 平面補正では、基準点として選んだポイントによって構成される平面上のポイントに対して補正を行うことができます。平面補正を選択する場合は、基準連続ポイントを平面上に配置してください。基準点数は、最低3点必要になります。 空間補正では、基準点として選んだポイントによって構成される立体空間上のポイントに対して補正を行うことができます。立体補正を選択する場合は、基準連続ポイントが補正したい領域を取り囲むように選択してください。基準点数は、最低4点必要になります。		
	補正の種類	定数	値
	平面	MODE_PLANE	2
	空間	MODE_SPACE	3

結果

- (1)の書式で指定すると、指定したエリア番号で補正エリアが設定されます。
- (2)の書式で指定すると、指定したエリア番号の内容を表示します。
- (3)の書式で指定すると、定義された補正エリアの内容がすべて表示されます。

解説

エリア補正機能で用いる補正エリアを設定します。補正エリアを設定し、AreaCorrection 関数, AreaCorrectionInv 関数, AreaCorrectionOffset 関数を使用することにより、補正エリア内のポイントの位置精度を向上することができます。

動作中は本命令を実行できません。停止した状態で使用してください。

基準位置の選定方法は、以下のマニュアルを参照してください。

EPSON RC+ ユーザーズガイド 22.3 エリア補正機能

注意

補正エリアデータについて

補正エリアは、コントローラーの電源がオフになるまで有効です。コントローラー起動時は、補正エリアが定義されていない状態です。

ツールについて

AreaCorrection 関数を用いて補正を行う場合は、補正エリアの基準点をティーチングしたときに使用したツールと同じツールを用いてください。異なるツールを使用した場合、補正効果が減少する場合があります。

補正エリアから遠い場合

AreaCorrectionSet で設定した補正エリアから遠い場合、補正の効果が減少します。補正エリアは動作点を取り囲むように基準点を設定してください。

補正の種類に平面を選択している場合、補正エリアに選択した平面から垂直方向に距離のあるポイントでは、補正の効果が減少します。適切な高さに補正エリアを設定するか、高さ方向に基準点を設けられる場合は、補正の種類に空間を指定してください。

補正エリアを設定した姿勢フラグと異なる場合

AreaCorrectionSet で設定した基準点と姿勢フラグが異なる場合、エラーが発生します。動作を行うポイントと同じ姿勢フラグを基準点として設定してください。

補正エリアを設定した姿勢(U,V,W)と異なる場合

スカラロボット(RS シリーズを含む)の場合、補正できます。

垂直 6 軸型ロボット(N シリーズを含む)では、補正前のポイントにおけるツール座標系 Z 軸と、補正エリアの基準点のポイントにおけるツール座標系 Z 軸が一致する場合、補正できます。一致しない場合は、補正対象外となり、エラーが発生します。ツール座標系 Z 軸の角度は、DiffToolOrientation 関数の軸番号に COORD_Z_PLUS を指定すると、取得できます。

参照

AreaCorrectionClr, AreaCorrectionDef 関数, AreaCorrectionInv, AreaCorrectionOffset 関数, DiffToolOrientation 関数

AreaCorrectionSet 使用例

使用例を示します。P11~P14 には、ティーチングしたポイントを用いてください。
以下のように P1~P4 を基準点の図面上での位置とした場合、補正エリアは P1, P2, P3, P4 を頂点とする幅 200mm の正方形になります。

```
Function AreaCorrectionTest
  ' P(1:4) 基準点
  P1 = XY(-100, 200, -20, 0)
  P2 = XY(100, 200, -20, 0)
  P3 = XY(-100, 400, -20, 0)
  P4 = XY(100, 400, -20, 0)
  ' P(11:14) 実際は、P(1:4)をティーチングしたポイントを用いる
  P11 = XY(-100, 200.5, -20, 0)
  P12 = XY(100.3, 200.1, -20, 0)
  P13 = XY(-100.4, 400.8, -20, 0)
  P14 = XY(100.2, 400.4, -20, 0)
  ' 補正エリアを設定
  AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
Fend
```

Arm

アームの選択と、現在選択されているアーム番号の表示をします。

書式

- (1) Arm アーム番号
- (2) Arm

パラメーター

アーム番号	整数値、または式で指定します。有効値範囲は、0~15 で、16 個までの異なるアームを選択できます。アーム 0 は、標準 (デフォルト設定) のロボットアームです。アーム 1~15 は、ArmSet によって定義される増設アームです。省略可能で、省略したときには、現在のアーム番号が表示されます。

結果

パラメーターの設定なしで Arm 命令が実行されると、現在選択されているアーム番号が表示されます。

解説

どのアームでロボット命令を実行するか指定します。Arm で増設アームにポジションデータを共用させることができます。増設アームの設定がない場合は、標準アーム (アーム番号 0) が作動します。出荷時にはアーム番号は“0”に設定されていますので、増設アームがない場合は設定を変更する必要はありません。しかし、増設アームを使う場合は、ArmSet により、最初に設定してください。

実際のロボット構成が標準的な構成とは異なる場合でも、増設アームは、個々のロボットに合った適切なパラメーター設定ができるよう対応しています。増設アームは、下記の条件下で正しく作動します。

- 1 つのポイントデータを 2 つ以上のアームが通過するように設定する。
- Pallet を使う。
- CP 動作を使う。
- 相対位置指定を行う。
- ローカル座標を使う。

回転関節を使う水平多関節型ロボット (RS シリーズを含む) の場合、関節角度は ArmSet パラメーターで設定される値を基準に演算されます。したがって、増設アームやハンドの設定が必要な状況では、このコマンドを使用してください。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意**Arm 0**

Arm 0 について ArmSet を使って設定, 変更することができません。Arm 0 は、標準的なロボット構成の設定に使われるようになっています。Arm 値を“0”に設定すると、ロボットアームの標準パラメーターを使うように設定します。

アーム長補正オプション使用時

ArmCalib を On にすると、Arm 0 にアーム長補正值が適用され、自動的に Arm 0 に切り替わります。アーム長補正值を適用した動作を行う場合は、Arm 0 で使用してください。ArmCalib を On にしても、Arm 0 以外では、アーム長補正值を適用した動作になりません。ロボットアームの標準パラメーターを使用する場合は、ArmCalib を Off にして、Arm 0 を使用してください。

設定されていないアーム番号

ArmSet で定義されていない増設アームの番号を選択するとエラーになります。

参照

ArmClr, ArmSet, ECPSet, TLSet, ArmCalibSet

Arm 使用例

下記のプログラム例では、ArmSet と Arm を使って、増設アームの設定を行っています。ArmSet で増設アームを定義し、Arm でどのアームを現在のアームとして選択するか設定します。(Arm 0 はデフォルトのロボットアームでユーザーが変更することはできません。)

コマンドウィンドウでの操作例

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  arm0 250  0  0  300  0
  arm1 300 -12 -30 300  0

> Arm 0
> Jump P1      '標準アームの設定で P1 へ Jump
> Arm 1
> Jump P1      '増設アーム 1 の設定で P1 へ Jump
```

Arm 関数

現在選択されているロボットのアーム番号を返します。

書式

Arm

戻り値

現在のアーム番号を返します。

参照

Arm

Arm 関数使用例

```
Print "The current arm number is: ", Arm
```

ArmCalib

現在選択されているロボットのアーム長補正を有効、または無効にします。

書式

ArmCalib On | Off

パラメーター

On | Off アーム長補正を有効状態にする場合には On を、無効状態にする場合には Off を指定します。

解説

ArmCalib On 命令はアーム長補正を有効状態にします。ArmCalib On を実行すると、Arm 0 に ArmCalibSet で指定した補正值が設定され、Arm が “0” に切り替わります。

ArmCalib Off 命令はアーム長補正を無効状態にします。ArmCalib Off を実行すると、Arm 0 に標準パラメーターが設定されます。ArmCalib Off を実行時、Arm は自動で切り替わりません。

アーム長補正オプションを購入した場合は、出荷時にアーム長補正が有効に設定されています。アーム長補正有効状態で Arm を “0” 以外に設定した場合は、設定した Arm 番号の設定で動作します。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

このコマンドは、アーム長補正オプションがインストールされている場合のみ使用できます。

ArmCalib On 状態のバックアップファイルをアーム長補正オプション無効のコントローラーへリストアしないでください。

ArmCalib On 状態のバックアップファイルをアーム長補正オプション無効のコントローラーへリストアすると、自動的に ArmCalib Off 状態になります。別コントローラーへリストアするさいに注意してください。

参照

ArmCalibClr, ArmCalibSet, ArmCalibDef

ArmCalib 使用例

次の例はコマンドウィンドウから実行します。

```
> ArmCalib On  
  
> ArmCalib Off
```

ArmCalib 関数

現在選択されているロボットのアーム長補正の状態を返します。

書式

ArmCalib

戻り値

0 = アーム長補正無効

1 = アーム長補正有効

参照

ArmCalib

ArmCalib 関数使用例

```
If ArmCalib = Off then
    Print "Arm length calibration disabled."
Else
    Print "Arm length calibration enabled."
Endif
```


ArmCalibClr

アーム長補正の設定をクリアします。

書式

ArmCalibClr

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

ArmCalibClr は、必要な時以外、絶対に使用しないでください。

ArmCalibClr は、ArmCalibSet で設定したアーム長補正パラメーターをクリアします。ArmCalibSet は、工場出荷時に精密に設定されています。(アーム長補正オプション購入時) 誤って設定値をクリアすると、再度工場にて精密な測定が必要になります。必要なとき以外は、ArmCalibClr を絶対に使わないでください。

参照

ArmCalib, ArmCalibSet

ArmCalibClr 使用例

ArmCalibClr

ArmCalibDef

アーム長補正の設定状態を返します。

書式

ArmCalibDef

戻り値

アーム長補正が設定されていれば “True”、設定されていなければ “False” を返します。

参照

ArmCalib, ArmCalibClr, ArmCalibSet

ArmCalibDef 使用例

```
Function DisplayArmCalibDef
  Integer i

  If ArmCalibDef = False Then
    Print "ArmCalib is not defined"
  Else
    Print "ArmCalib Definition:"
    For i = 1 to 3
      Print ArmCalibSet(i)
    Next i
  EndIf
Fend
```

ArmCalibSet

アーム長と関節オフセットの設定と表示をします。

書式

- (1) ArmCalibSet 設定値 1, 設定値 2, 設定値 3
- (2) ArmCalibSet

パラメーター

設定値	水平多関節型
1	第 1 関節から第 2 関節までの水平距離 (mm)
2	第 2 関節から姿勢中心までの水平距離 (mm)
3	第 2 関節のオフセット角度 (°)

結果

すべてのパラメーターを省略して ArmCalibSet を実行した場合、アーム長補正番のパラメーターが表示されます。

解説

ArmCalibSet は、アームの長さおよび関節オフセットに関するパラメーターを設定します。工場出荷時に、各個体のアーム長および関節オフセットを精密に測定し、本コマンドで設定することにより距離精度を上げることができます。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行は、必要最小限にとどめることを推奨します。

注意

ArmCalibSet は、必要な時以外、絶対に変更しないでください

ArmCalibSet は、工場出荷時に精密に設定されています。(Arm Length Calibration オプション購入時) 誤って設定値を変更すると、距離精度や軌跡精度が悪化します。必要なとき以外は、ArmCalibSet を絶対に変更しないようにしてください。

参照

ArmCalib, ArmCalibClr, ArmCalibDef

ArmCalibSet 使用例

次の例はコマンドウィンドウから実行します。

```
> ArmCalibSet 299.989, 250.001, 0.012
```

ArmCalibSet 関数

アーム長補正の設定値を 1 つ返します。

書式

ArmCalibSet(設定値番号)

パラメーター

設定値番号 参照する設定値番号(0~3 の整数)を、式または数値で指定します。
(以下を参照してください。)

水平多関節型ロボット

設定値番号	戻り値
1	第 1 関節から第 2 関節までの水平距離 (mm)
2	第 2 関節から姿勢中心までの水平距離 (mm)
3	第 2 関節のオフセット角度 (°)

戻り値

上記表のいずれかを指定したパラメーターの設定値を、実数値で返します。

参照

ArmCalibClr, ArmCalibSet

ArmCalibSet 関数使用例

```
Double L1, L2, Angle  
  
L1 = ArmCalibSet(1)  
L2 = ArmCalibSet(2)  
Angle = ArmCalibSet(3)
```

ArmClr

アームの設定をクリアします。

書式

ArmClr アーム番号

パラメーター

アーム番号 15 個のアームのうち、設定をクリアするアームの番号を整数と式で指定します。
(アーム 0 は、デフォルトアームなので、クリアすることはできません。)

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

Arm, ArmSet, ECPSet, Tool, Local, LocalClr, TLSet

ArmClr 使用例

```
ArmClr 1
```

ArmDef 関数

アームの設定状態を返します。

書式

ArmDef (アーム番号)

パラメーター

アーム番号 状態を返すアームの番号を整数値で指定します。

戻り値

指定したアームが設定されていれば “True”、設定されていなければ “False” を返します。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLClr, TLSet

ArmDef 使用例

```
Function DisplayArmDef (armNum As Integer)

    Integer i

    If ArmDef (armNum) = False Then
        Print "Arm ", armNum, "is not defined"
    Else
        Print "Arm ", armNum, " Definition:"
        For i = 1 to 5
            Print ArmSet (armNum, i)
        Next i
    EndIf
End
```

ArmSet

増設アームの設定と表示をします。

書式

- (1) ArmSet アーム番号, 設定値 1, 設定値 2, 設定値 3 [, 設定値 4] [, 設定値 5]: 水平多関節型(RS シリーズ含む)
- (2) ArmSet アーム番号, 設定値 1, 設定値 2, 設定値 3, 設定値 4, 設定値 5, 設定値 6, 設定値 7, 設定値 8, 設定値 9, 設定値 10, 設定値 11, 設定値 12: 垂直多関節型(N シリーズを含む)
- (3) ArmSet アーム番号
- (4) ArmSet

パラメーター

アーム番号 1~15 の整数を式または数値で指定します。増設アームは、15 個まで設定することができます。

設定値 番号	水平多関節型 (RS シリーズを含む)	垂直多関節型 (N シリーズを含む)
1	第 2 関節から姿勢中心までの水平距離 (mm)	ベースから第 2 関節までの垂直距離(mm)
2	第 2 関節のオフセット角度 (度)	第 1 関節から第 2 関節までの水平距離(mm)
3	作業先端の高さ方向のオフセット (mm)	第 2 関節から第 3 関節までの距離(mm)
4	第 1 関節から第 2 関節までの水平距離 (mm)	第 3 関節から第 5 関節までの垂直距離(mm)
5	第 4 関節角度のオフセット (度)	第 3 関節から第 5 関節までの水平距離(mm)
6	-	第 5 関節から姿勢中心までの距離(mm)
7	-	第 1 関節角度のオフセット(度)
8	-	第 2 関節角度のオフセット(度)
9	-	第 3 関節角度のオフセット(度)
10	-	第 4 関節角度のオフセット(度)
11	-	第 5 関節角度のオフセット(度)
12	-	第 6 関節角度のオフセット(度)

結果

すべてのパラメーターを省略して ArmSet を実行した場合、設定されているすべての増設アームのアーム番号と、パラメーターが表示されます。

アーム番号だけを指定した場合、指定したアーム番号と、パラメーターが表示されます。

解説

標準セットアップに、アームを増設したとき、その増設アームに関するパラメーターを設定します。ロボットに増設アームや増設ハンドが取り付けられたときに便利です。増設アームを使うときは、Arm 命令により作業するアームを指定します。

設定値 4 の第 1 関節~第 2 関節までの水平距離と、設定値 5 の姿勢関節角度のオフセットは、省略可能で、省略すると標準アームと同一に設定されます。

標準以外の構成でロボットを使いたい場合に、それに応じたパラメーター設定を可能にするのが、増設アームの設定機能です。例えば、第 2 ロボットに第 2 回転関節を取りつけた場合は、新たに構成されたロボットアームの設定を行わなければいけません。この設定を行って、増設アームは適切な動作をすることができますが、その場合、下記の条件が満たされなくてはなりません。

- 同一のデータポイントを複数のアームで使用

- Pallet を使う
- CP 動作を使う
- 相対位置指定を行う
- ローカル座標系を使用

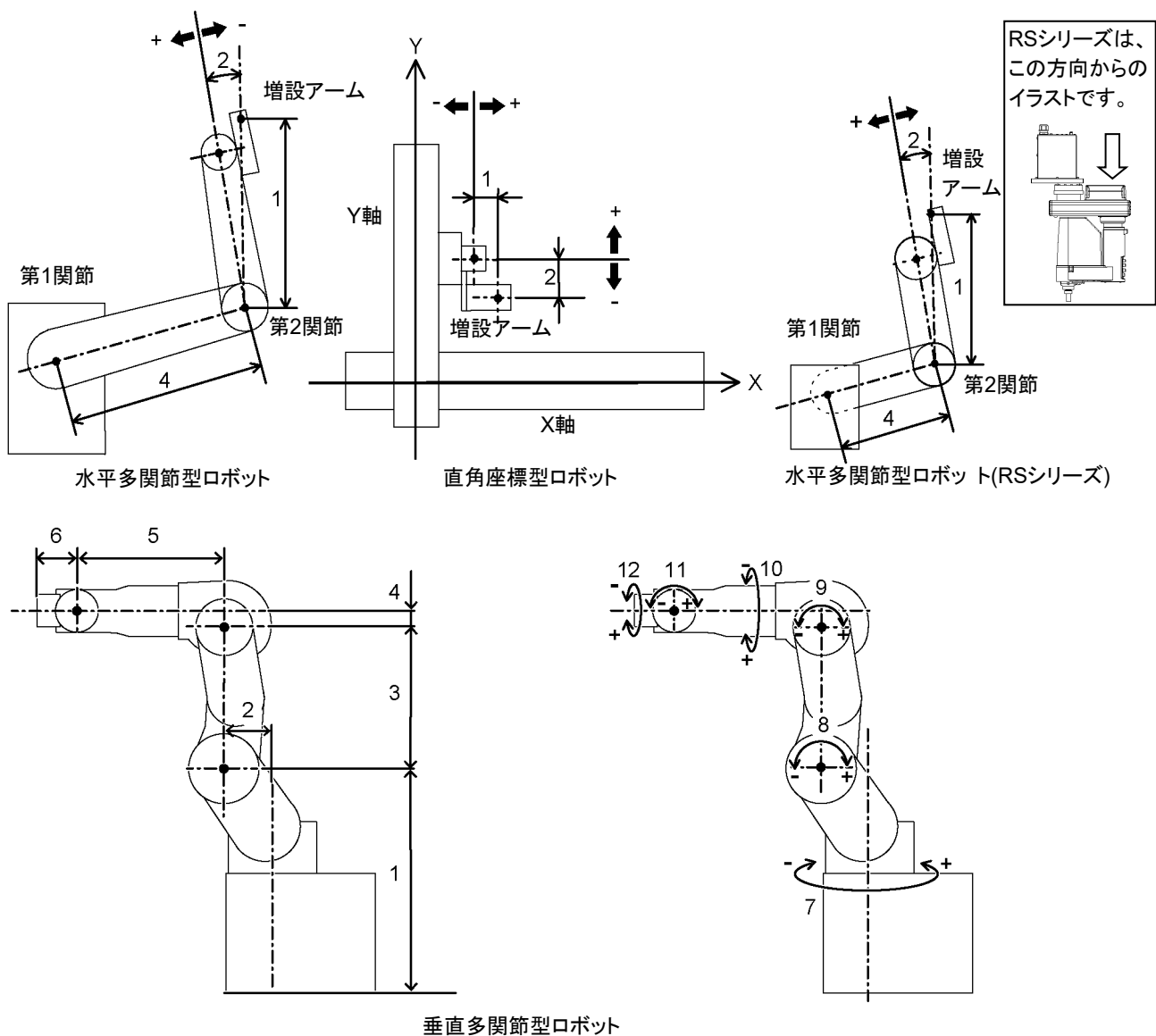
回転関節を持つ水平多関節型 (RS シリーズを含む) では、関節角度の演算は ArmSet のパラメーターに基づいて行われるので、増設アームやハンドの設定が必要な場合には、この設定が重要です。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

Arm 0

アーム番号 0 は、ロボットの標準構成時の設定に使われるため、ユーザーが定義したり変更することはできません。アーム番号 0 を設定すると、ロボットアームの標準パラメーターが使われます。



参照

Arm, ArmClr

ArmSet 使用例

下記は ArmSet と Arm を使った増設アームの設定例です。ArmSet で増設アームを設定し、Arm でどのアームを選択するか設定します。(アーム番号 0 は、ロボットのデフォルト設定アームで、ユーザーが変更することはできません。)

コマンドウィンドウでの操作例

```
> ArmSet 1, 300, -12, -30, 300, 0
> ArmSet
  Arm 0: 125.000, 0.000, 0.000, 225.000, 0.000
  Arm 1: 300.000, -12.000, -30.000, 300.000, 0.000

> Arm 0
> Jump P1      '標準アームの設定で P1 へ Jump
> Arm 1
> Jump P1      '増設アーム1の設定で P1 へ Jump
```

ArmSet 関数

増設アームの設定値を1つ返します。

書式

ArmSet(アーム番号, 設定値番号)

パラメーター

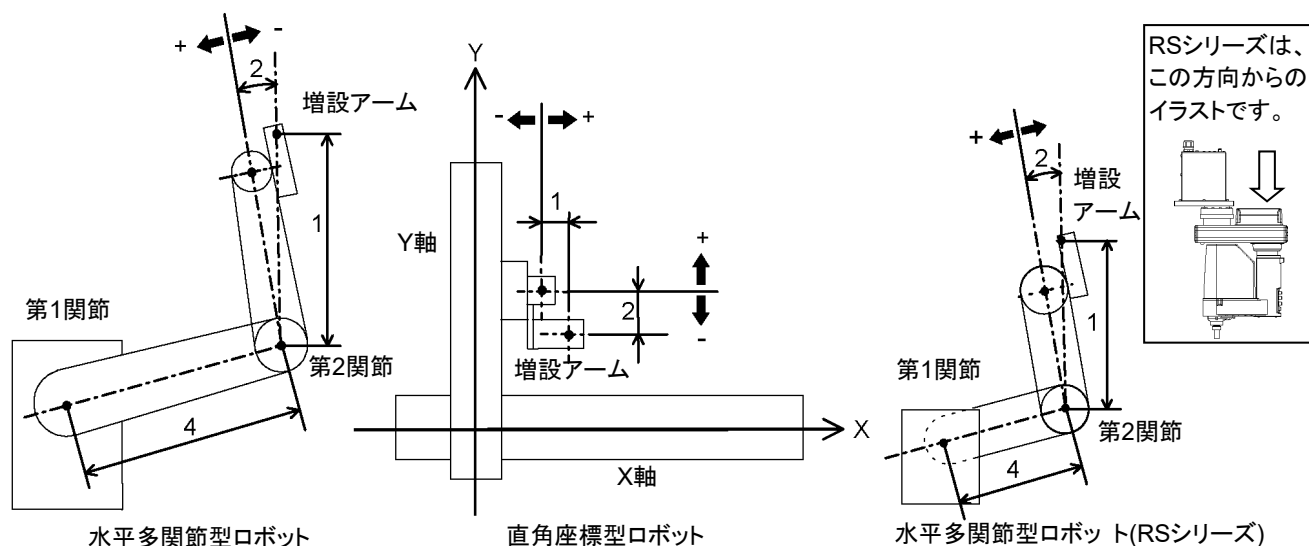
アーム番号 参照するアーム番号を、式または数値で指定します。

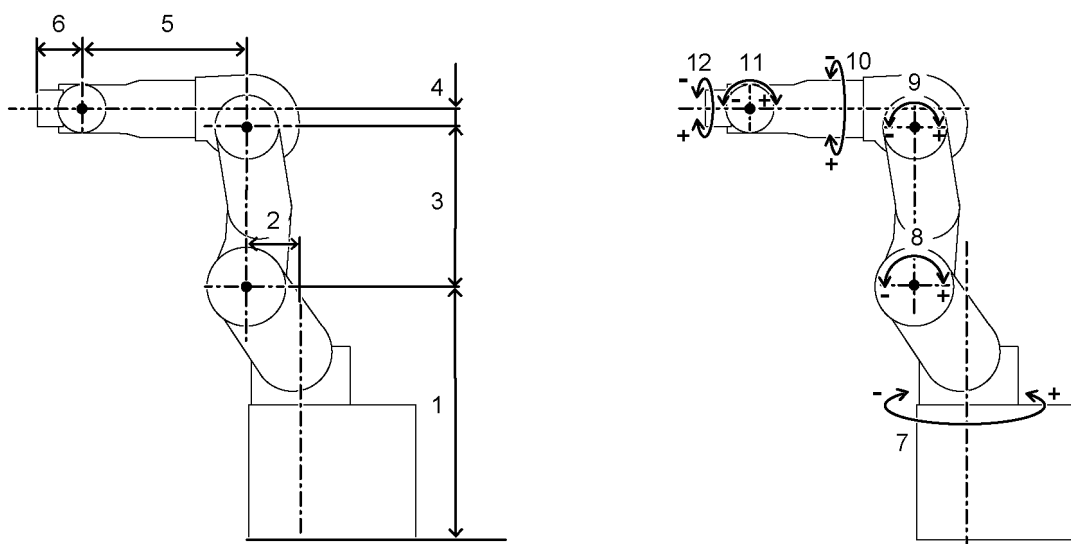
設定値番号 参照する設定値番号(0~5の整数)を、式または数値で指定します。
(下記を参照してください。)

設定値番号	水平多関節型 (RSシリーズを含む)	垂直多関節型 (Nシリーズを含む)
1	第2関節から姿勢中心までの水平距離(mm)	ベースから第2関節までの垂直距離(mm)
2	第2関節のオフセット角度(度)	第1関節から第2関節までの水平距離(mm)
3	作業先端の高さ方向のオフセット(mm)	第2関節から第3関節までの距離(mm)
4	第1関節から第2関節までの水平距離(mm)	第3関節から第5関節までの垂直距離(mm)
5	第4関節角度のオフセット(度)	第3関節から第5関節までの水平距離(mm)
6	-	第5関節から姿勢中心までの距離(mm)
7	-	第1関節角度のオフセット(度)
8	-	第2関節角度のオフセット(度)
9	-	第3関節角度のオフセット(度)
10	-	第4関節角度のオフセット(度)
11	-	第5関節角度のオフセット(度)
12	-	第6関節角度のオフセット(度)

戻り値

上記表のいずれかを指定したパラメーターの設定値を、実数値で返します。





垂直多関節型ロボット

参照

ArmClr, ArmSet

ArmSet 関数使用例

Real x

x = **ArmSet**(1, 1)

Asc 関数

文字列の先頭の第 1 文字に対する ASCII コードを返します。(キャラクターコードを 10 進数で返します。)

書式

Asc(置換文字列)

パラメーター

置換文字列 1 文字以上の文字列を、文字列式または直接文字列で指定します。

戻り値

Asc 関数に送られた文字列の、第 1 番目の文字に対する整数値を返します。

解説

Asc 関数は、文字列の先頭の 1 文字のキャラクターコードを 10 進数で返します。Asc 関数に送られる文字列は、定数でも変数でもかまいません。

注意

ASCII 値が返されるのは、第 1 文字についてだけです

Asc 関数では 1 文字以上の長さがある文字列を使うことができますが、実際には、その第 1 文字だけが使われます。Asc は、文字列の第 1 文字の ASCII 値だけを返します。

参照

Chr\$, InStr, Left\$, Len, Mid\$, Right\$, Space\$, Str\$, Val

Asc 関数使用例

この例では、プログラム中とコマンドウィンドウからの操作で、次のような Asc 関数を使っています。

```
Function asctest
    Integer a, b, c
    a = Asc("a")
    b = Asc("b")
    c = Asc("c")
    Print "The ASCII value of a is ", a
    Print "The ASCII value of b is ", b
    Print "The ASCII value of c is ", c
Fend
```

コマンドウィンドウでの操作例

```
>print asc("a")
97
>print asc("b")
98
>
```

Asin 関数

指定数値のアークサインを返します。

書式

Asin(数値)

パラメーター

数値 角度のサインを実数値で指定します。

戻り値

指定した数値のアークサイン値を実数値 (単位: ラジアン) で返します。

解説

Asin は、指定された数値のアークサインを返します。指定する数値は、 -1 ~ 1 の範囲です。戻り値の範囲は、 $-\pi/2 \sim \pi/2$ です。数値が -1 より小さい、または 1 より大きい場合は、エラーが発生します。

ラジアン値を、“度”に変換するためには、RadToDeg 関数を使います。

参照

Abs, Acos, Atan, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Asin 関数使用例

```
Function asintest
    Double x

    x = Sin(DegToRad(45))
    Print "Asin of ", x, " is ", Asin(x)
End
```

AtHome 関数

現在のロボット姿勢が Home 位置か否かを返します。

書式

AtHome

戻り値

現在のロボット姿勢が Home 位置上なら“True”、Home 位置以外なら“False”を返します。

解説

本関数は、現在のロボット姿勢が Home 位置か否かを返します。Home 位置の登録には HomeSet 命令、またはロボットマネージャーを使用します。Home 位置への移動には Home 命令を使用します。

参照

Home, HomeClr, HomeDef, HomeSet, Hordr, MCalComplete

Atan 関数

指定数値のアーктanジェントを返します。

書式

Atan(数値)

パラメーター

数値 角度のtanジェントを実数値で指定します。

戻り値

指定した数値のアーктanジェント値を実数値 (単位: ラジアン) で返します。

解説

Atan は、指定された数値のアークトanジェントを返します。指定する数値は、どんな数値でもかまいません。戻り値の範囲は、 $-\pi \sim \pi$ ラジアンです。

ラジアン値を、“度”に変換するためには、RadToDeg 関数を使います。

参照

Abs, Acos, Asin, Atan2, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Atan 関数使用例

```
Function atantest
  Real x, y
  x = 0
  y = 1
  Print "Atan of ", x, " is ", Atan(x)
  Print "Atan of ", y, " is ", Atan(y)
End
```

Atan2 関数

座標原点と指定点 (X, Y)を結ぶ直線の角度を返す関数です。単位はラジアンです。

書式

Atan2(X 座標値, Y 座標値)

パラメーター

X X 座標値を実数値で指定します。

Y Y 座標値を実数値で指定します。

戻り値

指定した座標のアーктanジェント値を返します。範囲は $-\pi \sim \pi$ です。

解説

Atan2 (X, Y) は、座標原点 (0, 0)と([x 座標値],[y 座標値])とを結ぶ直線の角度を返す関数です。
この三角関数はすべての 4 つの四分円でアーктanジェント値 (角度)を返します。

参照

Abs, Acos, Asin, Atan, Cos, DegToRad, RadToDeg, Sgn, Sin, Tan, Val

Atan2 関数使用例

```
Function at2test
  Real x, y
  Print "Please enter a number for the X Coordinate:"
  Input x
  Print "Please enter a number for the Y Coordinate:"
  Input y
  Print "Atan2 of ", x, ", ", y, " is ", Atan2(x, y)
Fend
```


ATCLR

関節の実効トルクをクリアし、初期化します。

書式

ATCLR [関節指定 1 [, 関節指定 2 [, 関節指定 3 [, 関節指定 4 [, 関節指定 5 [, 関節指定 6
[, 関節指定 7 [, 関節指定 8 [, 関節指定 9]]]]]]]]]

パラメーター

関節指定 1 -関節指定 9 関節番号を整数値または式で指定します。パラメーターを指定しない場合は、全関節の実効トルク値がクリアされます。

付加軸の S 軸は 8、T 軸は 9 になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

ATCLR は指定した関節の実効トルク値をクリアします。
ATRQ 実行前に、必ず ATCLR を実行してください。

参照

ATRQ, PTRQ

ATCLR 使用例

<例 1>

全関節の実効トルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

```
> atclr
> go pl
> atrq 1
    0.028
> atrq
    0.028      0.008
    0.029      0.009
    0.000      0.000
>
```

<例 2>

垂直多関節ロボットにおいて、J1, J4, J5 の実効トルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

```
> atclr 4, 1, 5
> go pl
> ptrq 1
    0.227
> ptrq 4
    0.083
```

ATRQ

指定した関節の実効トルク値を表示します。

書式

ATRQ [関節番号]

パラメーター

関節番号 関節番号を整数値または式で指定します。省略可能です。
付加軸の S 軸は 8、T 軸は 9 になります。

結果

全関節について現在の実効トルク値を表示します。

解説

ATRQ は、指定した関節の実効トルク値を表示します。ATRQ 命令で、モーターの負荷状態がわかります。結果は 0~1 までの実数値で表わされます。最大実効トルク値は “1” になります。

ATRQ 実行前に、必ず ATCLR を実行してください。

ATRQ 命令には、時間制限があります。ATCLR 実行後、60 秒以内に ATRQ を実行してください。60 秒を超えると、エラー 4030 “トルク実行値飽和状態”が発生します。

参照

ATCLR, ATRQ 関数, PTRQ

ATRQ 使用例

```
> atclr
> go pl
> atrq 1
    0.028
> atrq
    0.028      0.008
    0.029      0.009
    0.000      0.000
>
```

ATRQ 関数

指定した関節の実効トルクを返します。

書式

ATRQ (関節番号)

パラメーター

関節番号 関節番号を整数値または式で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

0~1 までの実数値で返します。

解説

ATRQ 関数は、指定した関節の実効トルクを返します。ATRQ 関数で、モーターの負荷状態がわかります。結果は 0~1 までの実数値で表わされます。最大実効トルクは 1 になります。

ATRQ 関数実行前に、必ず ATCLR を実行してください。

ATRQ 関数には、時間制限があります。ATCLR 実行後、60 秒以内に ATRQ 関数を実行してください。60 秒を超えると、エラー4030が発生します。

参照

ATRQ, PTCLR, PTRQ

ATRQ 関数使用例

プログラム中に ATRQ 関数を使用した例です。

```
Function CheckAvgTorque
  Integer i

  Go P1
  ATCLR
  Go P2
  Print "Average torques:"
  For i = 1 To 4
    Print "Joint ", i, " = ", ATRQ(i)
  Next i
End
```

AutoLJM

自動 LJM を設定します。

書式

AutoLJM { On | Off }

パラメーター

On | Off On: 自動 LJM を有効にします。
Off: 自動 LJM を解除します。

解説

AutoLJM は下記の命令で有効です。

Arc, Arc3, Go, Jump3, Jump3CP, Move

AutoLJM On すると、各動作命令に渡す位置データに LJM 関数を適用するかしないかにかかわらず、LJM 関数を適用したときと同様に関節移動量が最小になるように動作を実行します。

例えば Go LJM(P1)に対して、同じ効果を得るため

AutoLJM On

Go P1

AutoLJM Off

とすることができます。AutoLJM はプログラムの特定の区間で LJM を有効にできるため、個々の動作命令を変更する必要がなくなります。

AutoLJM Off すると、各動作命令に渡す位置データに LJM 関数を適用した場合にのみ、LJM 関数の機能は有効になります。

下記の場合、AutoLJM はコントローラー設定で指定した設定状態 (工場出荷時: Off)になります。

コントローラー起動時 Reset 実行時 全タスク中断時 Motor On 実行時 Auto / Programming 作業モード切替時

注意

AutoLJM と LJM 関数の 2 重適用について

AutoLJM On 時に動作コマンドに渡すポイントデータに LJM 関数が適用されていると、動作実行時に LJM が 2 重に適用されることとなります。

Move LJM(P1, Here)や Move LJM(P1)という動作命令に対しては AutoLJM を有効にしたときと、しなかったときで動作は変わりません。しかし、Move LJM(P1, P0)という動作命令に対し AutoLJM を有効にしたときの Move LJM(LJM(P1, P0), Here)の動作と、AutoLJM を有効にしなかったときの Move LJM(P1, P0)という動作では、動作完了位置が異なることがあります。

AutoLJM と LJM 関数が 2 重に適用されないようプログラムすることをお勧めします。

AutoLJM 使用上の注意

コントローラーの環境設定により、コントローラー起動時に AutoLJM 機能をオンすることができます。ただし、コントローラー環境設定やコマンドで AutoLJM 機能を常時有効にしていると、お客様が意図して関節を大きく移動しようとした動作命令に対しても自動的に関節移動量が少なくなる姿勢に変更して動作してしまいます。

LJM 関数や AutoLJM コマンドを使い、必要なときのみ LJM を適用するようプログラムすることをお勧めします。

参照

AuoLJM 関数, LJM 関数

AutoLJM 使用例

```
AutoLJM On
Go P1
Go P2
AutoLJM Off
```

AutoLJM 関数

AutoLJM の状態を返します。

書式

AutoLJM

戻り値

0 = 自動 LJM オフ

1 = 自動 LJM オン

参照

AutoLJM

AutoLJM 関数使用例

```
If AutoLJM = Off Then  
    Print "AutoLJM is off"  
EndIf
```

AutoOrientationFlag

N6-A1000**の姿勢フラグを変更します。

書式

AutoOrientationFlag { On | Off }

パラメーター

On | Off On: AutoOrientationFlag を有効にします。
Off: AutoOrientationFlag を解除します。(デフォルト)

解説

AutoOrientationFlag は、下記の命令で有効です。
Go, BGo, TGo, Jump3, JumpTLZ

下記の姿勢フラグを変更します。

対象機種	パラメーター OFF/ON	姿勢フラグ			備考
		腕	肘	手首	
N6-A1000**	OFF	-	-	-	ユーザーが指定した姿勢フラグで動作します。 (デフォルト)
	ON	-	○	○ *1	姿勢フラグに迷った場合は、 “ON”を選択してください。

○: AutoOrientationFlag を“ON”に設定すると、姿勢フラグが変更されます。

*1: 手首姿勢フラグは、肘姿勢フラグを変更する場合のみ変更されます。手首姿勢フラグを変更する場合、J4軸の移動量が最少となる姿勢フラグとなります。

LJM 関数との併用

LJM 関数を併用して使用した場合、手首姿勢 Flag, J4Flag, J6Flag は、LJM 関数で選択した姿勢になります。例えば、LJM 関数の姿勢フラグ選択を“3”にした場合、J5 軸の移動量が最少移動量となる手首姿勢 Flag, J4Flag, J6Flag を選択します。また、LJM 関数を併用しなかった場合、J4 軸が最少移動量となる手首姿勢 Flag, J4Flag, J6Flag を選択します。

AutoOrientationFlag 使用例

```
Motor On
Power High
AutoOrientationFlag On
```

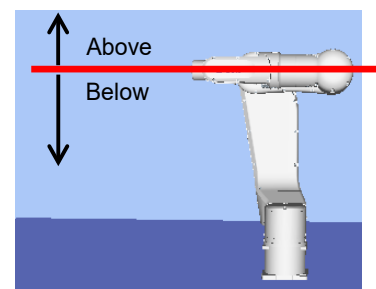
```
Go P1
Go P2
```

AutoOrientationFlagを“ON”にした場合:

P点と赤線との位置の関係により、次のようにフラグが変更されます。

P点が赤線より上の位置: Above

P点が赤線より下の位置: Below



AutoOrientationFlag 関数

AutoOrientationFlag の状態を返します。

書式

AutoOrientationFlag

戻り値

0 = AutoOrientationFlag オフ

1 = AutoOrientationFlag オン

参照

AutoOrientationFlag

AutoOrientationFlag 関数使用例

```
If AutoOrientationFlag = Off Then  
    Print " AutoOrientationFlag is off"  
EndIf
```


AvgSpeedClear

関節速度の絶対値の平均をクリアし、初期化します。

書式

AvgSpeedClear [関節指定 1 [, 関節指定 2 [, 関節指定 3 [, 関節指定 4 [, 関節指定 5 [, 関節指定 6 [, 関節指定 7 [, 関節指定 8 [, 関節指定 9]]]]]]]]]

パラメーター

関節指定 1-関節指定 9 関節番号を整数値、または式で指定します。パラメーターを指定しない場合は、全関節の速度の絶対値の平均がクリアされます。

付加軸の S 軸は 8、T 軸は 9 になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

AvgSpeedClear は指定した関節の速度の絶対値の平均をクリアします。

AvgSpeed 実行前に、必ず AvgSpeedClear を実行してください。

本コマンドは、PG 付加軸はサポートしません。

参照

AvgSpeed, PeakSpeed

AvgSpeedClear 使用例

<例 1>

全関節の平均速度値をクリアしたのち、指定した関節番号の平均速度値を表示させるコマンド実行例です。

```
> AvgSpeedClear
> Go P1
> AvgSpeed 1
    0.073
> AvgSpeed
    0.073      0.044
    0.021      0.069
    0.001      0.108
    0.000      0.000
    0.000
```

```
>
```

<例 2>

垂直多関節ロボットにおいて、第 1 関節、第 4 関節、第 5 関節の平均速度値をクリアしたのち、指定した関節番号の平均速度値を表示させるコマンド実行例です。

```
> AvgSpeedClear 4, 1, 5
> Go P1
> AvgSpeed 1
    0.226
> AvgSpeed 4
    0.207
```

AvgSpeed

指定した関節の速度の絶対値の平均値を表示します。

書式

AvgSpeed [関節番号]

パラメーター

関節番号 関節番号を整数値、または式で指定します。省略可能です。
付加軸の S 軸は 8、T 軸は 9 になります。

結果

指定した関節について、現在の速度の絶対値の平均を表示します。指定がない場合は、全関節の速度の絶対値の平均を表示します。

解説

AvgSpeed は、指定した関節の速度の絶対値の平均値を表示します。AvgSpeed 命令で、モーターの負荷状態がわかります。結果は 0~1 までの実数値で表わされます。最大平均速度値は “1” になります。平均値が 0.001 以下の値になる場合は “0” になります。

AvgSpeed 実行前に、必ず AvgSpeedClear を実行してください。

AvgSpeed 命令には、時間制限があります。AvgSpeedClear 実行後、60 秒以内に AvgSpeed を実行してください。60 秒を超えると、エラー 4088 が発生します。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度から速度の絶対値の平均を計算します。

本コマンドは、PG 付加軸はサポートしません。

参照

AvgSpeedClear, AvgSpeed 関数, PeakSpeed

AvgSpeed 使用例

```
> AvgSpeedClear
> Go P1
> AvgSpeed 1
    0.226
> AvgSpeed
    0.226      0.133
    0.064      0.207
    0.003      0.314
    0.000      0.000
    0.000
>
```

AvgSpeed 関数

指定した関節の速度の絶対値の平均値を返します。

書式

AvgSpeed (関節番号)

パラメーター

関節番号 関節番号を整数値、または式で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

0~1 までの実数値で返します。

解説

AvgSpeed 関数は、指定した関節の速度の絶対値の平均値を返します。AvgSpeed 関数で、モーターの負荷状態がわかります。結果は 0~1 までの実数値で表わされます。最大平均速度は 1 になります。

AvgSpeed 関数実行前に、必ず AvgSpeedClear を実行してください。

AvgSpeed 関数には、時間制限があります。AvgSpeed 実行後、60 秒以内に AvgSpeed 関数を実行してください。60 秒を超えると、エラー4088が発生します。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度から速度の絶対値の平均を計算します。

本関数は、PG 付加軸はサポートしません。

参照

AvgSpeed, AvgSpeedClear, PeakSpeed

AvgSpeed 関数使用例

プログラム中に AvgSpeed 関数を使用した例です。

```
Function CheckAvgSpeed
  Integer i

  Go P1
  AvgSpeedClear
  Go P2
  Print "Average speeds:"
  For i = 1 To 6
    Print "Joint ", i, " = ", AvgSpeed (i)
  Next i
End
```

AvoidSingularity

特異姿勢通過機能を設定します。

書式

AvoidSingularity { mode }

パラメーター

mode 使用する特異点通過モードを表す整数式

定数	値	モード
SING_NONE	0	特異姿勢通過機能を無効にします。
SING_THRU	1	特異姿勢通過機能を有効にします。
SING_THRUROT	2	ROT 修飾詞付きの CP 動作で特異姿勢通過機能を有効にします。
SING_VSD	3	可変速度 CP 動作機能を有効にします。
SING_AUTO	4	特異姿勢通過機能あるいは可変速度 CP 動作機能を自動選択します。
SING_AVOID	5	肘特異姿勢通過機能を有効にします。

解説

AvoidSingularity は下記の命令で有効です。

Move, Arc, Arc3, Jump3, Jump3CP, JumpTLZ

特異姿勢通過機能は、垂直 6 軸ロボット (N シリーズを含む)、および RS シリーズロボットが CP 動作実行中に特異姿勢に近づいたとき、加速度エラーを回避するために、速度は維持したまま、本来の軌跡とは異なる軌跡を通過し、特異姿勢から離れた後、通常の軌跡に戻すことができます。コントローラー起動時特異姿勢回避機能は“1: 有効”となっていますので、通常は変更する必要はありませんが、特異姿勢通過機能に対応していないソフトウェアとの互換性確保や特異姿勢回避動作による軌跡のずれを嫌う場合など、特異姿勢回避をさせたくない場合に無効にしてください。

可変速度 CP 動作機能は、垂直 6 軸ロボット (N シリーズを含む)、および RS シリーズロボットが CP 動作実行中に特異姿勢に近づいたとき、加速度エラーや過速度エラーを回避するために、軌跡は維持したまま、速度を自動的に抑制し、特異姿勢から離れた後、通常の数値指令に戻すことができます。軌跡を維持して特異点近傍を通過するため、第 1, 第 2, 第 4, 第 6 関節が大きく動作することがあります。

AvoidSingularity の設定値を変更した場合、次回コントローラー起動時まで有効です。

コントローラー起動時は、AvoidSingularity はコントローラー設定で指定した設定状態 (工場出荷時: 1) になります。また、AvoidSingularity の設定値を変更した場合、SingularityAngle, SingularitySpeed, SingularityDist の各パラメーターはデフォルト値に戻ります。

SING_AUTO は、SING_THRU と SING_VSD を合わせたモードです。動作または速度によって SING_THRU か SING_VSD が選択されます。

注意

垂直 6 軸ロボットと N シリーズロボットの特異姿勢近傍の条件設定

ロボットが手首特異姿勢近傍に近づいたかどうかを判断するため、第 5 関節の角度および第 4 関節の角速度を用いています。デフォルトでは第 5 関節角度は ± 10 度、第 4 関節角度は最大関節速度の $\pm 10\%$ に設定されています。これらを変更する場合、SingularityAngle および SingularitySpeed コマンドを使用します。また、腕特異姿勢に近づいたかどうかを判断するため、P 点の座標を用います。デフォルトでは、P 点とロボットの第 1 関節の回転軸の距離が 30 mm に設定されています。これを変更する場合、SingularityDist コマンドを使用します。

RS シリーズロボットの特異姿勢近傍の条件設定

ロボットが腕特異姿勢に近づいたかどうかを判断するため、デフォルトツール 0 座標系原点の座標を用います。デフォルトでは、ツール 0 座標系の原点とロボットの第 1 関節の回転軸の距離が 30 mm に設定されています。これを変更する場合、SingularityDist コマンドを使用します。

N シリーズロボットの注意事項

N2 シリーズは他の機種と異なり、コントローラー起動時の特異姿勢回避機能は“3: 可変速度 CP 動作機能を有効”に設定されています。

N6 シリーズは他の機種と同じく、コントローラー起動時の特異姿勢回避機能は“1: 特異姿勢通過機能を有効”に設定されています。

N シリーズは、手首特異点姿勢と腕特異点姿勢の他に、肘特異姿勢領域を持ちます。

肘特異姿勢領域は、第 3 関節が 0 度のとき (第 3 関節と第 2 関節が重なったとき)の姿勢です。

肘特異姿勢領域の通過動作についての詳細は、EPSON RC+ ユーザーズガイドを参照してください。

SING_THRU と SING_AVOID の違い

SING_THRU は、手首特異姿勢と肩特異姿勢を通過しますが、肘特異姿勢は通過しません。肘特異姿勢を通過させる場合、SING_AVOID を選択してください。ただし、肘特異姿勢通過は、他の特異姿勢通過に比べ大きく軌道が変わるため、注意して使用してください。また、N シリーズ以外で SING_AVOID を選択した場合、エラー 4002 が発生します。

参照

AvoidSingularity 関数, SingularityAngle, SingularitySpeed, SingularityDist

AvoidSingularity 使用例

```
AvoidSingularity 0 '特異姿勢回避を無効にして動作させる
Move P1
Move P2
AvoidSingularity 1
```

AvoidSingularity 関数

AvoidSingularity の状態を返します。

書式

AvoidSingularity

戻り値

- 0 = 特異姿勢通過機能無効
- 1 = 特異姿勢通過機能有効
- 2 = ROT 修飾詞付きの CP 動作命令で特異姿勢通過機能有効
- 3 = 可変速度 CP 動作機能有効
- 4 = 特異姿勢通過機能と可変速度 CP 動作機能の自動選択
- 5 = 肘特異姿勢通過機能有効

参照

AvoidSingularity

AvoidSingularity 関数使用例

```
If AvoidSingularity = Off Then  
    Print "AvoidSingularity is off"  
EndIf
```

Base

ベース座標系の定義と表示をします。

書式

- (1) Base 座標系データ
- (2) Base 原点, X 軸指定, Y 軸指定 [, { X | Y }]

パラメーター

座標系データ	ベース座標系の原点と向きをポイントデータで直接指定します。
原点	ベース座標系原点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。
X 軸指定	ベース座標系の X 軸上の点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。
Y 軸指定	ベース座標系の Y 軸上の点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。
X	X 軸指定を優先的に X 軸に一致させます。省略可能です。(デフォルト)
Y	Y 軸指定を優先的に Y 軸に一致させます。省略可能です。

解説

ロボットには、変更することのできない基準座標系があり、これを“ロボット座標系”とよびます。これに対し、一般のローカル座標系の基礎になり、原点座標を変更できる基本座標系を“ベース座標系”とよびます。

ロボット座標系に対するローカル座標系の基点と回転角を設定することにより、ローカル座標系を定義します。

ベース座標系をデフォルト値にリセットするには、次のステートメントを実行してください。これを実行すると、ベース座標系は、ロボット座標系と同じになります。

```
Base XY(0, 0, 0, 0)
```

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

ベース座標系を変更すると、すべてのローカル定義に影響があります。

ベース座標系を変更したら、ローカル座標系をすべて定義し直してください。

参照

Local

Base 使用例

ベース座標系の原点を X 軸 100mm、Y 軸 100mm 地点に定義します。

```
> Base XY(100, 100, 0, 0)
```

BClr 関数

指定した数値の 1 ビットをクリアし、その値を返します。

書式

BClr (数値, ビット番号)

パラメーター

数値 ビットをクリアする数値を、式または直接数値で指定します。

ビット番号 クリアするビット番号 (0~31 の整数値) を、式または直接数値で指定します。

戻り値

ビットをクリアした値 (整数) を返します。

参照

BClr64, BSet, BSet64, BTst, BTst64

BClr 関数使用例

```
flags = BClr(flags, 1)
```


BClr64 関数

指定した数値の 1 ビットをクリアし、その値を返します。

書式

BClr64 (数値, ビット番号)

パラメーター

数値 ビットをクリアする数値を、式または直接数値で指定します。

ビット番号 クリアするビット番号 (0~63 の整数値) を、式または直接数値で指定します。

戻り値

ビットをクリアした値 (整数) を返します。

参照

BClr, BSet, BSet64, BTst, BTst64

BClr64 関数使用例

```
flags = BClr64(flags, 1)
```

BGo

選択したローカル座標系での、オフセット PTP 動作を実行します。

書式

BGo 目標座標 [CP] [Till | Find] [!並列処理!] [SYNC]

パラメーター

目標座標	ポイントデータを用いて動作の目標位置を指定します。
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
!並列処理!	動作中に、I/O その他のコマンドを実行するよう、並列処理ステートメントを付け加えることができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

選択されているローカル座標系での、オフセット PTP 動作を実行します。目標座標を示すポイントデータで指定された座標系を基準としてオフセット PTP 動作を行います。

ローカル座標系を指定しないときは、ローカル 0 (ベース座標系)を基準として、オフセット PTP 動作を行います。

ポイントデータで与えられた姿勢フラグは無視され、現在の姿勢フラグが維持されます。ただし、垂直 6 軸型ロボット (N シリーズを含む)では、姿勢フラグを関節移動量が小さくなるよう自動的に変更します。これは、Move 命令において LJM 修飾パラメーターを指定した場合と同等です。このため、180 度以上の姿勢変化を行わせたい場合、複数回に分けて実行してください。

Till 修飾子を使って、Till 条件成立時にロボットを動作途中で減速、停止して、BGo 動作を完了することができます。

Find 修飾子を使って、動作中に Find 条件が真になったときに、ポイントを FindPos に保存します。

!並列処理!を使って、動作と並列に他の処理を実行することができます。

CP パラメーターをつけた場合には、動作の減速開始時に次の動作命令の加速を重ね合わせることが可能です。この場合には、目標座標には位置決めされません。

参照

Accel, BMove, Find, !並列処理!, P# = ポイント指定, Speed, Till, TGo, TMove, Tool

BGo 使用例

> **BGo** XY(100, 0, 0, 0) '(ローカル座標系で)X 方向に 100mm 移動

Function BGoTest

```
Speed 50
Accel 50, 50
Power High

P1 = XY(300, 300, -20, 0)
P2 = XY(300, 300, -20, 0) /L
Local 1, XY(0, 0, 0, 45)
```

```
GoP1
Print Here
BGo XY(0, 50, 0, 0)
Print Here
```

```
Go P2
Print Here
BGo XY(0, 50, 0, 0)
Print Here
```

```
BGo XY(0, 50, 0, 0) /1
Print Here
```

Fend

[出力結果]

X:	300.000	Y:	300.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/R	/0
X:	300.000	Y:	350.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/R	/0
X:	300.000	Y:	300.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/L	/0
X:	300.000	Y:	350.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/L	/0
X:	264.645	Y:	385.355	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/L	/0

BMove

選択したローカル座標系で、オフセット直線補間動作を実行します。

書式

BMove 目標座標 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]

パラメーター

目標座標	ポイントデータを用いて動作の目標位置を指定します。
ROT	ツール姿勢変化を優先させて、速度と加減速度を指定します。省略可能です。
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
! 並列処理 !	動作中に I/O その他のコマンドを実行するために、並列処理ステートメントをつけ加えることができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

選択したローカル座標系で、オフセット直線補間動作を実行します。目標座標を示すポイントデータで指定された座標系を基準としてオフセット直線補間動作を行います。

ローカル座標系を指定しないときは、ローカル 0 (ベース座標系)を基準として、オフセット PTP 動作を行います。

ポイントデータで与えられた姿勢フラグは無視され、現在の姿勢フラグが維持されます。ただし、垂直 6 軸型ロボット (N シリーズを含む)では、姿勢フラグを関節移動量が小さくなるよう自動的に変更します。これは、Move 命令において LJM 修飾パラメーターを指定した場合と同等です。このため、180 度以上の姿勢変化を行わせたい場合、複数回に分けて実行してください。

BMove の速度と加減速度はそれぞれ SpeedS と AccelS の設定値を使用します。速度と加減速度の関係については、“注意”の“BMove とともに CP を用いる”を参照してください。ただし、ROT 修飾パラメーターを指定しているときの速度と加減速度は、それぞれ SpeedR と AccelR の設定値を使用します。この場合、SpeedS と AccelS の設定値は無効となります。

通常、移動距離が“0”で、姿勢のみが変化するような場合にはエラーとなります。ROT 修飾パラメーターを付けてツール姿勢変化の加減速を優先することで、エラーなく動作が可能になります。ROT 修飾パラメーターを付けたとき、姿勢変化がなく、移動距離が“0”でない場合には、エラーとなります。

また、移動距離に対してツール姿勢変化速度が大きすぎる場合や、指定された回転速度がマニピュレーターの限度を越えている場合もエラーとなります。その場合は指定速度を落とすか、または ROT 修飾パラメーターを付けて姿勢変化の加減速度を優先してください。

Till 修飾子を用いることで、Till 条件成立時に、動作途中でロボットを減速停止させて、BMove を完了することができます。

Find 修飾子を用いることで、動作中に Find 条件の値が真(True)になったとき、ポイントデータを FindPos に保存します。

!並列処理!を使って、動作と並列に他の処理を実行することができます。

注意**BMove を CP とともに用いる**

CP パラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移動します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。
CP 指定なしの BMove 命令では、アームは必ず減速して、指定された目標座標に停止します。

参照

AccelS, BGo, Find, !並列処理!, P# = ポイント指定, SpeedS, TGo, Till, TMove, Tool

BMove 使用例

> **BMove** XY(100, 0, 0, 0) '(ローカル座標系で)X 方向に 100mm 移動

Function BMoveTest

```

Speed 50
Accel 50, 50
SpeedS 100
AccelS 1000, 1000
Power High

P1 = XY(300, 300, -20, 0)
P2 = XY(300, 300, -20, 0) /L
Local 1, XY(0, 0, 0, 45)

Go P1
Print Here
BMove XY(0, 50, 0, 0)
Print Here

Go P2
Print Here
BMove XY(0, 50, 0, 0)
Print Here

BMove XY(0, 50, 0, 0) /1
Print Here

```

Fend

[出力結果]

```

X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /R /0
X: 300.000 Y: 350.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /R /0
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0
X: 300.000 Y: 350.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0
X: 264.645 Y: 385.355 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0

```

Boolean

Boolean 型変数を宣言します。(サイズ: 2 バイト)

書式

Boolean 変数名 [(配列変数の最大要素番号)] [, 変数名[(配列変数の最大要素番号)]...]

パラメーター

変数名 Boolean 型と宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。

各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Boolean は、変数を Boolean 型と宣言するときに使います。Boolean 型変数は、“True”または“False”のどちらか 1 つです。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UINT64, UShort

Boolean 使用例

```
Boolean partOK
Boolean A(10)           'Boolean 型の 1 次元配列
Boolean B(10, 10)       'Boolean 型の 2 次元配列
Boolean C(5, 5, 5)      'Boolean 型の 3 次元配列

partOK = CheckPart()
If Not partOK Then
    Print "Part check failed"
EndIf
```

BOpen

ファイルをバイナリーモードで開きます。

書式

BOpen ファイル名 As #ファイル番号

.

Close #ファイル番号

パラメーター

ファイル名	パスを含んだファイル名の文字列を指定します。 ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。 詳細は、ChDisk を参照してください。
ファイル番号	30~63 の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号でオープンします。このステートメントは指定したファイルにバイナリーモードでアクセスするときに使います。指定したファイルが存在しない場合、新規でファイルを作成します。存在する場合は、既存データの先頭から、データを読み書きします。バイナリーモードで、データを読んだり、書いたりする場合は、それぞれ ReadBin, WriteBin コマンドを使用してください。

注意

ネットワークパスが使用できます。

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号は使えません。ファイル番号は、ファイル操作命令 (ReadBin, WriteBin, Seek, Eof, Flush, Close) で使用します。

Seek 命令で、ファイルの読み込み/書き込み位置 (ポインター) を切り替えられます。読み込みアクセスと書き込みアクセスを切り替える場合は、Seek 命令でファイルポインター位置を再設定してください。

Close ステートメントでファイルを閉じ、ファイル番号を解放します。

複数のタスクで同一の番号を使用しないように、FreeFile 関数でファイル番号を取得してください。

参照

Close, AOpen, FreeFile, ReadBin, ROpen, UOpen, WOpen, WriteBin

BOpen 使用例

```
Integer fileNum, i

fileNum = FreeFile
BOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    WriteBin #fileNum, i
Next i

Flush #fileNum
Seek #fileNum, 10
ReadBin #fileNum, i
Print "data = ", i
Close #fileNum
```

Box

進入検出エリアの設定と表示をします。

書式

- (1) Box エリア番号 [, ロボット番号], X 軸下限位置, X 軸上限位置, Y 軸下限位置, Y 軸上限位置, Z 軸下限位置, Z 軸上限位置 [ローカル番号]
- (2) Box エリア番号, ロボット番号, X 軸下限位置, X 軸上限位置, Y 軸下限位置, Y 軸上限位置, Z 軸下限位置, Z 軸上限位置, リモート出力論理 [ローカル番号]
- (3) Box エリア番号, ロボット番号
- (4) Box

パラメーター

エリア番号	設定するエリアの番号を 1~15 の整数値で指定します。
ロボット番号	設定するロボットの番号を整数値で指定します。 (1) の書式で省略した場合、現在選択されているロボットが対象となります。 (2), (3) の書式では、省略できません。
X 軸下限位置	設定するエリアの下限位置の X 座標値 (実数)を、数値または式で指定します。
X 軸上限位置	設定するエリアの上限位置の X 座標値 (実数)を、数値または式で指定します。
Y 軸下限位置	設定するエリアの下限位置の Y 座標値 (実数)を、数値または式で指定します。
Y 軸上限位置	設定するエリアの上限位置の Y 座標値 (実数)を、数値または式で指定します。
Z 軸下限位置	設定するエリアの下限位置の Z 座標値 (実数)を、数値または式で指定します。
Z 軸上限位置	設定するエリアの上限位置の Z 座標値 (実数)を、数値または式で指定します。
リモート出力論理	On Off リモート出力の論理を設定します。Box 進入時に I/O 出力を On したい場合には、On を設定、Box 進入時に I/O 出力を Off したい場合には、Off を設定します。パラメーターを省略した場合は、On が設定されます。
ローカル番号	ローカル座標系番号 (0 から 15)を指定します。 番号の前に、必ず“/LOCAL”をつけてください。パラメーターを省略した場合は、ローカル座標系番号 0 が設定されます。

結果

- (3)の書式で指定すると、指定したエリア番号のエリア設定を表示します。
- (4)の書式で指定すると、現在選択されているロボットの設定されているすべてのエリア設定を表示します。

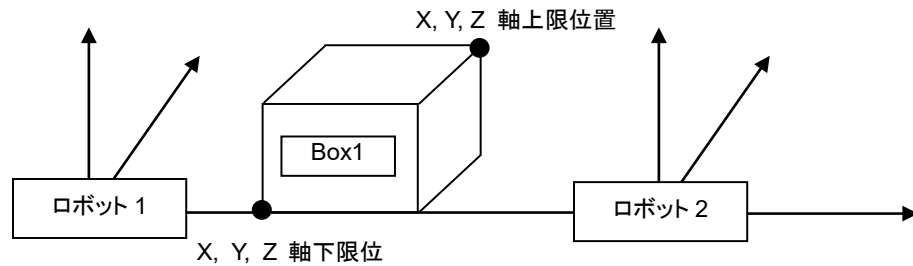
解説

Box は、進入検出エリアを設定します。進入検出エリアを設定すると、現在選択しているツールに基づいて算出される手先位置が、設定した進入検出エリア内に入っているかどうかを検出することができます。進入検出エリアはロボットのベース座標系、またはローカル番号で指定されたローカル座標系上に設定され、指定座標系の X, Y, Z 各軸に指定された下限位置と上限位置の間が進入検出エリアとなります。

進入検出エリアを設定すると、コントローラーの起動中はロボットのモーターパワーの状態に関わらず常に検出処理が実行されます。

進入検出の結果は、GetRobotInsideBox 関数、InsideBox 関数を使っていつでも取得できます。また Wait コマンドの待ち条件式として GetRobotInsideBox 関数を利用することができます。また、検出結果を I/O に出力するようにリモート出力設定することもできます。

複数のロボットで1つのエリアを共有するには、それぞれのロボット座標から見たエリアを定義する必要があります。



ロボット1から見た Box1 の設定

Box 1, 1, 100, 200, 0, 100, 0, 100

X, Y, Z 軸下限位置の座標が(100, 0, 0)で X, Y, Z 軸上限位置の座標が (200, 100, 100)

ロボット2から見た Box1 の設定

Box 1, 2, -200, -100, 0, 100, 0, 100

X, Y, Z 軸下限位置の座標が(-200, 0, 0)で X, Y, Z 軸上限位置の座標が (-100, 100, 100)

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

各座標軸に進入検出エリアの設定をオフにする

各座標軸に設定をオフすることができます。例えば、Z 軸の設定だけをオフにするためには、Z 軸下限位置および Z 軸上限位置を“0”に設定してください。例: Box 1, 200, 300, 0, 500, 0, 0

この場合、XY の 2 次元平面に進入しているかどうかを判別します。

進入検出エリアのデフォルト

Box のデフォルトは、“0, 0, 0, 0, 0, 0”です。(進入検出エリアの設定は、オフになります。)

ツール選択

進入検出は、現在選択しているツールで行います。したがって、ツールの選択を変更した場合、ロボットが動作していないのにエリア内からエリア外、またはその逆となるケースがあります。

付加軸

付加軸 (走行軸含む) S, T 軸を持つロボットの場合設定される進入検出平面は、付加軸の位置には依存しません。ロボットのベース座標系を基準に設定されます。

ヒント

Box をロボットマネージャーで設定する

EPSON RC+ では、[プロジェクト]メニュー-[ロボットマネージャー]の[進入検出エリア]パネルから Box 値を設定できます。

参照

BoxClr, BoxDef, GetRobotInsideBox, InsideBox, Plane

Box 使用例

<例 1>

下記はコマンドウィンドウから **Box** 値を設定し、値を表示する簡単な操作例です。

```
> Box 1, -200, 300, 0, 500, -100, 0

> Box
Box 1: 1, -200.000, 300.000, 0.000, 500.000, -100.000, 0.000, ON
/LOCAL0
```

<例 2>

下記はローカル番号に 1,2 を指定して **Box** 値を設定する簡単なプログラムです。

```
Function SetBox

Integer i

Box 1, -200, 300, 0, 500, -100, 0 /LOCAL1

i = 2
Box 2, 100, 200, 0, 100, -200, 100 /LOCAL(i)

Fend
```

Box 関数

設定された進入検出エリアを返します。

書式

Box(エリア番号[, ロボット番号], 参照データ)

パラメーター

エリア番号	確認するエリア番号を式または数値で指定します。
ロボット番号	設定するロボットの番号を整数値で指定します。 省略した場合、現在選択されているロボットが対象となります。
参照データ	戻り値として返すデータを整数値で指定します。 1: 下限位置 2: 上限位置

戻り値

参照データとして 1 を指定すると、Box 設定値で指定した X 軸下限位置をポイントデータの X として、Y 軸下限位置をポイントデータの Y として、Z 軸下限位置をポイントデータの Z として返します。

参照データとして 2 を指定すると、Box 設定値で指定した X 軸上限位置をポイントデータの X として、Y 軸上限位置をポイントデータの Y として、Z 軸上限位置をポイントデータの Z として返します。

参照

Box, BoxClr, BoxDef, GetRobotInsideBox, InsideBox

Box 関数使用例

```
P1 = Box(1, 1)
P2 = Box(1, 2)
```

BoxClr

設定された Box をクリアします。

書式

BoxClr エリア番号[, ロボット番号]

パラメーター

エリア番号	設定をクリアにする進入検出エリア番号 (1~15 の整数)を、式または数値で指定します。
ロボット番号	設定するロボットの番号を整数値で指定します。 省略した場合、現在選択されているロボットが対象となります。

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

Box, BoxDef, GetRobotInsideBox, InsideBox

BoxClr 関数使用例

下記は BoxClr 関数を使用したプログラムです。

```
Function ClearBox  
  
    If BoxDef(1) = True Then  
        BoxClr 1  
    EndIf  
End
```

BoxDef 関数

進入検出エリアが設定されているかどうかを返します。

書式

BoxDef(エリア番号[, ロボット番号])

パラメーター

エリア番号	状態を返す進入検出エリアの番号 (1~15 の整数)を指定します。
ロボット番号	設定するロボットの番号を整数値で指定します。 省略した場合、現在選択されているロボットが対象となります。

戻り値

エリア番号で指定された進入検出エリアが設定されている場合は“True”、設定されていない場合は“False”を返します。

参照

Box, BoxClr, GetRobotInsideBox, InsideBox

BoxDef 関数使用例

下記は BoxDef 関数を使用したプログラムです。

```
Function ClearBox  
  
    If BoxDef (1) = True Then  
        BoxClr 1  
    EndIf  
End
```

Brake

現在のロボットの指定した関節のブレーキを、オン、またはオフします。

書式

Brake 状態, 関節番号

パラメーター

状態 ブレーキをかけるとき: On を使います。
 ブレーキを解除するとき: Off を使います。

関節番号 1~6 までの関節番号を指定します。

解説

Brake コマンドは、垂直 6 軸型ロボット (N シリーズを含む) の 1 つの関節に対してブレーキをかけたり解除したりします。水平多関節ロボット (RS シリーズを含む) では、動作しません。

Brake コマンドは、メンテナンス作業者のみが使うように設計されたものです。

Brake 命令を実行すると、ロボット制御パラメーターを初期化します。

詳細は、Motor On を参照してください。



■ ブレーキをオフするときは、注意してください。関節が正しく保持されていることを必ず確認してください。関節が正しく保持されていないと、落下の原因となりロボットの故障や作業者が重傷を負う原因になります。

注意

必ず非常停止スイッチを手元に置いてから、Brake Off 命令を実行してください。

コントローラーが非常停止状態になると、モーターブレーキがロックされます。Brake Off 命令を実行すると、ロボットアームが自重で下降する可能性があります。

参照

Motor, Power, Reset, SFree, SLock

Brake 使用例

```
> brake on, 1  
  
> brake off, 1
```

Brake 関数

指定した関節のブレーキ状態を返します。

書式

Brake (関節番号)

パラメーター

関節番号 関節番号を整数値、または整数式で指定します。指定できる値は 1~ロボットの関節数までです。

戻り値

0 = ブレーキ Off, 1 = ブレーキ On

参照

Brake

Brake 関数使用例

```
If brake (1) = OFF Then  
  Print "Joint 1 brake is off"  
EndIf
```

BSet 関数

指定した数値の 1 ビットをセットし、その結果を返します。

書式

BSet (数値, ビット番号)

パラメーター

数値 ビットをセットする数値を、式または直接数値で指定します。

ビット番号 セットするビット番号 (0~31 の整数値)を、式または直接数値で指定します。

戻り値

ビットをセットした値 (整数)を返します。

参照

BCClr, BClr64, BSet64, BTst, BTst64

BSet 関数使用例

```
flags = BSet(flags, 1)
```


BSet64 関数

指定した数値の 1 ビットをセットし、その結果を返します。

書式

BSet64 (数値, ビット番号)

パラメーター

数値 ビットをセットする数値を、式または直接数値で指定します。

ビット番号 セットするビット番号 (0~63 の整数値) を、式または直接数値で指定します。

戻り値

ビットをセットした値 (整数) を返します。

参照

BClr, BClr64, BSet, BTst, BTst64

BSet 関数使用例

```
flags = BSet64(flags, 1)
```

BTst 関数

数値の 1 ビット状態を返します。

書式

BTst (数値, ビット番号)

パラメーター

数値 ビットテストをする数値を、式または直接数値で指定します。

ビット番号 テストするビット番号 (0~31 の整数値) を数値で指定します。

戻り値

ビットテストの値を 1 または 0 で返します。

参照

BClr, BClr64, BSet, BSet64, BTst64

BTst 関数使用例

```
If BTst(flags, 1) Then  
    Print "Bit 1 is set"  
EndIf
```

BTst64 関数

数値の 1 ビット状態を返します。

書式

BTst64 (数値, ビット番号)

パラメーター

数値 ビットテストをする数値を、式または直接数値で指定します。

ビット番号 テストするビット番号 (0~63 の整数値) を数値で指定します。

戻り値

ビットテストの値を 1 または 0 で返します。

参照

BClr, BClr64, BSet, BSet64, BTst

BTst64 関数使用例

```
If BTst64(flags, 1) Then
    Print "Bit 1 is set"
EndIf
```

Byte

Byte 型の変数を宣言します。(サイズ: 2 バイト)

書式

Byte 変数名 [(配列変数の最大要素番号)] [, 変数名[(配列変数の最大要素番号)]...]

パラメーター

変数名 Byte 型と宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Byte は、変数を Byte 型と宣言するときに使います。Byte 変数の範囲は、-128~+127 までです。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Byte 使用例

下記の例では、Byte 型変数を宣言して、その変数に値を与えています。変数 test_ok の最上位ビットが 1 か 0 かを見えています。その結果はディスプレイに表示されます。(この例では、変数に値 15 が与えられているため、変数 test_ok 値の高いビットが常に設定されています。)

```
Function Test
    Byte A(10)           'Byte 型の 1 次元配列
    Byte B(10, 10)       'Byte 型の 2 次元配列
    Byte C(5, 5, 5)      'Byte 型の 3 次元配列
    Byte test_ok
    test_ok = 15
    Print "Initial Value of test_ok = ", test_ok
    test_ok = (test_ok And 8)
    If test_ok <> 8 Then
        Print "test_ok high bit is ON"
    Else
        Print "test_ok high bit is OFF"
    EndIf
Fend
```

Calib

現在のアーム姿勢のパルス値を、CalPls により設定されているパルス値に置き換えます。

書式

Calib 関節番号 1 [, 関節番号 2] [, 関節番号 3] [, 関節番号 4] [, 関節番号 5] [, 関節番号 6]
[, 関節番号 7] [, 関節番号 8] [, 関節番号 9]

パラメーター

関節番号	キャリブレーションを行う関節番号 (1~9 の整数)を直接数値で指定します。 通常は、キャリブレーションは 1 関節ずつ行いますが、このコマンドでは、9 関節までのキャリブレーションを同時に行うことができます。 付加軸の S 軸は 8、T 軸は 9 になります。

解説

自動的にオフセット (Hofs) 値を演算して設定します。このオフセットは、各関節モーターの原点を対応するロボットのメカ原点に合わせるために必要です。

Calib は、モーターのパルス値が変わったときに使われるコマンドです。一般的な用法としては、モーターを交換した後の作業として使われます。通常、キャリブレーション位置パルス値は、CalPls パルス値と同じになるはずですが、モーターの交換などのメンテナンス作業の後では、この 2 つの値は一致しません。そのためキャリブレーションが必要になります。

キャリブレーションする位置までアームを移動させてから、Calib コマンドを実行します。Calib を実行すると、キャリブレーション位置パルス値が CalPls パルス値 (キャリブレーション位置に対する正しいパルス値) に変更されます。

正確にキャリブレーションを行うためには、Hofs 値が設定されなければいけません。キャリブレーションを行う位置までアームを移動させてから、Calib を実行すると、自動的に Hofs 値が演算されます。コントローラーは、キャリブレーションパルス値と CalPls パルス値に基づいて自動的に Hofs 値を演算します。

注意

Calib コマンドの使用には注意が必要です

Calib はメンテナンス作業だけの目的に設計されています。必要なとき以外は使わないでください。
Calib を実行すると Hofs 値が置き換えられます。Hofs 値が知らない間に変更されていると、ロボットが思わぬ動きをすることになりますので、Calib は必要なとき以外は絶対に使わないでください。
関節精度補正の対応機種では、Calib を実行すると、指定した軸について JointAccuracy で設定した補正值が“0”になります。

本コマンド実行後は、安全機能マネージャーを起動してください (Safety 基板を搭載したコントローラーのみ)

Safety 基板を搭載したコントローラーでは、コントローラーの Hofs 値と安全機能を実現する Safety 基板の Hofs 値が一致する必要があります。
本コマンドを実行すると、コントローラーの Hofs 値のみが変更され、Safety 基板設定と差異が生じるため、ワーニングが発生します。
このため、本コマンド実行後は、Safety 基板設定を更新するために、安全機能マネージャーを起動してください。
詳細は、以下のマニュアルを参照してください。
ロボットコントローラー 安全機能マニュアル

起こりやすいエラー

関節番号が指定されていないとエラーになります

Calib コマンドを使うとき、関節番号が指定されていないと、エラーになります。

参照

CalPls, JointAccuracy, HofsJointAccuracy, Hofs

Calib 使用例

コマンドウィンドウからの操作

```
> CalPls          '現在の CalPls 値を表示
65523, 43320, -1550, 21351
> Pulse          '現在の Pulse 値を表示
PULSE: 1: 65526 pls 2: 49358 pls 3: 1542 pls 4: 21299 pls
> Calib 2        '第2関節のみキャリブレーションを実行
> Pulse          '変更された Pulse 値を表示
PULSE: 1: 65526 pls 2: 43320 pls 3: 1542 pls 4: 21299 pls
>
```

Call

ファンクションをサブルーチンとして呼び出します。

書式

Call ファンクション名 [(引数のリスト)]

パラメーター

ファンクション名	呼び出すファンクション名を指定します。
引数のリスト	ファンクション宣言で指定された引数のリストを指定します。引数は、下記の書式を使用してください。省略可能です。 [ByRef]変数名[()]、または数式 ByRef 呼び出すファンクションの変数を参照する場合は、ByRefを指定します。この場合、ファンクション内での引数の変更を呼び出し側の変数に反映させることができます。参照で渡された値は変更できます。省略可能です。

解説

Call 命令により、Function...Fend が定義するファンクションに、プログラム制御を移します。Call 命令により、プログラムの実行は現在のファンクションから、Call 命令が指定するファンクションに移行します。プログラムは、Exit Function か Fend が見つかるまで、そのまま呼び出されたファンクションで続行します。そして、Call 命令の次のステートメントで、制御はもとのファンクションに戻ります。

Call キーワードや引数の括弧は、省略してもかまいません。下記例を参照してください。

```
Call MyFunc(1, 2)
MyFunc 1, 2
```

DLL (ダイナミックリンクライブラリー)で定義されている外部ファンクションを呼び出すことも可能です。詳細は Declare ステートメントを参照してください。

ファンクション内でサブルーチンを実行するには、GoSub...Return を使ってください。

引数として変数を指定することもできます。ByRef パラメーターを指定し、ファンクション内での引数の変更を呼出側変数に反映させることができます。

ByRef パラメーターを指定する場合、ファンクション定義 (Function 文)や、DLL ファンクション定義 (Declare 文)の引数リストでも同様に ByRef を指定する必要があります。

配列変数を引数として渡すときには、ByRef が必須となります。

参照

Function, GoSub

Call 使用例

```
<File1: MAIN.PR>
Function main
    Call InitRobot
Fend

<File2: INIT.PR>
Function InitRobot

    If Motor = Off Then
        Motor On
    EndIf
    Power High
    Speed 50
    Accel 75, 75
Fend
```

CalPls

キャリブレーションに使用する位置姿勢パルス値の設定と表示をします。

書式

- (1) CalPls 第 1 関節パルス値, 第 2 関節パルス値, 第 3 関節パルス値, 第 4 関節パルス値 [, 第 5 関節パルス値, 第 6 関節パルス値] [, 第 7 関節パルス値] [, 第 8 関節パルス値, 第 9 関節パルス値]
 (2) CalPls

パラメーター

- 第 1 関節パルス値 : 第 1 関節のパルス値 (整数)を式または数値で指定します。
 第 2 関節パルス値 : 第 2 関節のパルス値 (整数)を式または数値で指定します。
 第 3 関節パルス値 : 第 3 関節のパルス値 (整数)を式または数値で指定します。
 第 4 関節パルス値 : 第 4 関節のパルス値 (整数)を式または数値で指定します。
 第 5 関節パルス値 : 第 5 関節のパルス値 (整数)を式または数値で指定します。省略可能です。
 第 6 関節パルス値 : 第 6 関節のパルス値 (整数)を式または数値で指定します。省略可能です。
 第 7 関節パルス値 : 第 7 関節のパルス値 (整数)を式または数値で指定します。省略可能です。
 第 8 関節パルス値 : 第 8 関節のパルス値 (整数)を式または数値で指定します。省略可能です。
 第 9 関節パルス値 : 第 9 関節のパルス値 (整数)を式または数値で指定します。省略可能です。

戻り値

パルス値を省略すると、現在設定されているパルス値を表示します。

解説

キャリブレーションを行うための位置の、正しいパルス値を入力、保存します。

この命令はメンテナンス用です。モーター交換をしたときなど、モーターの原点がマニピュレーターアームのメカ原点とずれてしまったときに使用します。この原点を一致させる作業をキャリブレーションといいます。

正常な状態では、キャリブレーションを行うための位置のパルス値と、CalPls により設定されているパルス値は一致していますが、モーター交換などメンテナンスの後では、この 2 つの値が一致なくなり、キャリブレーションが必要になります。

キャリブレーションの 1 つとして、特定の位置にアームを移動し、Calib を実行するという方法があります。Calib を実行することで、キャリブレーションのための特定の位置のパルス値が、CalPls により設定されているパルス値(キャリブレーション位置に対する正しいパルス値)に変更されます。

キャリブレーションを実行するためには Hofs 値が設定されなければいけません。Hofs 値を自動的に演算するためには、アームをキャリブレーションを行うための位置までもっていき、Calib を実行します。コントローラーは、キャリブレーションを行うための位置のパルス値に基づいて自動的に Hofs 値を演算します。

注意**CalPls 値は電源オフによって変更できません**

CalPls 値は、コントローラーの電源をオフした後再起動しても初期化されません。CalPls 値を変更する方法は、Calib コマンドの実行のみです。

参照

Calib, Hofs

CalPls 使用例**モニターウィンドウ操作**

```
> CalPls      ' 現在の CalPls 値を表示
65523, 43320, -1550, 21351
> Pulse
PULSE: 1: 65526 pls  2: 49358 pls  3: -1542 pls  4: 21299 pls
> Calib 4
> Pulse
PULSE: 1: 65526 pls  2: 49358 pls  3: -1542 pls  4: 21351 pls
>
```

CalPls 関数

CalPls で設定されたキャリブレーションに使用するパルス値を返します。

書式

CalPls(関節番号)

パラメーター

関節番号 参照する関節番号か、CalPls 設定の有無を返す 0 を、式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

指定関節の CalPls 設定値 (整数値, 単位: パルス)を返します。関節番号が 0 の場合は、CalPls 設定の有無 (1 または 0)を返します。

参照

CalPls

CalPls 関数使用例

CalPls 関数を使用したプログラム例です。

```
Function DisplayCalPlsValues
  Integer i

  Print "CalPls Values:"
  For i = 1 To 4
    Print "Joint ", i, " CalPls = ", CalPls(i)
  Next i
End
```

ChDir

カレントディレクトリーの変更と表示をします。

書式

- (1) ChDir パス名
- (2) ChDir

パラメーター

パス名 変更先のパス名を、直接文字列または文字列式で指定します。
詳細は、ChDisk を参照してください。

解説

- (1) パラメーターを指定すると、指定したディレクトリーに変更します。
- (2) パラメーターを省略するとカレントディレクトリーを表示します。カレントディレクトリーが不明のときに使います。

ディスクが PC の場合に実行できます。
本命令をプログラムから実行するときは、パス名を[""]で囲ってください。

電源オン時は、プロジェクトを閉じた状態であれば、ルートディレクトリーがカレントディレクトリーとなります。プロジェクトを開いた状態であれば、そのプロジェクトがあるディレクトリーがカレントディレクトリーとなります。

ChDrive でドライブを変更した場合、ルートディレクトリーがカレントディレクトリーとなります。

パラメーターは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

ChDrive, ChDisk, CurDir\$

ChDir 使用例

コマンドウィンドウ操作例

```
> ChDir \           'カレントディレクトリーをルートに変更
> ChDir ..         'カレントディレクトリーを親ディレクトリーに変更

> Cd \TEST\H55     'カレントディレクトリーを\TEST\H55 ディレクトリーに変更

> Cd               'カレントディレクトリーを表示
A:\TEST\H55\
```

プログラム実行例

```
ChDir "\"           'カレントディレクトリーをルートに変更
ChDir ".."         'カレントディレクトリーを親ディレクトリーに変更
```

ChDisk

ファイル操作する対象ディスクを設定します。

書式

ChDisk PC|USB|RAM

パラメーター

PC	PC 部上のフォルダー (ハードディスクなど)
USB	SPEL+制御部上の USB メモリー
RAM	SPEL+制御部上のメモリー

解説

ファイル操作の対象ディスクを指定します。デフォルトは PC です。

ロボットコントローラーではファイル操作の対象として以下のディスクをサポートします。

PC	PC 部上のフォルダーを対象とします。 電源投入時の設定は PC で、通常のケースでは PC から変更する必要はありません。 プロジェクトフォルダなどに置かれたファイルにアクセスできます。
USB	コントローラーのメモリーポートに接続した USB メモリーを対象とします。 RC+との連携を行わないなど、PC 部を使用しないケースで、ファイルとのやりとりを行いたい場合に使用します。 T/VT シリーズでは、パラメーターに USB を指定することはできません。
RAM	メモリー上の一時ファイルを対象とします。 コントローラーの電源を落とすとファイルは保存されません。 一時的なデータの保管に使用します。

SPEL+のコマンドには、ChDisk の設定によりファイル操作の対象が変わるコマンドと、変わらないコマンドがあります。また、ChDisk の設定が PC の場合のみ有効なコマンドがあります。

ChDisk ChDrive ChDir の影響を受けない	Curve CVMove LoadPoints SavePoints ImportPoints のファイル名	常にプロジェクトのフォルダーを対象とする。 ファイル名のみが指定可能でパスを指定するとエラー
ChDisk の影響を受けない	OpenDB の Access, Excel ファイル名 ImportPoints のソースパス VLoadModel VSaveImage VSaveModel	常に Windows のフォルダーを対象とする。 ファイル名のみを指定した場合は、カレントドライブやカレントフォルダーの影響を受ける。 フルパスの指定も可能
ChDisk が PC の場合のみ 実行可能	ChDir FolderExists MkDir RenDir Rmdir	ChDisk が PC 以外の設定で実行するとエラー ファイル名やディレクトリー名のみを指定した場合は、カレントドライブやカレントフォルダーの影響を受ける。 フルパスの指定も可能 USB と RAM にはディレクトリーの概念がありません。

ChDisk が USB や RAM でも 実行可能	Copy Del FileDateTime FileExist FileLen AOpen, BOpen, ROpen, UOpen, WOpen Rename	ChDisk が PC の場合 ファイル名やディレクトリー名のみを指定した 場合は、カレントドライブやカレントフォル ダーの影響を受ける フルパスの指定も可能 ChDisk が USB や RAM の場合 ファイル名のみが指定可能でパスを指定す るとエラー
特殊	Declare	詳細は Declare を参照 指定されたファイル名をそのまま処理 カレントドライブやカレントフォルダーの影響 を受けない

ChDisk が PC の場合のフルパスの決定方法は下記の通りです。

ファイル名のみ	“abc.txt”	カレントドライブ+カレントディレクトリー+指定 ファイル名 “C:\EpsonRC70\Projects\ProjectName\abc.txt”
ドライブなしのフルパス	“\abc.txt”	カレントドライブ+指定フルパス “C:\abc.txt”
ドライブありのフルパス	“d:\abc.txt”	指定フルパスそのまま “d:\abc.txt”
ドライブがネットワークフォルダー	“k:\abc.txt”	指定フルパスそのまま “k:\abc.txt”
ネットワークパス	“\\Epson\data\abc.txt”	指定フルパスそのまま “\\Epson\data\abc.txt”

ChDisk の設定は、コントローラーで 1 つです。

システムとして複数のディスクを対象としたい場合、排他制御を行って ChDisk の設定を切り替える必要が
あります。

参照

ChDir, ChDrive, CurDisk\$

ChDisk 使用例

コマンドウィンドウ操作例

> ChDisk PC

ChDrive

カレントドライブを変更します。

書式

ChDrive ドライブ名

パラメーター

ドライブ名 有効なドライブ (1 文字)を指定します。

解説

ディスクが PC の場合に実行できます。

電源オン時は、プロジェクトを閉じた状態であれば、C がカレントドライブとなります。
プロジェクトを開いた状態であれば、そのプロジェクトがあるドライブがカレントドライブとなります。

詳細は、ChDisk を参照してください。

パラメーターは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

ChDir, ChDisk, CurDrive\$

ChDrive 使用例

コマンドウィンドウ操作例

> **ChDrive** d

ChkCom 関数

通信ポートの受信バッファ内の文字数を返します。

書式

ChkCom (通信ポート番号)

パラメーター

通信ポート番号	RS-232C のポート番号の整数値を指定します。
	SPEL+制御部 1~8
	PC 部 1001~1008

戻り値

受信文字数(整数値)を返します。

受信データが存在しない場合、ポートの状態を下記の負の値で返します。

- 2 ポートが他のタスクで使用中
- 3 ポートがオープンされていない

参照

CloseCom, OpenCom, Read, Write

ChkCom 関数使用例

```
Integer numChars  
  
numChars = ChkCom (1)
```

ChkNet 関数

ネットワークポートの受信バッファ内の文字数を返します。

書式

ChkNet(通信ポート番号)

パラメーター

通信ポート番号 TCP/IP のポート番号 (201~216)を指定します。

戻り値

受信文字数 (整数値)を返します。

受信データが存在しない場合、ポートの状態を下記の負の値で返します。

- 1 ポートはオープンされているが、通信が確立していない
- 2 ポートがほかのタスクで使用中
- 3 ポートがオープンされていない

参照

CloseNet, OpenNet, Read, Write

ChkNet 関数使用例

```
Integer numChars
numChars = ChkNet(201)
```


Chr\$ 関数

ASCII コードに対する文字を返します。

書式

Chr\$(コード番号)

パラメーター

コード番号 1~255 までの整数値を指定します。

戻り値

指定したコードに対応する文字を返します。

解説

Chr\$は ASCII コードに対応する文字 (1 文字)を返します。パラメーターを 1~255 以外に設定した場合、エラーが発生します。

参照

Asc, InStr, Left\$, Len, Mid\$, Right\$, Space\$, Str\$, Val

Chr\$ 関数使用例

次の例では、文字列変数を宣言し、文字列“ABC”を設定します。Chr\$関数は、ASCII コードを“A” “B” “C”に変換します。&H は続く数値が 16 進数であることを意味します。(&H41 は 16 進数の 41)

```
Function Test
    String temp$
    temp$ = Chr$(&H41) + Chr$(&H42) + Chr$(&H43)
    Print "The value of temp = ", temp$
Fend
```

ClearPoints

ロボットの、メモリー上の位置データを消去します。

書式

ClearPoints

解説

ClearPoints はロボットの、メモリー上の位置データ領域のデータを初期化します。この命令は新しい位置をティーチングする前に、メモリー上に存在する位置データを消去するときに使います。

参照

Plist, LoadPoints, Savepoints

ClearPoints 使用例

次の例はコマンドウィンドウから、ClearPoints コマンドを使用した簡単な例です。Clearpoints コマンドを実行した後、Plist コマンドを実行し、ティーチングポイントが存在しないことを確認してください。

```
>P1=100,200,-20,0/R
>P2=0,300,0,20/L
>plist
P1=100,200,-20,0/R
P2=0,300,0,20/L
>clearpoints
>Plist
>
```

Close

AOpen, BOpen, ROpen, UOpen, WOpen コマンドでオープンしたファイルをクローズします。

書式

Close #ファイル番号

パラメーター

ファイル番号 30~63 の整数値、または式で指定します。

解説

ファイル番号で参照中のファイルをクローズし、ファイル番号を開放します。

参照

AOpen, BOpen, Flush, FreeFile, Input #, Print #, ROpen, UOpen, WOpen

Close 使用例

下記は、ファイルをオープンしデータを書き込み、同じファイルを再度オープンしてデータを配列に読み込む例です。

```
Integer fileNumber, i, j

fileNumber = FreeFile
WOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

FileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print j
Next i
Close #fileNum
```

CloseCom

OpenCom でオープンした RS-232C ポートをクローズします。

書式

CloseCom #通信ポート番号 | All

パラメーター

通信ポート番号	クローズする RS-232C のポート番号を指定します。
SPEL+制御部	1~8
PC 部	1001~1008
All	All を指定するとそのタスクがオープンした RS-232C ポートをすべてクローズします。

参照

ChkCom, OpenCom

CloseCom 使用例

CloseCom #1

CloseDB

OpenDB コマンドでオープンしたデータベースをクローズし、データベース番号を解放します。

書式

CloseDB #データベース番号

パラメーター

データベース番号 OpenDB で指定したデータベース番号 (501~508 の整数値)を指定します。

解説

データベース, Excel ブックをクローズし、データベース番号を開放します。

注意

RC+ がインストールされている PC の接続が必要です

参照

OpenDB, SelectDB, UpdateDB, DeleteDB, Input #, Print #

CloseDB 使用例

OpenDB 使用例を参照してください。

CloseNet

OpenNet でオープンした TCP/IP ポートをクローズします。

書式

CloseNet #通信ポート番号| All

パラメーター

通信ポート番号 クローズする TCP/IP のポート番号 (201~216)を指定します。
All を指定するとそのタスクがオープンした TCP/IP ポートをすべてクローズします。

参照

ChkNet, OpenNet

CloseNet 使用例

CloseNet #201

Cls

EPSON RC+ の Run ウィンドウ, オペレーターウィンドウ, コマンドウィンドウのテキスト領域をクリアします。
TP のプリントパネルをクリアします。

書式

- (1) Cls #デバイス ID
- (2) Cls

パラメーター

デバイス ID	21 RC+
	24 TP (TP1 のみ)
	20 TP3
	省略された場合はディスプレイデバイスが対象になります。

解説

Cls を EPSON RC+ のコマンドウィンドウのプログラムから実行すると、コマンドウィンドウのテキストエリアをクリアします。

プログラム中で実行した場合、デバイス ID で指定したデバイスの画面をクリアします。

デバイス ID が省略された場合は、ディスプレイデバイスの画面をクリアします。

Cls 使用例

Run ウィンドウ、またはオペレーターウィンドウからこのプログラム例を実行すると、Cls 実行後にテキストエリアがクリアされます。

```
Function main
  Integer i

  Do
    For i = 1 To 10
      Print i
    Next i
    Wait 3
    Cls
  Loop
Fend
```

Cnv_AbortTrack

コンベヤーキューポイントへの追従動作を中断します。

書式

Cnv_AbortTrack [停止 Z 高さ]

パラメーター

停止 Z 高さ 停止時のロボットの Z 位置を実数値で指定します。省略可能です。

解説

コンベヤーキューポイント (コンベヤーの動きに同期したポイント) への動作コマンド実行中、Cnv_AbortTrack は動作コマンドを停止します。

パラメーターの停止 Z 高さを指定した場合、指定値が現在の Z 位置より高い場合のみ指定値に退避動作をした後、追従動作を減速停止します。

パラメーターの停止 Z 高さを省略すると、マニピュレーターは Z 方向の退避動作を行わずに、追従動作を減速停止します。

注意

このコマンドは、コンベヤトラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_RobotConveyor

Cnv_AbortTrack 使用例

- ・ 追従するワークが下流コンベヤーに流れたロボットを監視するタスク

```
Function WatchDownstream
    Robot 1
    Do
        If g_TrackInCycle And Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0 Then
            ' 現在のロボットの追従動作を停止し、Z 軸を 0 に移動
            g_AbortTrackInCycle = TRUE
            Cnv_AbortTrack 0
            g_AbortTrackInCycle = FALSE
        EndIf
        Wait 0.01
    Loop
End
```


Cnv_Accel

コンベヤトラッキングの追従までの加速度、減速度を設定します。

書式

Cnv_Accel コンベヤー番号, 加速度・減速度

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

加速度・減速度 追従動作の加速度, 減速度を設定します。

解説

コンベヤトラッキングの追従動作の加速度、および減速度を設定します。

加速度と減速度を別々に設定することはできません。

加速度指令エラーが発生する場合または、ワークをピックアップする時間を短縮する場合に変更してください。初期値は 2000[mm/sec²]です。

注意

このコマンドは、コンベヤトラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Accel 関数

Cnv_Accel 使用例

Cnv_Accel 1, 2000

Cnv_Accel 関数

コンベヤートラッキングの追従までの加速度、減速度を返します。

書式

Cnv_Accel (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値 (1~16)で指定します。

戻り値

実数値 (単位: mm)で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Accel

Cnv_Accel 関数使用例

```
Print Cnv_Accel (1)
```

Cnv_AccelLim

コンベヤートラッキング追従後の加速度、減速度の制限値を設定します。

書式

Cnv_AccelLim コンベヤー番号, モード番号, 加速度・減速度

パラメーター

コンベヤー番号	コンベヤーの番号を整数値 (1~16)で指定します。
モード番号	コンベヤートラッキングのトラッキングモードを整数値 (0~2)で指定します。
加速度・減速度	コンベヤートラッキング追従後の加速度, 減速度制限値を実数値 (単位: mm/sec ²)で設定します。設定範囲は AccelS コマンドと同じです。

初期値

ピッキング個数優先モード	500 [mm/sec ²]
ピッキング精度優先モード	2000 [mm/sec ²]
可変速コンベヤー対応モード	6000 [mm/sec ²]

解説

コンベヤートラッキング中にコンベヤーの停止と継続実行を繰り返すと、コンベヤーの速度変化に対するロボットの追従遅れが発生します。追従性を上げる場合に加速度、減速度の制限値を設定します。

加速度と減速度を別々に設定することはできません。

制限値を上げ過ぎた場合、コンベヤーの速度ムラやノイズによってロボットの動作が振動的になります。また下げ過ぎた場合、コンベヤーを停止してもロボットは追従して停止せず、ロボットの動作エリア外まで動く可能性があります。その場合は追従中止線を設定するか、下流限でトラッキングを止めるようにプログラムしてください。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_AccelLim 関数

Cnv_AccelLim 使用例

```
Cnv_AccelLim 1,2,7000
```

Cnv_AccelLim 関数

コンベヤートラッキング追従後の加速度、減速度の制限値を返します。

書式

Cnv_AccelLim (コンベヤー番号, モード番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

モード番号 コンベヤートラッキングのトラッキングモードを整数値 (0~2) で指定します。

戻り値

実数値 (単位: mm/sec²) で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_AccelLim

Cnv_AccelLim 関数使用例

```
Print Cnv_AccelLim (1,2)
```

Cnv_Adjust

コンベヤーに対する追従遅れ補正値を取得する動作を実施するかを設定します。

書式

Cnv_Adjust コンベヤー番号, On | Off

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値 (1~16) で指定します。

On | Off コンベヤートラッキングの追従遅れ補正値を取得する動作に設定する場合には、“On”を、取得しない動作にする場合には“Off”を指定します。

解説

コンベヤートラッキングの追従遅れ補正値を取得する動作を実行するかを設定します。

Cnv_Adjust を“On”にした状態で Cnv_QueueGet 関数を実行すること補正値を取得する動作が実施されます。ワークをピックアップする場合は、Cnv_Adjust を“Off”に設定して Cnv_QueueGet 関数を実行します。

Cnv_Adjust を“On”にした場合、補正値を取得後に必ず“Off”にしてください。

Cnv_Adjust は直線コンベヤーでのみ使用可能です。円形コンベヤーでは使用できません。円形コンベヤーの場合は、“On”に設定しても補正値は取得されません。

取得した補正値は、コントローラーの電源をオフにすることでクリアされ、コントローラーの電源オン時は、初期値 “0” が設定されます。そのため、補正値取得後に print Cnv_AdjustGet で補正値を返し、プログラムの Cnv_QueueGet 実行前に Cnv_AdjustSet で補正値を設定してください。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_AdjustGet 関数、Cnv_AdjustSet、Cnv_AdjustClear、Cnv_QueueGet 関数

Cnv_Adjust 使用例

```
Cnv_Adjust 1, On
Jump Cnv_QueueGet (1)
.
.
Cnv_Adjust 1, Off
```

Cnv_AdjustClear

コンベヤーに対する追従遅れ補正値を消去します。

書式

Cnv_AdjustClear コンベヤー番号

パラメーター

コンベヤー番号 コンベヤーの番号を、式または数値 (1~16)で指定します。

解説

コンベヤートラッキングの追従遅れ補正取得動作の結果、補正量、および補正時間を消去し、“0”に設定します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Adjust、Cnv_AdjustSet 関数、Cnv_AdjustGet、Cnv_QueueGet 関数

Cnv_AdjustClear 使用例

```
Cnv_AdjustClear 1
```

Cnv_AdjustGet 関数

コンベヤーに対する追従遅れ補正値を返します。

書式

Cnv_AdjustGet(コンベヤー番号, モード番号)

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値 (1~16) で指定します。
 モード番号 0: 補正取得動作の結果
 1: 補正量
 2: 補正時間

戻り値

モード番号 0: 実数値 0~2 を返します。

0: 補正値の取得動作は未実施
 1: 補正値の取得済み
 2: 補正値の取得失敗

モード番号 1: 実数値 (単位: mm) を返します。

モード番号 2: 実数値 (単位: 秒) を返します。

解説

Cnv_Adjust および Cnv_QueueGet 関数を使用してコンベヤートラッキングの追従遅れ補正値を取得していない場合、モード番号 0~2 のいずれも戻り値は “0” を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Adjust、Cnv_AdjustSet、Cnv_AdjustClear、Cnv_QueueGet 関数

Cnv_AdjustGet 関数使用例

```
Print Cnv_AdjustGet(1, 1)
```

Cnv_AdjustSet

コンベヤーに対する追従遅れ補正値を設定します。

書式

Cnv_AdjustSet コンベヤー番号, 補正量, 補正時間

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

補正量 補正量を実数値(単位: mm)で指定します。

補正時間 補正時間を実数値(単位: 秒)で指定します。

解説

Cnv_AdjustSetを実施しない場合、補正量と補正時間は前回設定した値が適用されます。
コントローラー電源オン後に1度も補正値を取得する動作を実施していない場合、補正量と補正時間は初期値“0”が設定されています。

Cnv_Mode コマンドのモード番号設定が 1: ピッキング精度優先モードの場合のみ、Cnv_AdjustSet コマンドの設定は有効になります。

Cnv_AdjustSet は円形コンベヤーでは使用できません。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Adjust、Cnv_AdjustGet、Cnv_AdjustClear、Cnv_QueueGet 関数

Cnv_AdjustSet 使用例

```
Cnv_AdjustSet 1, 4.5, 0.1
```


Cnv_Downstream

指定コンベヤーの下流限を設定します。

書式

Cnv_Downstream コンベヤー番号, 下流限

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値(1~16)で指定します。

下流限 追従開始領域の下流側の境界線を設定します

解説

キャリブレーションウィザードで設定した下流限を変更できます。ただし、斜め下流限を使用している場合は Cnv_Downstream から下流限の値を変更することはできません。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Upstream

Cnv_Downstream 使用例

```
Cnv_Downstream 1,500
```

Cnv_Downstream 関数

指定コンベヤーの下流限設定値を返します。

書式

Cnv_Downstream(コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値 (1~16)で指定します。

戻り値

直線コンベヤー: 実数値 (単位: mm)で返します。

円形コンベヤー: 実数値 (単位: deg)で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Upstream

Cnv_Downstream 関数使用例

```
Print "Downstream limit: ", Cnv_Downstream(1)
```

Cnv_Fine

指定コンベヤーに対し、コンベヤーの追従完了判断範囲の設定、表示をします。

書式

Cnv_Fine コンベヤー番号 [, Fine 設定値]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

Fine 設定値 トラッキング完了を判断する距離を示す実数値 (単位: mm)で指定します。
値を 0 にすると、Cnv_Fine は使用されません。Fine 設定値を省略すると現在の
Cnv_Fine 設定を表示します。

解説

コンベヤーの追従完了を判断し、次のコマンドを受付可能にするためのワークからの距離を指定します。
“0”に設定すると Cnv_Fine は使用されず、動作命令の完了で次のコマンドを受付可能となります。デフォルト値は“0”で、次のタイミングのとき、デフォルト値が自動的に設定されます。

コンベヤー定義時
コントローラー起動時

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Fine 関数

Cnv_Fine 使用例

```
Cnv_Fine 1, 5
```

Cnv_Fine 関数

指定コンベヤーの追従完了判断範囲の設定を返します。

書式

Cnv_Fine(コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

Cnv_Fine 設定を、実数値 (単位: mm)で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Fine

Cnv_Fine 関数使用例

```
Real f  
  
f = Cnv_Fine(1)
```

Cnv_Flag 関数

追従中止線に対する、追従状態を返します。

書式

Cnv_Flag (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

- 0: 正常に追従動作を実行しました。(キャンセル, 中止されていません)
- 1: ワークが追従中止線を超える見込みのため、追従動作実行をキャンセルしました。
- 2: ワークが追従中止線を超えたため、追従動作を中止しました。
Z 位置が指定の高さまで下がっていません。
- 3: ワークが追従中止線を超えたため、追従動作を中止しました。
Z 位置が指定の高さまで下がっています。
- 4: ワークが追従開始領域外のため、追従動作実行をキャンセルしました。

追従中止線を設定した場合のみ、0 以外の戻り値を返します。

追従中止線の詳細は、ユーザズガイドを参照してください。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

Cnv_Flag 関数使用例

```
Print Cnv_Flag (1)
```

Cnv_LPulse 関数

コンベヤーのトリガーによってラッチされたパルスを返します。

書式

Cnv_LPulse (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

解説

ハードウェアトリガー配線または Cnv_Trigger によってラッチされた最新のコンベヤーのパルスを返します。

戻り値

指定したコンベヤーのラッチされたパルスを Long 型数値で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Trigger, Cnv_Pulse

Cnv_LPulse 関数使用例

```
Print "Latched conveyor position: ", Cnv_LPulse(1)
```

Cnv_Mode

コンベヤートラッキングのトラッキングモードを設定します。

書式

Cnv_Mode (コンベヤー番号,モード番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

モード番号 0: ピッキング個数優先モード
 1: ピッキング精度優先モード
 2: 可変速コンベヤー対応モード

解説

コンベヤートラッキングのトラッキングモードを設定します。

Cnv_Mode は直線コンベヤーでのみ使用します。

トラッキング動作前にトラッキングモードを設定してください。設定がない場合、ピッキング個数優先モードが設定されます。

ピッキング個数優先モード :ピッキング精度はピッキング精度優先モードよりも劣りますが追従に要する時間が短いため、ワーク間隔の狭いコンベヤーシステムやコンベヤー速度の速いコンベヤーシステムに適したモードです。

ピッキング精度優先モード : 追従に要する時間はピッキング個数優先モードよりも長くなりますが、ピッキング精度が向上するため、小さめのワークを使うコンベヤーシステムに適したモードです。

可変速コンベヤー対応モード :ワークに接触して作業している最中に、コンベヤーを停止、または稼働させるようなコンベヤーシステムに適したモードです。

円形コンベヤーがサポートしているモード番号は、“0”のみです。“1”、または“2”を設定した場合は、ロボットはモード番号“0”と同じ動きをします。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Mode 関数

Cnv_Mode 使用例

```
Cnv_Mode 1, 1
```

Cnv_Mode 関数

コンベヤートラッキングのトラッキングモードを返します。

書式

Cnv_Mode (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

実数値 0~2 を返します。

0: ピッキング個数優先モード

1: ピッキング精度優先モード

2: 可変速コンベヤー対応モード

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Mode

Cnv_Mode 関数使用例

```
Print Cnv_Mode (1)
```


Cnv_Name\$関数

指定したコンベヤーの名称を返します。

書式

Cnv_Name\$ (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値(1~16)で指定します。

戻り値

コンベヤー名を文字列で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Number

Cnv_Name\$ 関数使用例

```
Print "Conveyor 1 Name: ", Cnv_Name$(1)
```

Cnv_Number 関数

指定したコンベヤー名称のコンベヤー番号を返します。

書式

Cnv_Number (コンベヤー名称)

パラメーター

コンベヤー名称 コンベヤー名称を文字列で指定します。

戻り値

コンベヤー番号を整数値で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Name\$

Cnv_Number 関数使用例

```
Integer cnvNum  
  
cnvNum = Cnv_Number("Main Conveyor")
```

Cnv_OffsetAngle

コンベヤーキューデータのオフセット値を設定します。

書式

Cnv_OffsetAngle コンベヤー番号 [, オフセット値]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

オフセット値 コンベヤーキューデータのオフセット値を実数値 (単位: degree)で指定します。
省略可能です。省略すると、現在のオフセット値が表示されます。

解説

コンベヤーキューデータのオフセット値を設定します。

Cnv_OffsetAngle は円形コンベヤーでのみ使用します。

コンベヤートラッキングは使用するコンベヤー速度によっては追従遅れが発生することがあります。追従遅れが発生した場合、ロボットは追従遅れ量だけずれた位置をハンドリングします。Cnv_OffsetAngle はキューにオフセット値を与えることでこのずれを修正します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_OffsetAngle 関数

Cnv_OffsetAngle 使用例

```
Cnv_OffsetAngle 1, 5
```

Cnv_OffsetAngle 関数

コンベヤーキューデータのオフセット値を返します。

書式

Cnv_OffsetAngle (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

実数値(単位: degree)を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_OffsetAngle

Cnv_OffsetAngle 関数使用例

```
Real offsetAngle  
offsetAngle = Cnv_OffsetAngle (1)
```

Cnv_Point 関数

センサー座標値をコンベヤー座標値に変換して返します。

書式

Cnv_Point (コンベヤー番号, センサーX 座標値, センサーY 座標値 [, センサーU 座標値])

パラメーター

コンベヤー番号	コンベヤーの番号を整数値 (1~16) で指定します。
センサーX 座標値	センサーX 座標値を実数値で指定します。
センサーY 座標値	センサーY 座標値を実数値で指定します。
センサーU 座標値	センサーU 座標値を実数値で指定します。省略可能です。

戻り値

コンベヤー座標値をポイントデータとして返します。

解説

Cnv_Point 関数は、コンベヤーキューとして追加された座標値を、ポイントデータとして追加するために使います。ビジョンコンベヤーでは、センサーX 座標値、センサーY 座標値は、カメラからのビジョン座標値です。センサーコンベヤーでは、センサー位置がコンベヤー座標系の原点となります。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Speed

Cnv_Point 関数使用例

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
    VGet FindParts.Part.CameraXYU(i), found, x, y, u
    Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_PosErr 関数

現在のトラッキング位置とターゲットとの位置偏差を返します。

書式

Cnv_PosErr (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

実数値 (単位: mm)を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_MakePoint

Cnv_PosErr 関数使用例

```
Print "Conveyor 1 position error: ", Cnv_PosErr(1)
```

Cnv_PosErrOffset

現在のトラッキング位置とターゲットとの位置偏差を補正する値を設定します。

書式

Cnv_PosErrOffset コンベヤー番号, 補正值

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

補正值 コンベヤー速度を予測する時間を実数値 (0~255, 単位: msec)で指定します。

解説

コンベヤートラッキング中にコンベヤーの停止と稼働を繰り返すと、コンベヤーの速度変化に対するロボットの追従遅れによってトラッキング位置とターゲットとの位置偏差が悪化します。

補正值で設定した時間後のコンベヤー速度を予測してコンベヤートラッキングすることで、位置偏差を改善することができます。

Cnv_PosErrOffset は可変速コンベヤー対応モードのみ使用可能です。ピッキング個数優先モードとピッキング精度優先モードの場合は、補正值を設定しても位置偏差は改善できません。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Mode, Cnv_PosErr

Cnv_PosErrOffset 使用例

Cnv_Mode 1, 2 '可変速コンベヤー対応モード

Cnv_PosErrOffset 1, 10 '補正值 10msec

Cnv_Pulse 関数

コンベヤーの現在位置のパルスを返します。

書式

Cnv_Pulse (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

指定したコンベヤーのパルスを Long 型数値で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Trigger, Cnv_LPulse

Cnv_Pulse 関数使用例

```
Print "Current conveyor position: ", Cnv_Pulse(1)
```


Cnv_QueueAdd

コンベヤーキューデータにポイントデータを追加します。

書式

Cnv_QueueAdd コンベヤー番号, ポイントデータ [, ユーザーデータ]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。
ポイントデータ コンベヤーキューデータに追加するポイントデータを指定します。
ユーザーデータ ポイントデータと一緒に登録するユーザーデータを実数値で指定します。省略可能です。

解説

ポイントデータは指定したコンベヤーキューデータの最後尾に追加されます。現在ラッチされているコンベヤーやパルス位置も一緒に登録されます。

Cnv_QueueReject で指定してある距離以下のキューデータがある場合、データは登録されませんが、エラーにはなりません。

キューデータの上限値は 1000 です。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_RobotConveyor

Cnv_QueueAdd 使用例

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
  VGet FindParts.Part.CameraXYU(i), found, x, y, u
  Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_QueGet 関数

指定したコンベヤーキューデータからポイントデータを返します。

書式

Cnv_QueGet(コンベヤー番号 [, インデックス])

パラメーター

コンベヤー番号	コンベヤーの番号を整数値 (1~16) で指定します。
インデックス	取得するキューデータのインデックスを整数値で指定します。(先頭インデックス番号は 0 です。)省略可能です。

戻り値

指定したコンベヤーキューからポイントデータを返します。

解説

Cnv_QueGet は、コンベヤーキューからポイントデータを取得します。インデックスを省略した場合、キューデータの先頭データを返します。インデックスを設定した場合は、設定したインデックスのポイントデータを返します。

Cnv_QueGet はコンベヤーキューからポイントデータを削除しません。削除するためには Cnv_QueRemove を使います。

コンベヤーを動作させながらワークをトラッキングしたい場合は、動作コマンドに Cnv_QueGet を入れる必要があります。

例:

```
Jump Cnv_QueGet (1) ' ワークを追従
```

Cnv_QueGet の戻り値をポイントに代入して、そのポイントにロボットを動作させ、ワークをトラッキングすることはできません。

```
P1 = Cnv_QueGet (1)  
Jump P1 ' ワークを追従しない
```

Cnv_QueGet の戻り値をポイントに代入すると、その座標値は命令実行時のワーク位置になります。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueLen, Cnv_QueRemove

Cnv_QueueGet 関数使用例

```
・ キューの先頭に Jump して追従  
Jump Cnv_QueueGet(1)  
On gripper  
Wait .1  
Jump place  
Off gripper  
Wait .1  
Cnv_QueueRemove 1
```

Cnv_QueueLen 関数

指定したコンベヤーキューのデータ数を返します。

書式

Cnv_QueueLen(コンベヤー番号 [, パラメーター番号])

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

パラメーター番号 キューデータをカウントするエリアを整数値で指定します。省略可能です。

定数	値	内容
CNV_QUELEN_ALL	0	キューデータの総数を返します。
CNV_QUELEN_UPSTREAM	1	追従開始領域より上流のキューデータ数を返します。
CNV_QUELEN_PICKUPAREA	2	追従開始領域内のキューデータ数を返します。
CNV_QUELEN_DOWNSTREAM	3	追従開始領域より下流のキューデータ数を返します。

戻り値

データ数を整数値で返します。

解説

有効なキューデータ数を返します。特に、追従開始領域内のデータ数を取得するのに有効です。

Wait 命令の引数としても使うことが可能です。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueGet

Cnv_QueueLen 関数使用例

```

Do
  Do While Cnv_QueueLen(1, CNV_QUELEN_DOWNSTREAM) > 0
    Cnv_QueueRemove 1, 0
  Loop
  If Cnv_QueueLen(1, CNV_QUELEN_PICKUPAREA) > 0 Then
    Jump Cnv_QueueGet(1, 0) C0
    On gripper
    Wait .1
    Cnv_QueueRemove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop

```

Cnv_QueueList

指定したコンベヤーキューデータの一覧を表示します。

書式

Cnv_QueueList コンベヤー番号 [, 表示数]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

表示数 表示数データ数を整数値で指定します。省略可能です。省略すると、すべてのキューデータを表示します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueGet

Cnv_QueueList 使用例

```
Cnv_QueueList 1
```

Cnv_QueueMove

上流コンベヤーのキューデータを下流コンベヤーのキューに移動します。

書式

Cnv_QueueMove コンベヤー番号 [, インデックス] [, ユーザーデータ]

パラメーター

コンベヤー番号	コンベヤーの番号を整数値 (1~16)で指定します。
インデックス	移動するキューデータのインデックスを整数値で指定します。(先頭インデックス番号は 0 です。) 省略可能です。
ユーザーデータ	データと一緒に登録するユーザーデータを実数値で指定します。省略可能です。

解説

キューデータを下流コンベヤーのキューに移動します。インデックスを省略した場合、先頭 (インデックスが 0)のキューデータを移動します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueGet

Cnv_QueueMove 使用例

Cnv_QueueMove 1

Cnv_QueueReject

キューの2重登録防止のための最小距離を設定、表示します。

書式

Cnv_QueueReject コンベヤー番号 [, 2重登録防止距離]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

拒絶距離 2重登録防止距離の最小値を実数値 (単位: mm)で指定します。マイナス値を指定した場合は、0 mm が設定されます。省略可能です。省略すると、現在の2重登録防止距離が表示されます。

解説

キューの2重登録防止のための最小距離を設定します。ビジョンシステムによりスキャンされ、1回以上検出されていても登録されるのは1度です。Cnv_QueueReject は2重登録防止のシステムフィルターとなります。デフォルトは0 mm です。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueReject 関数

Cnv_QueueReject 使用例

```
Cnv_QueueReject 1, 20
```

Cnv_QueueReject 関数

キューの 2 重登録防止距離を返します。

書式

Cnv_QueueReject (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

戻り値

実数値 (単位: mm) を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueReject

Cnv_QueueReject 関数使用例

```
Real rejectDist  
  
RejectDist = Cnv_QueueReject(1)
```


Cnv_QueueRemove

コンベヤーキューデータからキューデータを削除します。

書式

Cnv_QueueRemove コンベヤー番号 [, インデックス | All]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

インデックス 削除する先頭のインデックスを整数値、またはすべて削除する場合は All で指定します。省略可能です。

解説

コンベヤーキューデータから 1 つ以上のキューデータを削除します。キューデータを使い終わったときに、キューデータを削除します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueAdd

Cnv_QueueRemove 使用例

```
Jump Cnv_QueueGet (1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1

' コンベヤーからデータを削除
Cnv_QueueRemove 1
```

Cnv_QueueUserData

キューエントリに関連したユーザーデータの設定、表示をします。

書式

Cnv_QueueUserData コンベヤー番号 [, インデックス] [, ユーザーデータ]

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。
インデックス キューデータのインデックスを整数値で指定します。省略可能です。
ユーザーデータ ユーザーデータを実数値で指定します。省略可能です。

解説

コンベヤーキューのユーザーデータを設定します。パラメーターのユーザーデータは省略可能ですが、通常のオペレーションでは必要です。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueUserData 関数

Cnv_QueueUserData 使用例

```
Cnv_QueueUserData 1, 1, angle
```

Cnv_QueueUserData 関数

キューエントリーに関連したユーザーデータを返します。

書式

Cnv_QueueUserData (コンベヤー番号 [, インデックス])

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

インデックス キューデータのインデックスを整数値で指定します。省略可能です。

戻り値

実数値を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueUserData

Cnv_QueueUserData 関数使用例

```

' キューに追加
Cnv_QueueAdd 1, Cnv_Point(1, x, y), angle

' キューから削除
angle = Cnv_QueueUserData(1) ' デフォルトのインデックス“0”になる
Jump Cnv_QueueGet(1) :U(angle)
Cnv_QueueRemove 1

```

Cnv_RobotConveyor 関数

追従中のコンベヤー番号を返します。

書式

Cnv_RobotConveyor [(マニピュレーター番号)]

パラメーター

マニピュレーター番号 マニピュレーター番号を整数値で指定します。

戻り値

コンベヤー番号を整数値で返します。追従するコンベヤーがない場合は 0 を返します。

解説

複数のマニピュレーターを使用中の場合、マニピュレーターがどのコンベヤーを追従しているかを確認します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_MakePoint

Cnv_RobotConveyor 関数使用例

```
Integer cnvNum  
  
cnvNum = Cnv_RobotConveyor (1)
```

Cnv_Speed 関数

コンベヤーの動作速度を返します。

書式

Cnv_Speed (コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

実数値 (単位: mm/s)を返します。円形コンベヤーでは、実数値 (単位: degree/s)を返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Pulse

Cnv_Speed 関数使用例

```
Print "Conveyor speed: ", Cnv_Speed(1)
```

Cnv_Trigger

次の Cnv_QueueAdd ステートメントのために、コンベヤーの現在位置をラッチします。

書式

Cnv_Trigger コンベヤー番号

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16) で指定します。

解説

指定したコンベヤーエンコーダーのパルス出力基板に、ハードウェアトリガーが配線されていない場合に使用します。Cnv_Trigger はソフトウェア的なトリガーとなります。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_QueueAdd

Cnv_Trigger 使用例

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
    VGet FindParts.Part.CameraXYU(i), found, x, y, u
    Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```

Cnv_Upstream

指定コンベヤーの上流限を設定します。

書式

Cnv_Upstream コンベヤー番号, 上流限

パラメーター

コンベヤー番号 コンベヤーの番号を式または数値 (1~16) で指定します。

上流限 追従開始領域の上流側の境界線を設定します。

解説

キャリブレーションウィザードで設定した上流限を変更できます。ただし、斜め上流限を使用している場合は Cnv_Upstream から上流限の値を変更することはできません。

注意

このコマンドは、コンベヤトラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Downstream

Cnv_Upstream 使用例

```
Cnv_Upstream 1,200
```

Cnv_Upstream 関数

指定コンベヤーの上流限設定値を返します。

書式

Cnv_Upstream(コンベヤー番号)

パラメーター

コンベヤー番号 コンベヤーの番号を整数値 (1~16)で指定します。

戻り値

直線コンベヤー: 実数値 (単位: mm)で返します。

円形コンベヤー: 実数値 (単位: deg)で返します。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。

参照

Cnv_Downstream

Cnv_Upstream 関数使用例

```
Print "Upstream limit: ", Cnv_Upstream(1)
```


CollisionDetect

現在のロボットの衝突検出機能(ロボット動作の異常の検出機能)をオン、またはオフします。

書式

- (1) CollisionDetect 状態
- (2) CollisionDetect 状態, 関節番号
- (3) CollisionDetect

パラメーター

状態	On: 衝突検出(ロボット動作の異常の検出)を有効にします。 Off: 衝突検出(ロボット動作の異常の検出)を無効にします。
関節番号	スカラロボット (RS シリーズ含む)の場合、1 から 4 まで、 垂直 6 軸型ロボット (N シリーズ含む)の場合、1 から 6 までの関節番号を指定します。

結果

パラメーターが省略されているときは、現在の CollisionDetect 状態を表示します。

解説

ロボットを動かしたい速度と、実際に動いている速度の差(速度偏差値)により、ロボット動作の異常を検出します。本機能で検出できる異常は、A と B に分類されます。

- A: ロボットアームやハンドの衝突や接触の発生
- B: 衝突や接触以外のロボット動作での異常

さらに B の異常は、パワーの状態により、以下のように分類されます。

ハイパワー状態での異常:

- Weight や Inertia の過小設定によるトルク飽和
- 多関節軸の複合動作や長尺物を振り回すことによるトルク飽和
- 電源電圧低下によるトルク飽和
- ハードウェアの異常やソフトウェアの誤作動による異常動作

ローパワー時の異常:

- ハードウェアの異常やソフトウェアの誤動作による異常動作
- 仕様を超えたハンド質量や長尺物の保持によるローパワー時トルク飽和

衝突検出機能は、EPSON RC+7.0 Ver.7.2 以降でサポートされる汎用ロボット (垂直 6 軸型ロボット, スカラロボット)に対応しています。X5 シリーズなど、未サポートのロボットを接続した状態で本コマンドを使用すると、エラーが発生します。

本コマンドの実行は処理時間を要します。サイクルタイムが要求される場合は、コマンドの使用を最小限にしてください。

コマンドにより、全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オンです。

(ファームウェアバージョン Ver7.2.1.x 以降の場合はオン、Ver7.2.0.x 以前の場合はオフ)

コントローラーの電源をオフするとデフォルト値に戻りますが、それ以外の場合は、本コマンドで明示的に設定しない限り、状態は変化しません。

衝突検出時には、以下のメッセージを出力し、ロボットを停止します。

エラー5057 “ハイパワー状態で衝突を検知(ロボット動作の異常の検出)”

エラー5058 “ローパワー状態で衝突を検知(ロボット動作の異常の検出)”

ハイパワーモード時のダメージ低減には、LimitTorque コマンドによるトルク制限機能との併用、ローパワーモード時のダメージ低減には、LimitTorqueLP コマンドによるトルク制限機能との併用も効果的です。

EPSON RC+ 7.0 ユーザーズガイド「6.18.10 衝突検出機能(ロボット動作の異常の検出機能)」の説明も参照してください。

参照

LimitTorque, LimitTorque 関数, LimitTorqueLP, LimitTorqueLP 関数

CollisionDetect 使用例

CollisionDetect On	' 全軸衝突検出を ON とする
CollisionDetect Off, 5	' 第 5 関節のみ OFF とする
CollisionDetect	' on, on, on, on, off, on と表示されます。

CollisionDetect 関数

CollisionDetect 命令の設定値を返します。

書式

CollisionDetect(関節番号)

パラメーター

関節番号 1 から 6 の整数で指定します。

戻り値

CollisionDetect 命令の設定値を整数値で返します。

1 = ON

0 = OFF

参照

CollisionDetect

CollisionDetect 関数使用例

```
Print CollisionDetect(1)    '第 1 関節の CollisionDetect 値を表示
```

Cont

一時停止状態となったコントローラーを再開し、全タスクを継続実行する場合に使用します。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

Cont

解説

本命令をプログラムから実行するには、EPSON RC+ の[セットアップ] – [システム設定] – [コントローラー] – [環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。ただし、この設定を行っても **Trap SGCclose** で起動されるタスクから **Cont** 命令を実行することはできません。

Cont コマンドは、**Pause** ステートメントの実行や、安全扉開入力により一時停止状態となったコントローラーを再開し、全タスクを継続実行する場合に使用します。[オペレーターウィンドウ]の<継続実行>ボタンやリモート入力の “Continue” と同様の機能を持ちます。

WaitRecover 状態 (安全扉開後の復帰待ち状態)で **Cont** コマンドを実行した場合、すべてのロボットが励磁と復帰動作を行ったのち、プログラムの実行が再開します。
ロボットの励磁と復帰動作のみを行いたい場合は **Recover** コマンドを使用してください。



■ プログラムから**Cont**コマンドを実行する場合、コマンドの仕様を理解して、システムとして継続実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

Pause, Recover

Cont 使用例

```
Function main
  Xqt 2, monitor, NoPause
  Do
    Jump P1
    Jump P2
  Loop
Fend

Function monitor
  Do
    If Sw(pswitch) = On then
      Pause
      Wait Sw(pswitch) = Off and Sw(cswitch) = On
      Cont
    EndIf
  Loop
Fend
```

Copy

ファイルをコピーします。

書式

Copy コピー元, コピー先

パラメーター

コピー元	コピーするファイルのパスとファイル名を指定します。 詳細は、ChDisk を参照してください。
コピー先	ソースファイルをコピーする先のパスと、コピーするファイル名を指定します。 詳細は、ChDisk を参照してください。

解説

指定したソースファイルを、指定したコピー先ファイル名にコピーします。

コピー元とコピー先で同じパス名またはファイル名を共有することはできません。
コピー先がすでに存在する場合は、エラーになります。

注意

ネットワークパスが使用できます。

ファイル名には “*”, “?”などのワイルドカードは使えません。

コマンドウィンドウで使った場合、引用符やカンマは省略することができます。

参照

ChDir, Mkdir

Copy 使用例

コマンドウィンドウでの操作例

```
> copy TEST.DAT TEST2.DAT

> Copy TEST.DAT c:           'NG
!!エラー: 7203 アクセスが拒否されました。
> Copy TEST.DAT c:\          'OK
>
```

Cos 関数

指定角度のコサインを返す関数です。

書式

Cos(数値)

パラメーター

数値 角度を数値 (ラジアン) で指定します。

戻り値

指定した数値のコサイン値 (実数値) を返します。

解説

Cos は、指定した角度 (ラジアン) のコサインを返します。指定する角度 (数値) は、ラジアンで指定してください。戻り値の範囲は、-1~1 の間になります。

角度が度で与えられている場合は、DegToRad 関数を使ってラジアンに変換しなければなりません。

参照

Abs, Atan, Atan2, Int, Mod, Not, Sgn, Sin, Sqr, Str\$, Tan, Val

Cos 関数使用例

下記は、Cos を使った簡単なプログラム例です。

```
Function costest
  Real x
  Print "Please enter a value in radians"
  Input x
  Print "COS of ", x, " is ", Cos(x)
End
```

下記は、Cos をコマンドウィンドウから使った例です。

Display the cosine of 0.55:

```
>print cos(0.55)
0.852524522059506
>
```

Display cosine of 30 degrees:

```
>print cos(DegToRad(30))
0.866025403784439
>
```

CP

パスモーションを設定します。

書式

- (1) CP { On | Off }
- (2) 動作命令 目標座標 CP

パラメーター

On | Off On: パスモーションを有効にします。
Off: パスモーションを解除します。

解説

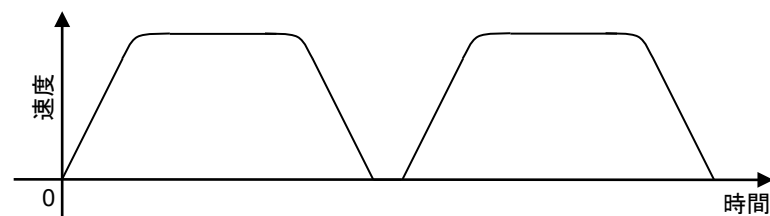
パスモーションは下記の命令で使います。

Arc, Arc3, Go, Jump, Jump3, Jump3CP, JumpTLZ, Move

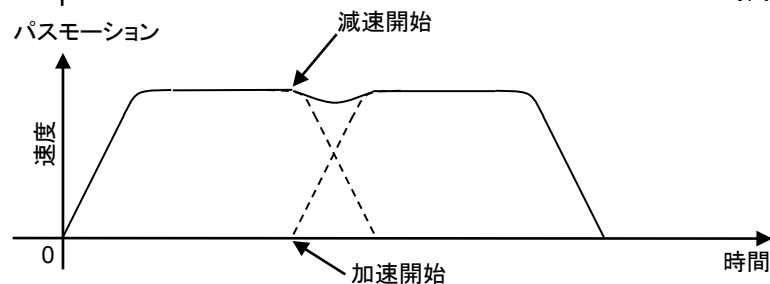
CP On すると、各動作命令で CP パラメーターを指定するかしないかにかかわらず、動作命令は減速開始と同時に次のステートメントが実行されます。この結果、減速動作中に次の動作を開始すると動作軌道が合成されます。

CP Off すると、各動作命令で CP パラメーターを指定した場合にのみ、この機能は有効になります。

通常動作



パスモーション



CP 動作 (Arc, Arc3, Jump3, Jump3CP, JumpTLZ, Move)同士、または PTP 動作 (Go, Jump)同士は、CP On により動作軌道が合成されます。

しかし、CP 動作と PTP 動作の連続軌道の場合は動作軌道が合成されずにいったん減速します。

また、垂直 6 軸型ロボット (N シリーズ含む)での手首特異点を目標値とする CP 動作は、次動作と動作軌道が合成されず、いったん減速します。

下記の場合は CP Off になります。

コントローラー起動時
Motor On 実行
SFree, SLock, Brake 実行
Reset, Reset Error 実行
停止ボタン, Quit All 実行などによるタスクの終了

参照

CP 関数, Arc, Arc3, Go, Jump, Jump3, Jump3CP, JumpTLZ, Move

CP 使用例

```
CP On
Move P1
Move P2
CP Off
```


CP 関数

パスモーションの状態を返します。

書式

CP

戻り値

0 = パスモーションオフ

1 = パスモーションオン

参照

CP

CP 関数使用例

```
If CP = Off Then  
    Print "CP is off"  
EndIf
```

CP_Offset

CP On 時に、後続する動作命令を開始するオフセット時間を設定します。

書式

- (1) CP_Offset [On [, OffsetTime]]
- (2) CP_Offset Off

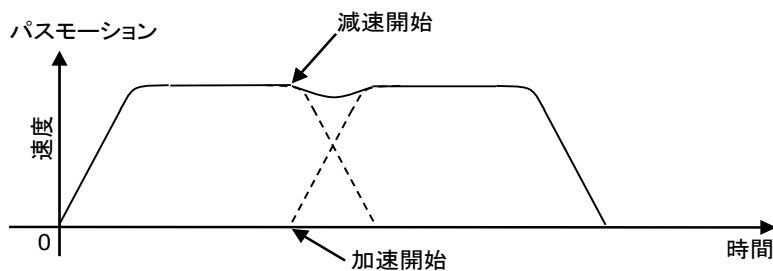
パラメーター

- On | Off** On: CP On 時の動作開始オフセット機能を有効にします。省略した場合は、現在の設定を表示します。
Off: CP On 時の動作開始オフセット機能を無効にします。
- OffsetTime** CP On 時の動作開始オフセット時間を 10~24 (単位: ms)の範囲の実数値で設定します。省略した場合、デフォルト値 10 ms が設定されます。

解説

CP_Offset は以下の動作命令が対象となります。
Move, Arc, Arc3, CVMove

CP On や動作命令に CP パラメーターを付加すると、前の動作の減速開始と同時に次のステートメントが実行されます。この結果、下図のように減速区間と次の動作の加速区間が重なるパスモーションになります。この時、前の動作の減速開始と次の動作の加速開始は、ステートメントの開始処理にかかわるオーバーヘッド時間があるため、厳密に一致しません。そのため、パスモーションのつなぎ目付近では速度が落ち込み、等速軌道になりません。これを改善するため、後続する動作ステートメント開始時間を前倒しするのが CP_Offset の機能です。



CP_Offset をオンにすると、OffsetTime パラメーターに設定された時間分、後続する動作命令の開始処理が早められ、実際のロボットの減速開始と次の動作の加速開始が同期し、パスモーションの等速性能が改善します。OffsetTime パラメーターにはデフォルト値が設定されていますが、アプリケーションによって微調整してください。特に、後続する動作命令が“!並列処理!”を含む場合、動作開始にかかるオーバーヘッド時間が長くなるため、OffsetTime をデフォルト値より大きな値 (16ms 程度) に設定します。

CP_Offset の設定時間 (OffsetTime) を調整するために、調整したい動作を実行中に TCPSpeed を用いてツール先端速度を観測します。適切な OffsetTime が設定されると、動作のつなぎ目での速度が一定に近くなります。OffsetTime が大きすぎる場合 TCPSpeed が上昇し、OffsetTime が小さすぎる場合は、TCPSpeed が落ち込みます。CP_Offset の調整は実機環境で行ってください。実コントローラーとシミュレーターでは動作開始の処理時間が異なるため、適切な調整ができません。

TCPSpeed 計測プログラム例

```

Function main
  Motor On
  Power High

  SpeedsS 250; Accels 1500
  Speed 50; Accel 50, 50

  Go XY(300, 500, 500, 90, 0, 180)

  CP_Offset On
  Xgt printTcPSpeed

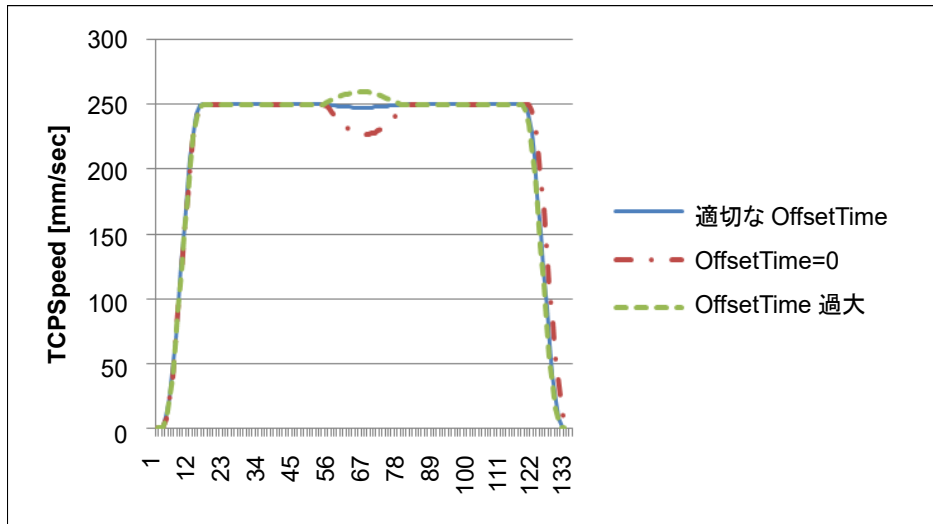
  Move XY(0, 500, 500, 90, 0, 180) CP
  Move XY(-300, 500, 500, 90, 0, 180)

  Quit printTcPSpeed
  CP_Offset Off
Fend

Function printTcPSpeed
  Do
    Print TCPSpeed
  Loop
Fend

```

OffsetTime 調整例



本命令は PTP 動作を対象としません。PTP 動作の場合は、通常通りのパスモーションとなります。

下記の場合は **CP_Offset** はオフになります。

コントローラー起動時 Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

参照

CP_Offset 関数, CP, Move, Arc, Arc3, CVMove

CP_Offset 使用例

```
CP_Offset On
Move P1
Move P2
CP_Offset Off
```

CP_Offset 関数

CP On 時に、後続する動作命令を開始するオフセット時間を返します。

書式

CP_Offset

戻り値

動作の開始オフセット時間を表す実数

参照

CP_Offset

CP_Offset 関数使用例

```
If CP_Offset = 0 Then  
    Print "CP_Offset is off"  
EndIf
```

Ctr 関数

カウンターのカウント値を返す関数です。

書式

Ctr(入力ビット番号)

パラメーター

入力ビット番号 カウンターに定義されている入力ビットの番号です。同時に有効にできるカウンターの最大数は、16 個までです。

戻り値

カウンターの現在カウント値 (0~65535 の整数)

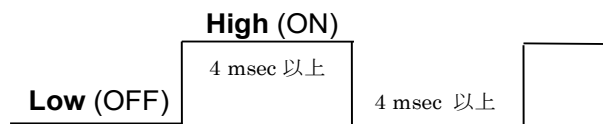
解説

Ctr は、CTReset ステートメントと一緒に使うことで、I/O 入力をカウントとして使えるように設定します。

カウンターに設定された I/O 入力が入力からオンに変わるたびに、その入力は、カウンター上のカウントを 1 つずつ増加させます。

Ctr 関数は、指定するカウンター入力の現在のカウンター値を得るために使えます。どの I/O 入力でもカウンターに指定することができます。しかし、同時に有効にできるカウンターの数は、16 個までです。

カウンターパルス入力のタイミングチャート



参照

CTReset

Ctr 関数使用例

下記は、I/O 入力のカウンターの値を得るプログラムの一例です。

```
CTReset 3 '入力 3 のカウンターを 0 にリセット
On 0      '出力スイッチをオン
Wait Ctr(3) >= 5
Off 0     '入力 3 の入力サイクルが 5 になった場合はスイッチをオフ (出力 0 オフ)
```

CTReset

指定した入力カウンターの値をリセットします。また、入力をカウンター入力と設定します。

書式

CTReset(入力ビット番号)

パラメーター

入力ビット番号 カウンターとして設定される入力ビットの番号です。有効な入力ビットの番号を整数で入力します。使用可能なカウンターの数は、16 までです。

解説

CTReset は、Ctr 関数と一緒に使い、入力をカウンターとして使えるようにします。CTReset は、指定された入力ビットをカウンターとして設定し、そのカウンターをスタートします。もし、指定された入力がすでにカウンターに設定されている場合は、リセットして再スタートします。

注意

電源をオフした場合のカウンター

電源をオフするとすべてのカウンター値は解除されます。

Ctr 関数を使う場合

Ctr 関数を使って、現在の I/O 入力のカウンター値を知ることができます。

参照

Ctr

CTReset 使用例

下記は、I/O の入力カウンター値を得るプログラムの一例です。

```
CTReset 3   '入力 3 のカウンターを 0 にリセット
On 0        '出力スイッチをオン
Wait Ctr(3) >= 5
Off 0       '入力 3 の入力サイクルが 5 になった場合はスイッチをオフ (出力 0 オフ)
```

CtrlDev 関数

現在の制御デバイスの番号を返します。

書式

CtrlDev

戻り値

- 21 PC
- 22 リモート I/O
- 26 リモートイーサネット
- 29 リモート RS232C
- 20 TP3

参照

CtrlInfo 関数

CtrlDev 関数使用例

```
Print "The current control device is: ", CtrlDev
```


CtrlInfo 関数

コントローラーの情報を返します。

書式

CtrlInfo (インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値で指定します。

解説

CtrlInfo 関数で取得可能な情報は次の表のとおりです。

インデックス	ビット	値	解説
0	N/A		旧互換のために残されています。 コントローラーのファームウェアバージョンを取得するにはインデックス 9 を使用してください。
1	コントローラー状態		
	0	&H1	Ready 状態
	1	&H2	Start 状態
	2	&H4	Pause 状態
	3-7		未定義
	8	&H100	非常停止状態
	9	&H200	安全扉開状態
	10	&H400	エラー状態
	11	&H800	重大エラー状態
	12	&H1000	ワーニング状態
	13	&H2000	WaitRecover 状態 (安全扉開からの復帰待ち)
	14	&H4000	Recover 状態 (安全扉開からの復帰動作中)
	15-31		未定義
2	0	&H1	TP1 の Enable スイッチがオン
	1-31		未定義
3	0	&H1	TEACH モード回路異常検出
	1	&H2	安全扉回路異常検出
	2	&H4	非常停止回路異常検出
	3-31		未定義
4	N/A		0 - 実ラン モード 1 - ドライラン モード
5	N/A		制御デバイス: 21 - RC+ 22 - リモート 26 - リモートイーサー ネット 29 - リモート RS232C 20 - TP3
6	N/A		設定されているロボットの台数
7	N/A		操作 モード: 0 - Programing モード 1 - AUTO モード
8	N/A		未定義

インデックス	ビット	値	解説
9	N/A		コントローラーのファームウェアバージョン メジャー番号*1000000 + マイナー番号*10000 + レビジョン番号 *100 + ビルド番号 例 1.6.2.4 の場合 1060204
10	N/A		ハードディスクの SMART ステータス 0 : SMART ステータス正常 1 : SMART ステータス異常 SMART ステータスが異常の場合、ハードディスクが故障する可能性があるので、すみやかにデータのバックアップを行い、新しいハードディスクに交換してください。 RAID オプション使用時は、SMART ステータスは使用できません。常に正常が返ります。
15	N/A		DC 電源電圧値 入力電圧の 100 倍した値が得られます。 例 48.01V の場合、4801 として取得されます。 DC 電源に対応していない機種では、エラーが返ります。
16	N/A		PLC ベンダータイプ 0: None 1: Allen Bradley

戻り値

必要な Long 整数式の値を返します。

参照

RobotInfo, TaskInfo

CtrlInfo 関数使用例

```
Print "The number of robots is: ", CtrlInfo(6)
```

CurDir\$関数

現在のカレントディレクトリーを文字列で返します。

書式

CurDir\$

戻り値

ドライブ名とパス名の文字列を返します。

参照

ChDir, CurDrive\$, CurDisk\$

CurDir\$ 関数使用例

```
Print "The current directory is: ", CurDir$
```

CurDisk\$関数

現在のディスク名を文字列で返します。

書式

CurDisk\$

戻り値

ディスク名の文字列を返します。

参照

ChDisk, CurDir\$, CurDrive\$

CurDisk\$関数使用例

```
Print "The current disk is: ", CurDisk$
```

CurDrive\$関数

現在のドライブ名を文字列で返します。

書式

CurDrive\$

戻り値

ドライブ名の文字列を返します。

参照

ChDrive, CurDir\$, CurDisk\$

CurDrive\$ 関数使用例

```
Print "The current drive is: ", CurDrive$
```

CurPos 関数

ロボットの現在の動作目標位置を返します。

書式

CurPos

戻り値

ロボットの現在の動作目標位置を返します。

参照

InPos, FindPos, RealPos

CurPos 関数使用例

```
Function main
    Xqt showPosition
    Do
        Jump P0
        Jump P1
    Loop
Fend

Function showPosition
    Do
        P99 = CurPos
        Print CX(P99), CY(P99)
    Loop
Fend
```

Curve

自由曲線による CP 制御のためのデータとポイントを作成します。多くのポイントデータを定義し、パスを正確にします。

書式

Curve ファイル名, 動作曲線の開閉, モード指定, 座標軸数, 連続ポイントデータ指定

パラメーター

ファイル名

ポイントデータが保存されるファイル名を文字列で指定します。拡張子は“.CVT”固定です。拡張子を省略した場合、.CVT の拡張子が自動的に追加されます。Curve 命令が実行されると、ファイルが作成されます。

パスを指定することはできません。また、ChDisk などの影響を受けません。詳細は、ChDisk を参照してください。

動作曲線の開閉

曲線動作の終了時における動作曲線の開閉を指定します。このパラメーターは、下記のいずれかの値を指定します。

C -生成する曲線が閉曲線

O -生成する曲線が開曲線

開曲線を指定すると、Curve 命令は、連続ポイントデータの最後のポイントでアームが停止します。閉曲線を指定すると、Curve 命令は、最終ポイントを通過した後も動作を継続し、アームを連続ポイントデータの始点に戻して動作を停止します。

モード指定

姿勢補正するかどうか (アームが U 軸の接線方向に自動的に補間するかどうか)を指定します。上位 4 ビットで ECP 番号が指定できます。

モード指定		姿勢補正	ECP 番号
16 進数	10 進数		
&H00	0	しない	0
&H10	16		1
&H20	32		2
...	...		...
&HA0	160		10
&HB0	176		11
&HC0	192		12
&HD0	208		13
&HE0	224		14
&HF0	240		15
&H02	2	する	0
&H12	18		1
&H22	34		2
...	...		...
&HA2	162		10
&HB2	178		11
&HC2	194		12
&HD2	210		13
&HE2	226		14
&HF2	242		15

姿勢補正を指定する場合は、アームは、連続ポイントデータの始点の U 軸座標のみを使います。姿勢補正は、姿勢軸が XY 平面で常に曲線に接しているように姿勢を維持します。これは、カッターのように、継続して接線方向への制御が必要なツールのときに指定します。U 軸の接線方向に自動円弧補間する閉曲線を指定するときは、U 軸は、始点から 360 度回転します。したがって、CVMove 命令を実行する前に、U 軸の回転によるエラーを防ぐために、Range 命令で U 軸の動作範囲指定を行ってください。

ECPを使用する場合は、上位 4ビットに ECP 番号を指定してください。

ポイントデータに含まれる付加軸の位置を考慮した自由曲線を生成する場合、9 ビット目を“1”に指定してください。例えば、姿勢補正および ECP を使用せず、付加軸位置を考慮した自由曲線を生成する場合には、“&H100”を指定します。

付加軸で自由曲線を生成する場合は、ロボット座標系とは関係なく S 関節、T 関節それぞれ独立に連続ポイントデータをつなぎます。

ただし、付加軸が PG 軸で構成されている場合は、連続ポイントによる自由曲線は生成されず、最終ポイントに動作するデータが作成されます。

座標軸数

曲線動作中に制御される座標軸数を 2, 3, 4, 6 の整数で指定します。

- 2 - 姿勢を含まない XY 平面で自由曲線を生成します。
(垂直 6 軸型 (N シリーズを含む)以外)
- 3 - 姿勢を含まない XYZ 空間で自由曲線を生成します。
(垂直 6 軸型 (N シリーズを含む)以外)
- 4 - 姿勢を含む XYZ 空間で自由曲線を生成します。
(垂直 6 軸型 (N シリーズを含む)以外)
- 6 - 姿勢を含む XYZ 空間で自由曲線を生成します。
(垂直 6 軸型 (N シリーズを含む)のみ)

制御対象に選択されていない軸は、前回のエンコーダーパルス位置を保持し、Curve 動作中は動作しません。

連続ポイントデータ指定 { ポイントデータ| ポイント番号(初め:終わり) } [,出力コマンド] ...

このパラメーターそれぞれのポイントデータをカンマ (,)で区切って指定します。ポイントデータに抜けがなく、昇順、または降順に並んでいる場合は、2 つのポイント番号をコロンで結合して P(1:5)のように指定できます。

動作に同期して、途中で I/O の出力ポートをオン、またはオフする場合は、カンマ (,)で区切って出力命令を記述することができます。

通常は、連続ポイントデータは、下記のようにカンマで区切って指定します。

```
Curve "MyFile", 0, 0, 4, P1, P2, P3, P4
```

または、コロンを使って下記のように指定します

```
Curve "MyFile", 0, 0, 4, P(1:4)
```

上記の例では、P1、P2、P3、P4 を使って曲線を指定しています。出力コマンドは、省略可能で、曲線動作中に出力操作を制御するときに使います。このコマンドは、I/O またはメモリー I/O のオンオフを指定します。出力命令は、直前の、連続ポイントデータの特定のポイントを、アームが通過した後に実行されます。1 つの Curve ステートメントに含むことのできる出力コマンドの数は、最大 16 個です。下記の例では、アームが P2 を通過した後に“On 2”コマンドが実行され、その後アームは、P3 から P10 までのすべてのポイントを通過します。

```
Curve "MyFile", C, 0, 4, P1, P2, ON 2, P(3:10)
```


解説

この命令は、指定されたポイントデータによって、ロボットアームが自由曲線 CP 動作を行うためのファイルを作成し、そのデータをコントローラーのファイルに保存します。この命令によって作成されたデータは、CVMove 命令による CP 動作実行時に使用されます。

カーブファイルは、コントローラー内のコンパクトフラッシュに格納されています。このため、Curve を実行するとコンパクトフラッシュへの書き込みが発生します。あまりにも頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。必要な場合にのみ Curve を実行することを推奨します。

Curve 命令は指定された各ポイントの X, Y, Z, U, V, W 座標値を独立に、3 次式スプライン関数を用いて計算し、それにより軌跡を生成します。よって、ポイント間の間隔が大きかったり、姿勢の変化が大きい場合には、生成される軌跡は想定しづらくなります。

動作時の速度・加減速度は、Curve 命令前に指定しておく必要はありません。CVMove 実行前なら SpeedS や AccelS などの命令を使って変更できます。

ローカル座標系で設定するポイントは、Curve 命令のパラメーターのポイント群の中で使えば、その位置を通過するように曲線を設定できます。指定するポイントデータでローカル属性を持つようにしておくと、Curve 命令の後に続く Local ステートメントのローカル座標系上のポイントに変更することができます。

注意

できる限り姿勢補正を行ってください

できるだけ姿勢補正を行うことをお勧めします。とくに、CVMove を使って同じポイント群を連続ループするような場合は、姿勢補正をしてください。姿勢補正をしないと、特に高速で動作させるような場合、ロボットは正確な位置を保持できないことがあります。

開曲線におけるポイント群のポイント数の範囲

開曲線では、3~1000 ポイントを指定してください。

閉曲線におけるポイント群のポイント数の範囲

RC700 系、RC90 系コントローラーの場合、閉曲線では、3~1000 ポイントを指定してください。

T/VT の場合、閉曲線では、3~300 ポイントを指定してください。(シミュレーターで T/VT を指定した場合、1000 ポイントまで動作しますが、T/VT の実機環境では 300 ポイントまでです。)

ポイント数が多い場合、処理時間が長くなります

最大ポイント数で Curve コマンドを実行すると、開曲線では数秒程度、閉曲線では数十秒程度の時間がかかります。

特に、閉曲線は処理時間が長いので、ポイント数が多い場合は開曲線を使うことを奨励します。

開曲線を使って、閉曲線と近い軌跡を作るための使用例を Curve 使用例 2 に記載します。

ただし、Curve コマンドで自由曲線ファイルを一度生成し、同じファイルで複数回 CVMove 動作を行う場合、Curve コマンド実行時の 1 度のみ上記の時間がかかります。

ファイルの互換性

Ver.7.5.1 以降のファームウェアで作成されたファイルは、それ以前バージョンのファームウェアで使用できません。また、Ver.7.5.1 以前のファームウェアで作成されたファイルは、Ver.7.5.1 以降のファームウェアで使用できます。

起こりやすいエラー

可動範囲外で、アームを動作させようとしたとき

Curve 命令は、設定された曲線の動作範囲のチェックを行うことができません。これは、ロボットアームの動作中にユーザーが設定した曲線の軌跡が、可動範囲外に出るかもしれないことを意味します。このような場合、「動作範囲外」のエラーになります。

参照

AccelS 関数, Arc, CVMove, ECP, Move, SpeedS

Curve 使用例 1

下記の例では、MYCURVE.CVT という名前の自由曲線ファイルを使って、P1~P7 を通過する曲線をトレースし、その間、P2 で出力ポートをオンし、P7 でアームを減速します。

自由曲線を設定

```
> curve "mycurve", 0, 0, 4, P1, P2, On 2, P(3:7)
```

アームを P1 へ直線で移動

```
> jump P1
```

定義した自由曲線“mycurve”でアームを移動

```
> cvmove "mycurve"
```

Curve 使用例 2

以下の例では、(1)開曲線、(2)閉曲線、(3)開曲線で閉曲線に近い動作の例を示します。
教示点は以下のとおりです。

```
P0 = XY(0, 300, -50, 0) '始点・終点
P1 = XY(300, 200, -50, 0)
P2 = XY(300, 400, -50, 0)
P3 = XY(-300, 400, -50, 0)
P4 = XY(-300, 200, -50, 0)
P10 = XY(10, 299.7, -50, 0) '始点の直後
P11 = XY(-10, 299.7, -50, 0) '終点の直前
```

(1)開曲線

開曲線の自由曲線を設定

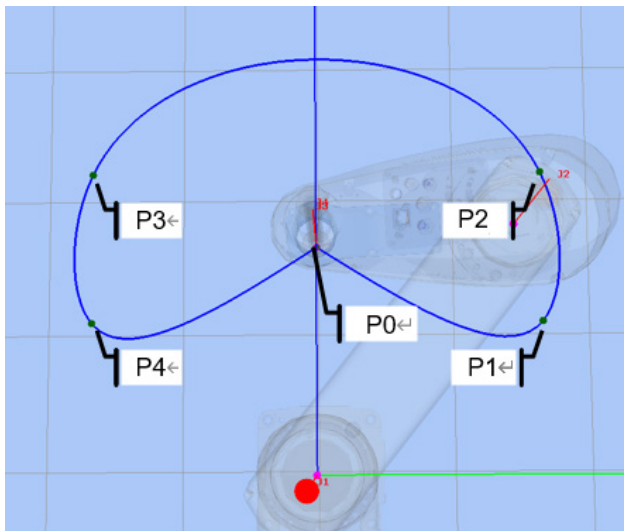
```
> Curve "mycurve_O", 0, 0, 2, P(0:4), P0
```

アームを P0 へ直線で移動

```
> jump P0
```

定義した開曲線の自由曲線“mycurve_O”でアームを移動

```
> CVMove "mycurve_O"
```



開曲線の為、始点と終点と同じだが滑らかにつながらない。

(2)閉曲線

閉曲線の自由曲線を設定

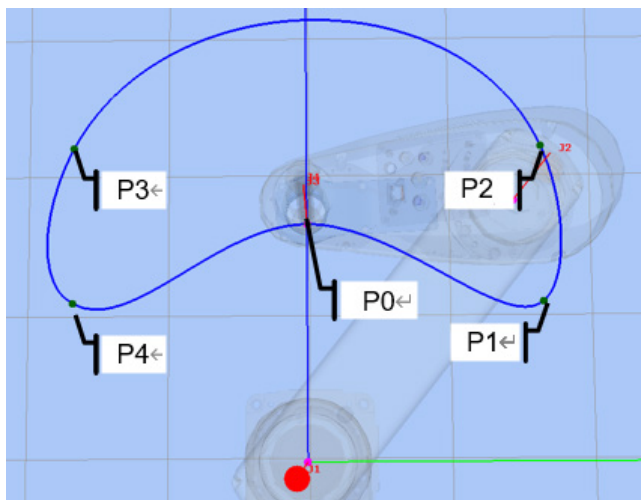
```
> Curve "mycurve_C", 0, 0, 2, P(0:4)
```

アームを P0 へ直線で移動

```
> jump P0
```

定義した閉曲線の自由曲線“mycurve_C”でアームを移動

```
> CVMove "mycurve_C"
```



閉曲線の為、始点と終点は滑らかにつながる。

(3)開曲線で閉曲線に近い動作

開曲線の自由曲線を設定。始点の直後・終点の直前にポイントを設定。

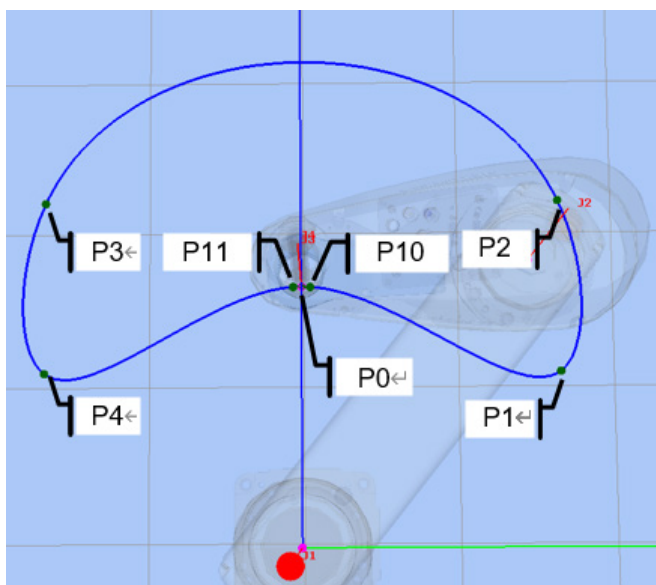
```
> Curve "mycurve_O_mod", 0, 0, 2, P0, P10, P(1:4), P11, P0
```

アームを P0 へ直線で移動

```
> jump P0
```

定義した開曲線の自由曲線“mycurve_O_mod”でアームを移動

```
> CVMove "mycurve_O_mod"
```



開曲線だが、P10, P11 を通ることで、始点と終点は滑らかにつながる。

CVMove

Curve 命令で定義した自由曲線 CP 動作を実行します。

書式

CVMove ファイル名[CP] [Till | Find] [SYNC]

パラメーター

ファイル名	Curve 命令で作成され、コントローラーに保存されたファイル名を、文字列式または直接文字列で指定します。 パスを指定することはできません。また、ChDisk などの影響を受けません。 詳細は、ChDisk を参照してください。
CP	最終ポイントの後のパスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

CVMove は、ファイルのデータが設定する自由曲線 CP 動作を実行します。このファイルは、あらかじめ Curve 命令によって作成されていなければなりません。

ファイル名に拡張子がないと、自動的に“.cvt”がつきます。

CVMove による CP 動作の速度と加減速度は、SpeedS や AccelS 命令によって変更することができます。Curve 命令が、前に Local 定義のポイントを使って動作実行した場合、Local 命令によって位置を変更することができます。

CVMove 実行時には、周辺装置との干渉に十分注意してください。特に、垂直 6 軸型ロボット (N シリーズを含む) では隣り合うポイント間で急激に姿勢を変化させようとする、3 次スプライン関数の性質上、その前後のポイントから姿勢の変化が開始され想定しない動作となる可能性があります。CVMove 実行時には周辺装置との干渉に注意し、軌跡の確認を十分行ってください。

ポイントの指定はできるだけ等間隔で細かくし、隣り合うポイントでのハンド(腕系)姿勢は急激な変化を与えないようにしてください。

CP パラメーターをつけた場合には、動作の減速開始時に次の動作命令の加速を重ね合わせることが可能です。この場合には、目標座標には位置決めされません。

参照

AccelS 関数, Arc, Curve, Move, SpeedS, Till, TillOn

CVMove 使用例

下記の例では、MYCURVE.CVT という名前の自由曲線ファイルを使って、P1 から P7 を通過する曲線をトレースし、P2 で出力ポートをオンし、P7 でアームを減速します。

自由曲線を設定

```
> curve "mycurve", 0, 0, 4, P1, P2, On 2, P(3:7)
```

アームを P1 へ直線で移動

```
> jump P1
```

定義した自由曲線“mycurve”でアームを移動

```
> cvmove "mycurve"
```

CX, CY, CZ, CU, CV, CW, CR, CS, CT

ポイントデータの座標値を設定します。CV, CW は、垂直 6 軸型ロボット (N シリーズを含む)のみ使用することができます。

CR は、Joint 型ロボットのみ使用することができます。

CS, CT は、付加軸が設定されているロボットのみ使用することができます。

書式

CX(座標) = 値

CY(座標) = 値

CZ(座標) = 値

CU(座標) = 値

CV(座標) = 値

CW(座標) = 値

CR(座標) = 値

CS(座標) = 値

CT(座標) = 値

パラメーター

座標 P 番号、P(式)、またはポイントラベルを指定します。

値 設定する座標値を実数値で指定します。

参照

CX, CY, CZ, CU, CV, CW, CR, CS, CT 関数

CX, CY, CZ, CU, CV, CW, CR, CS, CT 使用例

CX(pick) = 25.34

CX, CY, CZ, CU, CV, CW, CR, CS, CT 関数

ポイントデータの座標値を取得します。

CV, CW 関数は、垂直 6 軸型ロボット (N シリーズを含む)のみ使用することができます。

CS, CT 関数は、付加軸が設定されているロボットのみ使用することができます。

書式

CX(座標)
CY(座標)
CZ(座標)
CU(座標)
CV(座標)
CW(座標)
CR(座標)
CS(座標)
CT(座標)

パラメーター

座標 ポイントデータを指定します。

戻り値

指定された座標値を返します。

CX, CY, CZ の各関数の戻り値: 実数値 (単位:mm)です。

CU, CV, CW の各関数の戻り値: 実数値 (単位:deg)です。

CS, CT の各関数の戻り値: 実数値 (単位:mm)、もしくは実数値 (単位:deg)です。付加軸の設定により異なります。

解説

指定したポイントデータの座標値を取得します。

ロボットの現在位置の座標値を取得する場合は、パラメーターに Here を使います。

参照

CX, CY, CZ, CU, CV, CW, CR, CS, CT

CX, CY, CZ, CU, CV, CW, CR, CS, CT 関数使用例

次の例ではポイント“pick”から X 座標値を取り出し、変数 x にセットします。

```
Function ctest
  Real x
  x = CX(pick)
  Print "The X Axis Coordinate of point 'pick' is", x
End
```

Date

日付の表示を行います。

書式

Date

結果

現在の日付を表示します。

参照

Time, Date\$

Date 使用例

コマンドウィンドウでの実行例

```
> Date  
2009/08/01
```

Date\$ 関数

現在の日付を返します。

書式

Date\$

戻り値

日付を文字列で返します。

書式は、yyyy/mm/dd [西暦年号/月/日]です。

参照

Date, Time, Time\$

Date\$ 関数使用例

```
Print "Today's date: ", Date$
```


Declare

DLL (ダイナミックリンクライブラリー)で定義されている外部ファンクションを呼び出します。

書式

Declare ファンクション名, "DLL ファイルパス", " DLL 内ファンクション名" [, (引数リスト)]
As 関数型

パラメーター

ファンクション名	プログラムから呼び出すときのファンクション名を指定します。
DLL ファイルパス	ライブラリーファイルのパスと名前を、コーテーションマーク ("")で囲んだ文字列か、#define で定義するマクロで指定します。 パスが指定されない場合は RC+はカレントプロジェクトディレクトリーの中のファイルを検索します。見つからない場合は、ファイルは Windows system32 ディレクトリーにあると仮定します。拡張子は省略できますが、省略した場合は、.DLL であると仮定されます。
DLL 内ファンクション名	オプションパラメーターです。DLL の中の実際のファンクション名か、ファンクションインデックスを指定します。名前は大文字と小文字を区別します。ファンクション名はコーテーションマーク ("")で囲んだ文字列で指定します。インデックスを使用する場合はインデックスの前に、#をつけます。省略した場“ファンクション名”パラメーターで指定されたファンクション名を DLL 内ファンクション名として使用します。
引数リスト	DLL 引数のリストです。引数は下記の書式を使用してください。省略可能です。 [{ByRef ByVal}] 変数名[()] As 変数の型 ByRef 呼び出すファンクションの変数を参照する場合は、ByRefを指定します。この場合、ファンクション内での引数の変更を呼び出し側の変数に反映させることができます。参照で渡された値は変更することができます。省略可能です。 ByVal デフォルト設定です。値 (ByVal)によってパラメーターは渡されます。値だけが渡されるので、ファンクションが戻るとき、その変数を変えることはできません。呼び出しているファンクションで、変数の値の変更をしないとき、指定します。省略可能です。 変数名 必須パラメーターです。引数を意味する変数名で、変数の名前をつけるときの規則にしたがい名前をつけます。配列変数を引数として使用する時には、ByRefを必ず指定してください。 変数の型 必須パラメーターです。引数の型を宣言します。
関数型	必須パラメーターです。関数型を宣言してください。

解説

現在のプログラムから DLL ファンクションを呼び出すときに使います。Declare は、ファンクションの外で使用してください。

Declare ステートメントは、コンパイル時に DLL ファイルとファンクションが存在していることを確認します。

数値変数を ByVal で渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a As Long) As Long
VC++ long _stdcall MyDllFunc(long a);
```

文字列変数を ByVal で渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a$ As String) As Long
VC++ long _stdcall MyDllFunc(char *a);
```

数値変数を ByRef で渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a As Long) As Long
VC++ long _stdcall MyDllFunc(long *a);
```

文字列変数を ByRef で渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a$ As String) As Long
VC++ long _stdcall MyDllFunc(char *a);
```

ByRef で文字列を渡すと、DLL の中で文字列を変更することができます。文字列は最長 255 文字まで使えます。最大文字数を超えないように注意してください。SPEL+は文字列変数のために、内部で固定の 255 文字領域を確保します。

数値配列を ByRef で渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a() As Long) As Long
VC++ long _stdcall MyDllFunc(long *a);
```

DLL ファンクションからの戻り値

DLL ファンクションは、文字列(String 型)以外のあらゆるデータ型で戻り値を返すことができます。文字列を返す必要がある場合は、上記の「文字列変数を ByRef で渡す」を参考に、引数として文字列変数を指定して下さい。
戻り値に文字列変数を指定すると、エラー3614:「Declare コマンドで、戻り値に文字列を指定することはできません。」が発生します。

変数の型

EPSON RC+ 7.0 のデータ型と C/C++ の変数型は、下表のように対応しています。
EPSON RC+ 7.0 に対応するデータがないため、C/C++ の byte 型や構造体は使用することができません。

EPSON RC+ 7.0 と C/C++ データ型対応表

EPSON RC+ 7.0	C/C++
Boolean	short
Byte	short
Short	short
Integer	short
Long	int
Real	float
Double	double
String	char [256] * Null を含む

プログラム例

```
Declare ReturnLong, "mystuff.dll", "ReturnLong", As Long

Function main
    Print "ReturnLong = ", ReturnLong
Fend
```

参照

Function...Fend

Declare 使用例

- ・ Declare 命令で外部ファンクションを定義します。
- ・ DLL ファイルパスにフルパスを設定しない場合、カレントプロジェクトフォルダー、
- ・ Windows System32 フォルダーに DLL ファイルを置くことができます。

```
Declare MyDLLTest, "mystuff.dll", "MyDLLTest" As Long

Function main
    Print MyDLLTest
Fend
```

- ・ Declare 命令で 2 つの Integer 型引数を持つ外部ファンクションを定義します。

```
#define MYSTUFF "mystuff.dll"
```

```
Declare MyDLLCall, MYSTUFF, "MyTestFunc", (var1 As Integer, var2 As Integer) As Integer
```

- ・ Declare 命令で外部ファンクションをフルパス指定し、
- ・ ファンクションをインデックスで指定して定義します。

```
Declare MyDLLTest, "c:\mydlls\mystuff.dll", "#1" As Long
```

DegToRad 関数

角度をラジアンに変換します。

書式

DegToRad(角度)

パラメーター

角度 ラジアンに変換する度の値 (実数値)を指定します。

戻り値

Double 型のラジアン値を返します。

参照

ATan, ATan2, RadToDeg 関数

DegToRad 関数使用例

```
s = Cos (DegToRad (x) )
```

Del

ファイルを削除します。

書式

Del ファイル名

パラメーター

ファイル名 削除するパスとファイル名を指定します。ファイル名は拡張子をつけます。
詳細は、ChDisk を参照してください。

解説

指定したファイルを削除します。

Del 使用例

コマンドウィンドウでの操作例

```
> Del TEST.PTS                      'カレントディレクトリーからポイントファイルを削除  
  
> Del c:TEST.PTS                    'NG  
!! エラー: 7213 指定されたファイルが見つかりません。  
> Del c:\TEST.PTS                   'OK
```

DeleteDB

オープンしているデータベース内のテーブルからデータを削除します。

書式

DeleteDB #データベース番号,テーブル名[, 削除条件]

パラメーター

データベース番号	OpenDB で指定したデータベース番号 (501~508 の整数値)を指定します。
テーブル名	データ削除を行うテーブル名を指定します。
削除条件	削除条件を指定します。 AND, OR を使用して複合条件を指定できます。 削除条件を指定しない場合、テーブルの全データが削除されます。

解説

オープンされたデータベースの指定されたテーブルから削除条件に一致するデータを削除します。
オープンされたデータベースが Excel ブックの場合、このコマンドは、実行できません。

注意

RC+ がインストールされている PC の接続が必要です

参照

OpenDB, CloseDB, SelectDB, UpdateDB

DiffPoint 関数

指定された 2 つのポイントの差分を返します。

書式

DiffPoint (ポイントデータ 1, ポイントデータ 2)

パラメーター

ポイントデータ 1 1 つ目のポイントデータを指定します。

ポイントデータ 2 2 つ目のポイントデータを指定します。

戻り値

ポイントデータ 1 から見たポイントデータ 2 の位置姿勢をポイントデータとして返します。

解説

ポイントデータ 1 を原点とした座標系における、ポイントデータ 2 の位置姿勢をポイントデータとして返します。返されるポイントデータのローカル番号や、Hand などのフラグ情報はデフォルト値を返します。

2 つのポイントデータについて、いずれかのポイントデータで、未定義の値がある場合、“0”として計算されます。

例えば、Point1 を“XY (10,0,0,0,0,0): ST (10, 10)”として、Point2 を“XY (10,0,0,0,0,0)”と指定した場合、Point2 では、S および T の値が定義されず、Point1 では定義されるため、Point2 の S および T の値を、“0”として計算した値を返します。

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

DiffPoint 関数使用例

'ポイント P1 から見た P2 の位置姿勢を表示します。

```
Print DiffPoint (P1, P2)
```

'現在位置(Here)から見た P1 の位置姿勢を表示します。

```
Print DiffPoint (Here, P1)
```

DiffToolOrientation 関数

指定された 2 つのポイントで実現されるそれぞれのツール姿勢の変化量を表すため、ツール座標系の各座標軸のなす角度 (単位: 度) を返します。

書式

DiffToolOrientation (ポイントデータ 1, ポイントデータ 2, 軸番号)

パラメーター

ポイントデータ 1	1 つ目のポイントデータを指定します。
ポイントデータ 2	2 つ目のポイントデータを指定します。
軸番号	角度変化量を求めるツール座標系の座標軸を指定します。
	定数 値
	COORD_X_PLUS 1: +X 軸
	COORD_Y_PLUS 2: +Y 軸
	COORD_Z_PLUS 3: +Z 軸
	COORD_ALL 4: 任意軸

戻り値

角度 (0~180 度の実正数値)

解説

指定された 2 つのポイントデータのそれぞれで実現されるツール姿勢がなす姿勢変化量を、指定されたツール座標系の座標軸がなす角度 (0~180 度の実正数値) によって返します。結果はポイントデータ 1, 2 の順番には影響を受けません。また、結果は 2 点間の原点の位置関係 (X, Y, Z の座標値) には影響を受けません。

COORD_ALL を指定した場合、任意軸回りの回転量を返します。任意軸とは、2 つの姿勢 (U, V, W) があるとき、仮想的な軸 (1 本の直線) 回りに、1 度の回転で移動可能である軸のことです。各軸に限定せず、総合的な回転角度を求める場合に使用します。

注意

使用可能なコントローラーについて

T/VT シリーズでは、軸番号に COORD_ALL を指定することはできません。

DiffToolOrientation 関数使用例

・ポイント P1 と P2 のツール座標 Z 軸がなす角度を表示します。

```
Print DiffToolOrientation (P1, P2, COORD_Z_PLUS)
```


DispDev

現在のディスプレイデバイスを設定します。

書式

DispDev (デバイス ID)

パラメーター

デバイス ID 指定するディスプレイデバイスのデバイス ID を指定します。
21 RC+
24 TP (TP1 のみ)
20 TP3

以下の定数を使用することも可能です。

DEVID_SELF 21
DEVID_TP 24
DEVID_TP3 20

参照

DispDev 関数

DispDev 使用例

```
DispDev DEVID_TP
```

DispDev 関数

現在のディスプレイデバイスを返します。

書式

DispDev

戻り値

デバイス ID が設定されているディスプレイデバイスの整数値を返します。

21 RC+

24 TP (TP1 のみ)

20 TP3

参照

DispDev

DispDev 使用例

```
Print "The current display device is ", DispDev
```

Dist 関数

2 つのロボット座標間の距離を返します。

書式

Dist (座標 1, 座標 2)

パラメーター

座標 1, 座標 2 2 つのロボット座標を指定します。

戻り値

2 つの座標間の距離を実数値で返します。(単位: mm)

解説

付加軸を使用している場合でも、ロボットの移動距離のみを返します。
例えば、付加軸を走行軸に使用するケースでも付加軸移動距離を考慮しません。
ジョイント型ロボットでは、本関数が返す値は意味を持ちません。

参照

CU, CV, CW, CX, CY, CZ

Dist 関数使用例

```
Real distance
```

```
distance = Dist(P1, P2)
```

Do...Loop

条件が一致している間、あるいは、不一致の場合は条件が一致するまで、DO...LOOP 間を繰り返します。

書式

```
Do [ { While | Until } 条件式 ]  
    [ステートメント]  
[Exit Do]  
    [ステートメント]  
Loop
```

または、次の書式を使います。

```
Do  
    [ステートメント]  
[ Exit Do ]  
    [ステートメント]  
Loop [ { While | Until } 条件式]
```

Do Loop ステートメントの書式には、条件式とステートメントがあります。

条件式	True または False を表わす数字、または文字列式。条件式が空 (Null) の場合、条件は False として扱われます。省略可能です。
ステートメント	条件が一致している間、あるいは、条件が一致するまで繰り返される、1 つ以上のステートメント

解説

Do...Loop を抜けるもう 1 つの方法として、Do...Loop 中、どこでも、いくつでも Exit Do ステートメントを挿入することができます。Exit Do は、If...Then など、いくつかの条件を評価した後によく用いられます。If...Then で Exit Do ステートメントが用いられると、制御を Loop の次のステートメントに移します。

ネスティングされた Do...Loop ステートメントの中で用いられると、Exit Do は、それが起こったループの 1 レベル上のループに制御を移します。

注意

ループ文の中で XQT 命令を頻繁に繰り返すような使い方はしないでください

Do...Loop などループ文の中で XQT 命令を頻繁に繰り返すような使い方はしないでください。コントローラーがハングアップ状態になる可能性があります。このような使い方をする場合は、Wait 命令 (Wait 0.1) を追加してください。

空の無限ループ、および無限ループと同様の処理は、できる限り Wait 命令と併用してください

空の Do...Loop、および同様の処理は、システムに影響を与える可能性があるため、できるだけ行わないでください。コントローラーでは、無限ループタスクの検出を行っており、システムに影響が出るとコントローラーが判断した場合、エラー 2556 (過剰なループを検出した) が発生することがあります。ループを必要とする演算や、I/O 待ちを行う場合は、Wait 命令 (Wait 0.1) などをループ処理内で実行し、CPU の占有を回避してください。

Exit Do を使用せず、ネスト構造からループを抜けた場合

Exit Do 命令以外のコマンド (Gosub 命令、Goto 命令、Call 命令など) でループから抜けるプログラムを繰り返し実行すると、エラー 2020 が発生します。ループの途中で抜ける場合は、Exit Do 命令で抜けるようにしてください。

参照

For...Next, Select...Send

Do...Loop 使用例

```
Do While Not Lof(1)
    Line Input #1, tLine$
    Print tLine$
Loop
```

Double

Double 型変数を宣言します。(8 バイトの倍精度数)

書式

Double 変数名 [(配列変数の最大要素番号)] [, 変数名[(配列変数の最大要素番号)]...]

パラメーター

変数名 Double 型と宣言する変数の名前を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Double は、指定した変数を Double 型に宣言します。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

Double 型の有効桁数は 14 桁です。

参照

Boolean, Byte, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Double 使用例

下記は、Double を使っていくつかの変数を宣言したプログラム例です。

```
Function doubletest
    Double var1
    Double A(10)           'Double 型の 1 次元配列
    Double B(10, 10)       'Double 型の 2 次元配列
    Double C(5, 5, 5)      'Double 型の 3 次元配列
    Double arrayvar(10)
    Integer i
    Print "Please enter a Number:"
    Input var1
    Print "The variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter a Number:"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

ECP

現在の ECP (外部制御点)の選択または表示をします。

書式

- (1) ECP ECP 番号
- (2) ECP

パラメーター

ECP 番号 次が続く動作命令で、16ある ECP 設定 (整数値 0~15)のうち、どれを使うかを指定します。省略可能です。ECP 0 は、ECP 選択を無効にします。

結果

パラメーターを省略すると、現在設定されている ECP 番号を表示します。

解説

ECP 命令は、ECP 番号で指定した外部制御点を選択します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

電源オフによる ECP 選択への影響

選択された ECP は、主電源をオフすると、選択状態がクリアされます。

参照

ECPSet

ECP 使用例

```
>ecpset 1, 100, 200, 0, 0
>ecp 1
```

ECP 関数

現在指定されている ECP (外部制御点)番号を返します。

書式

ECP

戻り値

現在指定されている ECP 番号を整数値で返します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

ECP

ECP 関数使用例

```
Integer savECP  
  
savECP = ECP  
ECP 2  
Call Dispense  
ECP savECP
```


ECPClr

外部制御点設定をクリア (未設定)します。

書式

ECPClr ECP 番号

パラメーター

ECP 番号 1 から 15 の外部制御点のうち、クリア (未設定)する番号を整数値で指定します。
(ECP の 0 番は初期設定値なので、クリアはできません。)

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet

ECPClr 使用例

```
ECPClr 1
```

ECPDef 関数

外部制御点の設定状態を返します。

書式

ECPDef (ECP 番号)

パラメーター

ECP 番号 状態を呼び出したい ECP 番号を整数値で指定します。

戻り値

指定した ECP 番号の外部制御点が設定されている場合は“True”、設定されていない場合は“False”を返します。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLEClr, TLSet

ECPDef 使用例

```
Function DisplayECPDef(ecpNum As Integer)

    If ECPDef(ecpNum) = False Then
        Print "ECP ", ecpNum, "is not defined"
    Else
        Print "ECP ", ecpNum, ": ",
        Print ECPSet(ecpNum)
    EndIf
Fend
```

ECPSet

外部制御点の定義または表示をします。

書式

- (1) ECPSet ECP 番号, ECP ポイント指定
- (2) ECPSet ECP 番号
- (3) ECPSet

パラメーター

ECP 番号 外部制御点として定義する番号を、式または 1~15 の整数値で指定します。

ECP ポイント指定 P 番号または P(式), ポイントラベル, ポイントデータで指定します。

戻り値

すべてのパラメーターを省略すると、現在の ECP 設定を表示します。

ECP 番号だけを指定すると、指定の ECP 設定を表示します。

解説

外部制御点を設定します。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

ECPSet 使用例

```
ECPSet 1, P1
ECPSet 2, 100, 200, 0, 0
```

ECPSet 関数

指定した ECP 番号に割り当てられている外部制御点のポイントデータを返します。

書式

ECPSet (ECP 番号)

パラメーター

ECP 番号 ポイントデータを呼び出したい ECP 番号を整数値で指定します。

戻り値

指定した ECP 番号の外部制御点のポイントデータを返します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

ECPSet

ECPSet 使用例

```
P1 = ECPSet(1)
```

ElapsedTime 関数

タクトタイム計測用タイマーがスタートしてからの経過時間を単位秒で返します。

書式

ElapsedTime

戻り値

タクトタイム計測用タイマーの経過時間を実数値 (秒単位) で返します。タイマーの範囲は、0~約 1.7E+31 です。タイマー解像度は、0.001 秒です。

解説

タクトタイム計測用タイマーがスタートしてからの経過時間を返します。Tmr 関数と異なり、プログラムが一時的停止状態となっている時間は経過時間としてカウントされません。

タクトタイム計測用タイマーは ResetElapsedTime でリセットできます。

```
Real overhead

ResetElapsedTime
overHead = ElapsedTime
```

参照

ResetElapsedTime, Tmr 関数

ElapsedTime 使用例

```
ResetElapsedTime      'タクトタイム計測用タイマーをリセット
For i = 1 To 10        '10回実行
    GoSub Cycle
Next
Print ElapsedTime / 10 'タクトタイムを計算して表示
```

Elbow

ポイントの肘姿勢を設定します。

書式

- (1) Elbow ポイント指定 [, 設定値]
- (2) Elbow

パラメーター

ポイント指定 P 番号、P(式) またはポイントラベルを指定します。

設定値 整数または式を指定します。

1 = Above (/A)

2 = Below (/B)

戻り値

2つのパラメーターが両方省略されると、ロボットの現在位置の肘姿勢を表示します。

設定値パラメーターが省略されると、指定したポイントの肘姿勢を表示します。

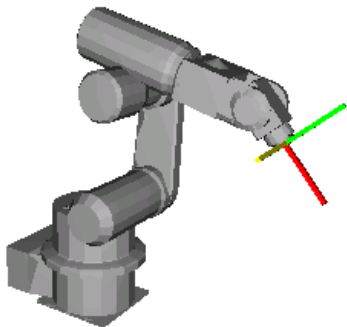
参照

Elbow 関数, Hand, J4Flag, J6Flag, Wrist

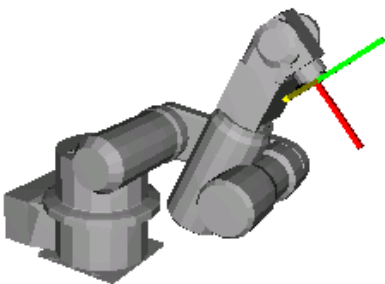
Elbow 使用例

```
Elbow P0, Below
Elbow pick, Above
Elbow P(myPoint), myElbow
```

```
P1 = 0.000, 490.000, 515.000, 90.000, -40.000, 180.000
```



```
Elbow P1, Above
Go P1
```



```
Elbow P1, Below
Go P1
```

Elbow 関数

ポイントの肘姿勢を返します。

書式

Elbow [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。
省略可能で、省略するとロボットの現在位置の肘姿勢を返します。

戻り値

- 1 Above (/A)
- 2 Below (/B)

参照

Elbow, Hand, Wrist, J4Flag, J6Flag

Elbow 関数使用例

```
Print Elbow(pick)
Print Elbow(P1)
Print Elbow
Print Elbow(P1 + P2)
```

Eof 関数

ファイルの EOF (オープンしたファイルのポインターが終端にある)を返します。

書式

Eof (ファイル番号)

パラメーター

ファイル番号 30~63 の整数値、または式で指定します。

戻り値

ファイルが EOF の場合は“True”、それ以外は“False”を返します。

解説

ファイルを読み出しモードで開いている場合に機能します。

AOpen, WOpen コマンドで開いたファイルの場合には、エラーになります。

参照

Lof

Eof 関数使用例

```
Integer fileNum
String data$

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Do While Not Eof(fileNum)
    Line Input #fileNum, data$
    Print "data = ", data$
Loop
Close #fileNum
```


Era 関数

エラーが発生した関節の番号を返します。

書式

Era [(タスク番号)]

パラメーター

タスク番号 0~32 のタスク番号を整数で指定します。
省略または“0”の場合は現在のタスクです。

戻り値

エラーが発生した関節番号を下記の 0~9 の整数値で知らせます。

- 0 -現在のエラーは、関節が原因ではありません。
- 1 -現在のエラーは、第 1 関節が原因です。
- 2 -現在のエラーは、第 2 関節が原因です。
- 3 -現在のエラーは、第 3 関節が原因です。
- 4 -現在のエラーは、第 4 関節が原因です。
- 5 -現在のエラーは、第 5 関節が原因です。
- 6 -現在のエラーは、第 6 関節が原因です。
- 7 -現在のエラーは、第 7 関節が原因です。
- 8 -現在のエラーは、第 8 関節 (付加軸 S)が原因です。
- 9 -現在のエラーは、第 9 関節 (付加軸 T)が原因です。

解説

Era は、エラーが発生したとき、そのエラーがどの関節で発生したかを見つけ、その関節番号を知らせます。もし、関節が現在のエラーの原因でないときは、“0”を返します。

自動運転モード(AUTO)の通常タスクと NoPause タスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。

NoEmgAbort タスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、“エラー 2261”が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

参照

Erl, Err, ErrMsg\$, Ert, OnErr, Trap

Era 関数使用例

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

EResume

エラー処理ルーチンの終了後に、プログラムの実行を再開します。

書式

EResume [{ラベル | Next }]

解説

EResume

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、エラーの原因となったステートメントからプログラムの実行が再開されます。

呼び出されたファンクション内でエラーが発生した場合、エラー処理ルーチンを含むファンクション内の Call ステートメントからプログラムの実行が再開されます。

EResume Next

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、エラーの原因となったステートメントの次のステートメントから、プログラムの実行が再開されます。

呼び出されたファンクション内でエラーが発生した場合、エラー処理ルーチンを含むファンクションが呼び出した最後の Call ステートメントの次のステートメントから、プログラムの実行が再開されます。

EResume {ラベル}

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、指定したラベルを含むステートメントから、プログラムの実行を再開します。

参照

OnErr

EResume 使用例

```
Function main
  Integer retry

  OnErr GoTo eHandler
  Do
    RunCycle
  Loop
  Exit Function

eHandler:
  Select Err
  Case MyError
    retry = retry + 1
    If retry < 3 Then
      EResume '再実行
    Else
      Print "MyError has occurred ", retry, " times
    EndIf
  Send
Fend
```

Err\$関数

エラーが発生した関数名を返します。

書式

Err\$ [(タスク番号)]

パラメーター

タスク番号 タスク番号を 0~32 の整数値で指定します。
省略または“0”の場合は現在のタスクです。

戻り値

最後のエラーが発生した関数名を返します。

解説

Err\$は、OnErrorとともに使います。Err\$は、エラーが発生した関数名を返します。Err\$を Err, Ert, Erl, Era などと組み合わせることで、発生したエラーについてより詳しい情報を集めることができます。

自動運転モード(AUTO)の通常タスクと NoPause タスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。

NoEmgAbort タスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、“エラー 2261”が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

参照

Era, Erl, Err, ErrMsg\$, Ert, OnErr, Trap

Err\$ 関数 使用例

下記は、次の内容を調べる簡単なプログラムです。

どのタスクでエラーが発生したか (Ert 関数)

どの関数でエラーが発生したか (Err\$関数)

どこで発生したか (Erl 関数)

どの関節でエラーが発生したか (Era 関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "Function at which error occurred is ", Err$(errTask)
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
End
Fend
```

Erl 関数

エラーが発生した行番号を返します。

書式

Erl [(タスク番号)]

パラメーター

タスク番号 タスク番号を 0~32 の整数値で指定します。
省略または“0”の場合は現在のタスクです。

戻り値

最後のエラーが発生した行番号を返します。

解説

Erl は OnErr とともに使います。Erl は、エラーが発生した行番号を返します。Erl を Err, Ert, Era などと組み合わせることで、発生したエラーについて詳しい情報を集めることができます。

自動運転モード(AUTO) の通常タスクと NoPause タスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。

NoEmgAbort タスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、“エラー 2261”が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

参照

Era, Erf\$, Err, ErrMsg\$, Ert, OnErr

Erl 関数使用例

下記は、次の内容を調べる簡単なプログラムです。

どのタスクでエラーが発生したか (Ert 関数)
どこで発生したか (Erl 関数)
どんなエラーが発生したか (Err 関数)
どの関節でエラーが発生したか (Era 関数)

```
Function main
  OnErr Goto eHandler
Do
  Call PickPlace
Loop
Exit Function
eHandler:
Print "The Error code is ", Err
Print "The Error Message is ", ErrMsg$(Err)
errTask = Ert
If errTask > 0 Then
  Print "Task number in which error occurred is ", errTask
  Print "The line where the error occurred is Line ", Erl(errTask)
  If Era(errTask) > 0 Then
    Print "Joint which caused the error is ", Era(errTask)
  EndIf
EndIf
Fend
```

Err 関数

最新のエラーステータスを返します。

書式

Err [(タスク番号)]

パラメーター

タスク番号 タスク番号を 0~32 の整数値で指定します。“0”は、現在のタスクを指定します。省略可能です。

戻り値

エラーコードを整数値で返します。

解説

Err 関数は、ユーザーに現在のエラーコードを知らせます。SPEL+のエラー対応機能とともにどのエラーが発生したか知らせ、適切に対応できるようにします。Err は OnErr とともに用います。

コントローラーのエラーを取得するには、SysErr 関数を使用します。

自動運転モード(AUTO)の通常タスクと NoPause タスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。

NoEmgAbort タスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、“エラー 2261”が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

参照

Era, Erf\$, Erl, ErrMsg\$, EResume, Ert, OnErr, Return, SysErr

Err 使用例

下記の例は、ポイント P0-P399 があるかどうかをチェックする、簡単なユーティリティープログラムです。ポイントがなければ、それをユーザーに知らせるメッセージが画面に表示されます。各ポイントが定義されているか、CX 命令を使ってテストしています。定義されていないポイントがあると、エラー処理に制御が移り、画面にそのポイントが定義されていないと表示します。

```
Function errtest
  Integer i, errnum
  Real x

  OnErr GoTo eHandle
  For i = 0 To 399
    x = CX(P(i))
  Next i
  Exit Function
,
,
,
' *****
' * Error Handler *
' *****
eHandle:
  errnum = Err
  ' 未定義ポイントを使用していないか確認
  If errnum = 78 Then
    Print "Point number P", i, " is undefined!"
  Else
    Print "ERROR: Error number ", errnum, " Occurred."
  EndIf
  EResume Next
Fend
```

Errb 関数

エラーが発生したロボットの番号を返します。

書式

Errb

戻り値

エラーが発生したロボット番号を返します。

解説

Errb 関数は、エラーが発生したとき、そのエラーがどのロボットで発生したかを見つけ、そのロボット番号を知らせます。もしロボットが現在のエラーの原因でないときは、“0”を返します。

参照

Era, Erl, Err, ErrMsg\$, OnErr, Trap

Errb 関数使用例

次のプログラム例では下記の内容を表示します。

どのタスクでエラーが発生したか (Ert 関数)

どこで発生したか (Erl 関数)

どんなエラーが発生したか (Err 関数)

どの関節でエラーが発生したか (Era 関数)

どのロボットでエラーが発生したか (Errb 関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
    Print "Robot number in which error occurred is ", errb
  EndIf
Fend
```

ErrMsg\$関数

指定したエラー番号のエラーメッセージを返します。

書式

ErrMsg\$(エラー番号, 言語番号)

パラメーター

エラー番号 メッセージを返すエラー番号を整数値で指定します。

言語番号 言語を次の整数値で指定します。省略可能です。

0 - 英語

1 - 日本語

2 - ドイツ語

3 - フランス語

4 - 中国語 (簡体字)

5 - 中国語 (繁体字)

6 - スペイン語

省略した場合、英語が指定されます。

戻り値

エラーコード表のエラーメッセージを返します。

参照

Era, Erl, Err, Ert, OnErr, Trap

ErrMsg\$関数使用例

下記は、次の内容を調べる簡単なプログラムです。

どのタスクでエラーが発生したか (Ert 関数)

どこで発生したか (Erl 関数)

関節でエラーが発生したか (Era 関数)

```
Function main
  OnErr Goto eHandler
Do
  Call PickPlace
Loop
Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
End
Fend
```

Error

ユーザー定義エラーを発生させます。

書式

- (1) Error タスク番号, エラー番号
- (2) Error エラー番号

パラメーター

タスク番号	タスク番号を 0~32 の整数値で指定します。“0”は、現在のタスクを指定します。省略可能です。
エラー番号	エラー番号を整数値で指定します。ユーザーエラー番号は 8000~8999 です。

解説

Error ステートメントは、システムまたはユーザー定義エラーを発生させます。EPSON RC+ 開発環境のユーザーエラーエディターを使って、ユーザーエラーのラベルと記述を定義できます。

参照

Era, Erl, Err, OnErr

Error 使用例

```
#define ER_VAC 8000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```


ErrorOn 関数

コントローラーのエラー状態を返します。

書式

ErrorOn

戻り値

エラー状態のときは“True”、それ以外の場合は“False”を返します。

解説

本関数は、NoEmgAbort タスク (Xqt 時に NoEmgAbort を指定して開始された特別なタスク)とバックグラウンドタスクでのみ使用します。

参照

ErrorOn, SafetyOn, SysErr, Wait, Xqt

ErrorOn 関数 使用例

下記例は、コントローラーがエラー状態になるのを監視し、エラーが発生した場合、エラー番号に応じて I/O をオンオフするプログラムです。

注意

Forced フラグ

本プログラム例では、On/Off 命令に Forced フラグを指定しています。
エラー発生中、および、非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

```
Function main
    Xqt ErrorMonitor, NoEmgAbort
    :
    :
Fend

Function ErrorMonitor
    Wait ErrorOn
    If 4000 < SysErr Then
        Print "Mortion Error = ", SysErr
        Off 10, Forced
        On 12, Forced
    Else
        Print "Other Error = ", SysErr
        Off 11, Forced
        On 13, Forced
    EndIf
Fend
```

Ert 関数

エラーが発生したタスク番号を返します。

書式

Ert

戻り値

エラーが発生したタスク番号を返します。

解説

Ert 関数は、どのタスクでエラーが発生したかを取得するために使います。

エラーが発生したタスクなし (0), 通常タスク (1~32), バックグラウンドタスク (65~80), TRAP タスク (257~267)の番号を返します。

参照

Era, Erl, Err, ErrMsg\$, OnErr, Trap

Ert 関数使用例

次のプログラム例では下記の内容を表示します。

どのタスクでエラーが発生したか (Ert 関数)

どこで発生したか (Erl 関数)

どの関節でエラーが発生したか (Era 関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
Exit Function
eHandler:
Print "The Error code is ", Err
Print "The Error Message is ", ErrMsg$(Err)
errTask = Ert
If errTask > 0 Then
  Print "Task number in which error occurred is ", errTask
  Print "The line where the error occurred is Line ", Erl(errTask)
  If Era(errTask) > 0 Then
    Print "Joint which caused the error is ", Era(errTask)
  EndIf
EndIf
Fend
```

EStopOn 関数

非常停止状態を返します。

書式

EstopOn

戻り値

非常停止状態のときは“True”、それ以外のときは“False”を返します。

解説

本関数は、NoEmgAbort タスク (Xqt 時に NoEmgAbort を指定して開始された特別なタスク)でのみ使用します。

参照

ErrorOn, SafetyOn, Wait, Xqt

EstopOn 関数使用例

下記例は、コントローラーが非常停止になるのを監視し、非常停止が発生した場合 I/O をオンオフするプログラムです。

注意

Forced フラグ

本プログラム例では、On/Off 命令に Forced フラグを指定しています。

エラー発生中、および、非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

非常停止で出力ポートを OFF

本例のように、非常停止後も I/O のオンオフをタスクが行う場合は、[セットアップ]-[システム設定]-[コントローラー]-[環境設定]-[非常停止で出力ポートを OFF]のチェックをはずすことを推奨します。この設定がチェックされたままの場合、コントローラーによる I/O のオフとタスクによる I/O のオンのどちらが先に実行されるか保証されません。

```
Function main
    Xqt EStopMonitor, NoEmgAbort
    :
    :
Fend

Function EStopMonitor
    Wait EStopOn
    Print "EStop !!!"
    Off 10, Forced
    On 12, Forced
Fend
```

Eval 関数

コマンドウィンドウのステートメントを実行し、エラー状態を返します。

書式

Eval(コマンド [, コマンドの出力結果])

パラメーター

コマンド	実行するコマンドを文字列で指定します。
コマンドの出力結果	コマンドの出力結果を格納する文字列変数を指定します。省略可能です。 コマンドがエラーとなった場合は“!Error: エラーコード”が返されます。 出力結果が 255 文字を超えた場合、超えた結果は切り捨てられます。

戻り値

コマンドの実行によって戻されるエラーコードを返します。
コマンド実行がエラーとなっても、本関数そのものはエラーとなりません。また、システムログにもエラーは残りません。
コマンドが正常終了した場合は“0”を返します。

解説

Eval 関数を使用すると、TCP/IP などの通信ポートから任意のコマンドを実行することができます。
実行できるコマンドは、コマンドウィンドウから実行可能なコマンドです。
本関数の実行には、通常のステートメント実行より処理時間がかかります。

コマンドからの戻り値を得るためには、コマンドの出力結果パラメーターを使います。例えば、“Print Sw(1)”というコマンドに対し、コマンドの出力結果は、“1”あるいは“0”が返されます。

参照

エラーメッセージ一覧

Eval 関数使用例

下記の例では、RS-232C で読まれるコマンドをどのように実行するか示しています。コマンドが実行された後、エラーコードがホストに返されます。例えば、ホストからは“motor on”などのコマンドが送られます。

```
Integer errCode
String cmd$

OpenCom #1
Do
    Line Input #1, cmd$
    errCode = Eval (cmd$)
    Print #1, errCode
Loop
```

Exit

ループまたはファンクションを強制終了します。

書式

Exit { Do | For | Function }

解説

Exit ステートメントの書式には、下記があります。

ステートメント	解説
Exit Do	Do...Loop ステートメントから抜けます。これは、Do...Loop ステートメント内でのみ使うことができます。Exit Do で、制御は、Loop ステートメントの次のステートメントに移ります。ネスティングされた Do...Loop ステートメント内で用いると、Exit Do のあるループの 1 レベル上のループに制御が移ります。
Exit For	For ループから抜けます。For...Next ループ内でのみ使うことができます。Exit For は、Next ステートメントの次のステートメントに制御を移します。ネスティングされた For ループで使われると、Exit For のあるループの 1 レベル上のループに制御が移ります。
Exit Function	任意の位置でファンクションから抜きたいときに使います。プログラムの実行は、ファンクションを呼び出したステートメントの次のステートメントに続きます。

参照

Do...Loop, For...Next, Function...Fend

Exit 使用例

```

For i = 1 To 10
  If Sw(1) = On Then
    Exit For
  EndIf
  Jump P(i)
Next i

```

ExportPoints

ポイントファイルを指定のパスにエクスポートします。

書式

ExportPoints ファイル名, 保存先

パラメーター

ファイル名	エクスポートしたい特定のファイルを表す文字列式 拡張子は、“ .pts ”です。パスを指定することはできません。また、ChDisk などの影響を受けません。詳細は、ChDisk を参照してください。
保存先	保存先のパスとファイル名を指定します。拡張子は、“ .pts ”です。 詳細は、ChDisk を参照してください。

解説

ExportPoints は、指定したポイントファイルを、PC 上のフォルダーにコピーします。
PC 上のフォルダーに同じファイルが既に存在していた場合は上書きされます。

起こりやすいエラー

保存先パスが存在しない場合

指定した保存先パスが存在しない場合、エラーが発生します。

指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。

参照

Dir, LoadPoints, SavePoints, FileExists, FolderExists

ExportPoints 使用例

```
Function main
  LoadPoints "robot1.pts"
  :
  SavePoints "robot1.pts"
  If FolderExists("c:\mypoints\") Then
    ExportPoints "robot1.pts", "c:\mypoints\model1.pts"
  EndIf
Fend
```

FbusIO_GetBusStatus 関数

指定したフィールドバスの状態を返します。

書式

FbusIO_GetBusStatus(バス番号)

パラメーター

バス番号 フィールドバスシステム番号を表す整数式
この番号は必ず 16 で、コントローラーの PC 側にあるフィールドバスマスターボードに接続されているバスの ID です。

戻り値

0 - OK
1 - 未接続
2 - パワーオフ

解説

FbusIO_GetBusStatus 関数では、フィールドバスの状態を確認することができます。

注意

このコマンドは、フィールドバスマスターオプションが有効のときのみ使用できます。

参照

FbusIO_GetDeviceStatus, FbusIO_SendMsg

FbusIO_GetBusStatus 関数使用例

```
Long sts  
sts = FbusIO_GetBusStatus(16)
```

FbusIO_GetDeviceStatus 関数

指定したフィールドバスデバイスの状態を返します。

書式

FbusIO_GetDeviceStatus(バス番号, デバイス ID)

パラメーター

バス番号	フィールドバスシステム番号を表す整数式 この番号は必ず 16 で、コントローラーの PC 側にあるフィールドバスマスターボードに接続されているバスの ID です。
デバイス ID	デバイスのフィールドバス ID を表す整数式

戻り値

- 0 - OK
- 1 - 未接続
- 2 - パワーオフ
- 3 - 同期エラー: デバイスが初期化中、またはデバイスのボーレートが正しくありません。

解説

FbusIO_GetDeviceStatus 関数では、フィールドバスデバイスの状態を確認することができます。

注意

このコマンドは、フィールドバスマスターオプションが有効時のみ使用できます。

参照

FbusIO_GetBusStatus, FbusIO_SendMsg

FbusIO_GetDeviceStatus 関数使用例

```
Long sts  
sts = FbusIO_GetDeviceStatus(16, 10)
```


FbusIO_SendMsg

フィールドバス I/O デバイスに、メッセージを送り、回答を返します。

書式

FbusIO_SendMsg(バス番号, デバイス ID, msgParam, sendData(), recvData())

パラメーター

バス番号	フィールドバスシステム番号を表す整数式 この番号は必ず 16 で、コントローラーの PC 側にあるフィールドバスマスターボードに接続されているバスの ID です。
デバイス ID	デバイスのフィールドバス ID を表す整数式
msgParam	メッセージパラメーターを表す整数式 DeviceNet には使用できません。
sendData	デバイスに送信されるデータを Byte 型の配列で指定します。この配列の次元は、送信されるバイト数の次元と同じでなくてはなりません。データを送信しない場合は、“0”を指定します。
recvData	デバイスから受信されるデータを Byte 型の配列で指定します。この配列は自動的に受信したバイト数に応じた次元数に変換されます。

解説

FBusIO_SendMsg は、フィールドバス I/O デバイスに対するクエリーとして用います。サポートするメッセージについては、機器メーカーにお問合わせください。

注意

このコマンドは、フィールドバスマスターオプションが有効時のみ使用できます。

参照

FbusIO_GetBusStatus, FbusIO_GetDeviceStatus

FbusIO_SendMsg 使用例

```
' DeviceNet デバイスに明示メッセージを送る
Byte sendData(5)
Byte recvData(0)
Integer i

sendData(0) = &H0E ' コマンド
sendData(1) = 1     ' クラス
sendData(3) = 1     ' インスタンス
sendData(5) = 7     ' 属性
' DeviceNet の msgParam は“0”
FbusIO_SendMsg 16, 1, 0, sendData(), recvData()
' 回答を表示
For i = 0 to UBound(recvData)
    Print recvData(i)
Next i

' Profibus デバイスにメッセージを送る
Byte recvData(0)
Integer i

' msgParam はサービス番号
FbusIO_SendMsg 16, 1, 56, 0, recvData()
' 回答を表示
For i = 0 to UBound(recvData)
    Print recvData(i)
Next i
```

FileDateTime\$関数

ファイルの日付と時間を返します。

書式

FileDateTime\$(ファイル名)

パラメーター

ファイル名 確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDisk を参照してください。

注意

ネットワークパスが使用できます。

戻り値

ファイルの最終修正日時を下記のフォーマットで返します。

月/日/西暦年号 時間:分:秒

参照

FileExists, FileLen

FileDateTime\$ 関数使用例

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

FileExists 関数

ファイルの有無をチェックします。

書式

FileExists(ファイル名)

パラメーター

ファイル名 確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDisk を参照してください。

注意

ネットワークパスが使用できます。

戻り値

ファイルが存在する場合	True
ファイルが存在しない場合	False

参照

FolderExists, FileLen, FileDateTime\$

FileExists 関数使用例

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

FileLen 関数

ファイルサイズを返します。

書式

FileLen(ファイル名)

パラメーター

ファイル名 確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDisk を参照してください。

注意

ネットワークパスが使用できます。

戻り値

ファイルのバイト数を返します。

参照

FileDateTime\$, FileExists

FileLen 関数使用例

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

Find

動作命令中に座標を保存するための条件の設定と表示をします。

書式

Find [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子(=, <, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

関数 : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr
GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out,
AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

変数 : Byte, Inr32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ
変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

演算子: And, Or, Xor

<例> Find Sw(5) = On

Find Sw(5) = On And Sw(6) = Off

解説

Find ステートメントは、単独で記述するか、動作命令ステートメントの修飾子として記述してください。

Find 条件式には、上記の関数を 1 つ以上含んでいなくてはなりません。

Find 条件式に変数が含まれたときは、その値は、Find 条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。複数の Find ステートメントを記述することもできます。その場合、最後に実行された Find 条件が有効となります。

パラメーターを省略すると、現在の Find 設定が表示されます。

注意**電源オン時の Find 設定**

電源オン時の Find 条件の初期設定は、Find Sw(0)= On です。入力ビット番号 0 がオンのとき、座標を保存するように設定されています。

Find 条件成立をチェックする PosFound 関数

Find 修飾子を用いた動作コマンドを実行した後で、Find 条件が成立したかどうかを、PosFound 関数を用いてチェックすることができます。

イベント条件式で変数を使用する場合

- 使用できる変数型は整数型(Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
 - 配列変数は、使用できません。
 - ローカル変数は、使用できません。
 - 変数値が 0.01 秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
 - システム内で使用できる変数待ちの数に制限があります。1 システム内で使用できる変数待ちの個数は最大 64 個です (Wait などのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
 - 変数待ちを行う変数を Byref で参照渡しするとエラーになります。
 - イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。
-

参照

FindPos, Go, Jump, PosFound, SF_GetStatus

Find 使用例

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```

FindPos 関数

動作命令実行中に Find で保存した座標を返します。

書式

FindPos

戻り値

動作コマンド実行中に Find で保存した座標を返します。

参照

Find, Go, Jump, PosFound, CurPos, InPos

FindPos 関数使用例

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```


Fine

目標位置の位置決め終了判断範囲の設定、表示をします。

書式

- (1) Fine 第 1 関節設定値, 第 2 関節設定値, 第 3 関節設定値, 第 4 関節設定値 [, 第 5 関節設定値, 第 6 関節設定値] [, 第 7 関節設定値] [, 第 8 関節設定値, 第 9 関節設定値]
 (2) Fine

パラメーター

第 1 関節設定値	第 1 関節の位置決め許容範囲を、0~65535 の整数で指定します。
第 2 関節設定値	第 2 関節の位置決め許容範囲を、0~65535 の整数で指定します。
第 3 関節設定値	第 3 関節の位置決め許容範囲を、0~65535 の整数で指定します。
第 4 関節設定値	第 4 関節の位置決め許容範囲を、0~65535 の整数で指定します。
第 5 関節設定値	第 5 関節の位置決め許容範囲を、0~65535 の整数で指定します。 省略可能です。垂直 6 軸型ロボット (N シリーズを含む) のみに使用します。
第 6 関節設定値	第 6 関節の位置決め許容範囲を、0~65535 の整数で指定します。 省略可能です。垂直 6 軸型ロボット (N シリーズを含む) のみに使用します。
第 7 関節設定値	第 7 関節の位置決め許容範囲を、0~65535 の整数で指定します。 省略可能です。ジョイント型 7 軸ロボット のみに使用します。
第 8 関節設定値	第 8 関節の位置決め許容範囲を、0~65535 の整数で指定します。 省略可能です。付加軸 S 関節 のみに使用します。
第 9 関節設定値	第 9 関節の位置決め許容範囲を、0~65535 の整数で指定します。 省略可能です。付加軸 T 関節 のみに使用します。

*C8, C12 シリーズの場合、位置決め許容範囲は 0~131070 の整数です。

結果

パラメーターを指定しないときは、Fine は、各関節の現在の設定値を表示します。

解説

Fine は、各関節に対し、動作終了時に位置決めを確認する動作命令で、指定位置に対して位置決め終了と判断する許容範囲を指定します。

この位置決め終了判断は、CPU からサーボシステムへの、目標座標パルスを送信が終わってから始まります。サーボのディレイにより、その段階ではロボットは目標位置に到達していません。判断は、各関節の設定された許容範囲まで、毎数ミリ秒ごとに行われます。すべての関節が指定された許容範囲内になると位置決め終了となります。位置決めが終了すると、プログラム制御は次のステートメントに移りますが、サーボシステムは引き続き目標位置へロボットを制御します。

Fine 命令で比較的大きい範囲が指定されると、位置決めは比較的早い段階で終了し、次のステートメントに進みます。

Fine のデフォルト設定は、ロボットタイプにより異なります。詳細はマニピュレーターマニュアルを参照してください。

注意

サイクルタイムと Fine 命令

Fine 値自体は、アームの加速または減速制御に影響しませんが、細かい Fine 値が設定されると、サーボがその許容範囲を達成するまで時間 (数ミリ秒)がかかります。システムのサイクルタイムは、その時間だけ遅くなります。アームが Fine 命令の設定する許容範囲内に位置決めされると、CPU は次の命令を実行します。

Fine の初期化 (Motor On, SLock, SFree などの命令による)

下記のいずれかのコマンドが使われると、Fine の設定値は、デフォルト値に初期化されます。

SLock, SFree, Motor

これらのコマンドが実行された後では、かならず Fine 値を再設定してください。

起こりやすいエラー

Fine による位置決めが 2 秒以内に終了しないと、エラーコード 4024 になります。このエラーは、通常、サーボシステムのバランス調整が必要という意味です。

参照

Accel, AccelR, AccelS, Arc, Go, Jump, Move, Speed, SpeedR, SpeedS, Pulse, FineDist, FineStatus

Fine 使用例

下記のプログラムは、プログラムファンクションから Fine を実行する例と、モニターウィンドウから Fine を実行する例です。

```
Function finetest
    Fine 5, 5, 5, 5          '精度を+/-5 パルスに設定
    Go P1
    Go P2
Fend

> Fine 10, 10, 10, 10
>
> Fine
10, 10, 10, 10
```

Fine 関数

指定した関節の Fine 設定を返します。

書式

Fine(関節番号)

パラメーター

関節番号 Fine 設定を取得する関節番号を整数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

実数値を返します。

参照

Accel, AccelS, Arc, Go, Jump, Move, Speed, SpeedS, Pulse

Fine 関数使用例

下記は Fine 関数を使用したプログラム例です。

```
Function finetst
  Integer a
  a = Fine(1)
Fend
```

FineDist

目標位置の位置決め終了判断範囲の設定、表示をします。設定値の単位は mm です。

書式

- (1) FineDist 設定値
- (2) FineDist

パラメーター

設定値 位置決め許容範囲は 0.001[mm]~10[mm]です。

結果

パラメーターを指定しない場合、FineDist は現在の設定値を表示します。

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

Fine と FineDist について

Fine と FineDist の違いは、ロボットの動作終了時の位置決め判断の単位です。

Fine は pulse で位置決め判断値を設定でき、軸ごとに位置決め判断を行います。

FineDist は mm で位置決め判断値を設定でき、ツール番号 0 の座標系で位置決め判断を行います。

Fine と FineDist の両方を使用することはできません。下記のように Fine と FineDist の 2 つをプログラムに使用した場合、FineDist で位置決め判断を行います。

(Fine と FineDist の順番を逆にした場合は、Fine で位置決め判断を行います。)

```
Function test
  Fine 5, 5, 5, 5
  FineDist 0.1

  Go P1
  Go P2
Fend
```

注意**FineDist の初期化 (Motor On, SLock, SFree などの命令による)**

下記のいずれかのコマンドが使われると、FineDist の設定値は、デフォルト値に初期化され、位置決め判断は Fine で行います

SLock, SFree, Motor

これらのコマンドが実行された後では、かならず FineDist 値を再設定してください。

起こりやすいエラー

FineDist による位置決めが 2 秒以内に終了しないと、エラーコード 4024 になります。このエラーは、通常、サーボシステムのバランス調整が必要という意味です。

参照

Accel, AccelR, AccelS, Arc, Go, Jump, Move, Speed, SpeedR, SpeedS, Pulse, Fine, FineStatus

FineDist 使用例

下記のプログラムは、プログラムファンクションから FineDist を実行する例と、モニターウィンドウから Fine を実行する例です。

```
Function fineDistttest
    Fine 0.1    '精度を+/-0.1mm に設定
    Go P1
    Go P2
Fend
```

```
> FineDist 0.1
>
> FineDist
0.1
```

FineStatus 関数

Fine を使用しているか、FineDist を使用しているかを整数値で返します。

書式

FineStatus

戻り値

Fine を使用しているか、FineDist を使用しているかを整数値で返します。

0 = Fine を使用中

1 = FineDist を使用中

参照

Fine, FineDist

FineStatus 関数使用例

```
Print FineStatus
```

Fix 関数

実数値から整数部分を取り出します。

書式

Fix(数値)

パラメーター

数値 実数値を指定します。

戻り値

パラメーターで設定した実数値の整数の値を返します。

参照

Int

Fix 関数使用例

```
>print Fix(1.123)
1
>
```

Flush

バッファをファイルへ書き込みます。

書式

Flush #ファイル番号

パラメーター

ファイル番号 30~63 の整数値、または式で指定します。

解説

バッファを現在開いているファイルへ書き込みます。
ROpen コマンドで開いたファイルの場合は使用できません。

Flush 使用例

```
Integer fileNum, i

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Flush #fileNum
Close #fileNum
```


FmtStr

数値式、または日付/時刻式をフォーマットします。

書式

FmtStr 書式指定する式, 書式指定文字列, 出力文字列変数

パラメーター

書式指定する式	書式指定する数値式、または日付/時刻式を指定します。 日付/時刻式は、“yyyy/mm/dd hh:nn:ss”で指定してください。
書式指定文字列	書式指定文字列を指定します。
出力文字列変数	出力文字列変数を指定します。

解説

書式設定文字列にしたがって、書式設定された文字列を返します。

数値書式指定子

文字 解説

None	フォーマットなしでそのまま番号を表示します。
(0)	桁位置を示す桁表示子です。数値またはゼロを表示します。数値式に含まれる 10 進数を、書式指定文字列の“0”のところに表示します。それ以外の場合は、その位置に“0”を表示します。数値の桁数が、書式指定文字列の (小数点の前後いずれかにおける)“0”の数より少ない場合は、先頭ないしは後方の“0”をいくつか表示します。もし、数値の小数点の右側の桁数が、書式指定文字列の小数点の右側にある“0”の数より多い場合は、“0”の数に合わせて数値の桁数を丸めます。また、数値の小数点左側の桁数が、書式指定文字列の小数点左側にある“0”の数より多い場合は、その余分な桁をそのまま表示します。
(#)	桁位置を示す桁表示子です。数値を示すか、何も表示しないかのいずれかを行います。数値式の 10 進数を書式指定文字列の“#”の位置に表示しますが、それ以外の場合は、その位置に何も表示しません。この文字(#)は、0 桁表示子のような役目をしますが、数値の桁数が書式指定文字列の小数点の左右にある“#”の数と同じか、それより少ない場合は、先頭ないしは後方の“0”は表示されません。
(.)	小数点位置を表示し、小数点の左右にいくつの桁を表示するか設定します。ローカルによっては、カンマで小数点を示すものもあります。書式指定文字列中、この(.)の左に数字しかない場合は、1 より小さい数字が小数点をあたりに始まり、表示されます。先頭のゼロを表示するには、“0”を小数点の左に使います。書式指定されて出力されてくる値で小数点を示す文字は、使用する Windows が認識する数値フォーマットによります。
(,)	数値を 1000 で区切る千桁区切り子です。ローカルによっては、ピリオドを用います。小数点の左に 4 桁以上ある数値で、千ずつ区切るのに使います。標準的な使い方は、書式指定文字列が左右に桁表示子 (0 または #)をとまう千桁区切り子 (,)を含んでいると指定されます。小数の指定があるなしにかかわらず、小数点のすぐ左に、千桁区切り子が 2 つ並んでいる場合、または単独で使われている場合、「その数値を 1000 で割って、必要ならば四捨五入せよ」という書式指定を意味します。例えば、書式指定文字列 “##0,,” は、1 億を“100”と表示せよ、という意味になります。百万より小さい数値は“0”と表示されます。小数点のすぐ左隣以外の位置に千桁区切り子が 2 つ並んで置かれると、単に千桁を区切るために使われます。書式指定された出力値で実際に千桁区切り子として使われる文字は、使用する Windows が認識する数値フォーマットによります。

日付/時間書式指定子

キャラクター 説明

(:)	時刻区切りです。ローカルによっては、他の文字が用いられることがあります。時刻区切りは、時間値が書式指定されたとき、時、分、秒の値を区切ります。書式指定された出力で実際に使われる文字は、Windows の設定によって異なります。
(/)	日付区切り子です。ローカルによっては、他の文字が用いられることがあります。日付区切り子は、日付値が書式指定されたとき、日、月、年の値を区切ります。書式指定された出力で実際に使われる文字は、Windows の設定によって異なります。
c	日付を ddddd で、時間を ttttt で、この順序で表示します。日付シリアル番号に分数部分がないと日付だけを、また、時間情報に整数部分がないと時間情報だけを、表示します。
d	日にちを先頭にゼロをつけないで表示します。(1~31)
dd	日にちを先頭にゼロをつけて表示します。(01~31)
ddd	曜日を略して表示します。(Sun~Sat)
dddd	曜日を略さないで表示します。(Sunday~Saturday)
dddddd	Windows のショートデータ表示設定のフォーマットにしたがって、日付、月、年をすべて表示します。Windows では、ショートデータフォーマットの設定は、デフォルトで、m/d/yy となっています。
dddddd	日付のシリアル数値を、日付、月、年として、Windows のロングデータ表示設定のフォーマットにしたがって表示します。Windows では、ロングデータフォーマットの設定は、デフォルトで mmmm dd, yyyy となっています。
w	曜日を数値で表示します。(1: 日曜日~7: 土曜日)
ww	1 年のうち第何週目かを、数値で表示します。(1~54)
m	月を先頭にゼロをつけない数値で表示します。(1~12) “h”または“hh”の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、“n”または“nn”を使用してください。
mm	月を先頭にゼロをつけた数値で表示します。(01~12) “h”または“hh”の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、“n”または“nn”を使用してください。
mmm	月名を略して表示します。(Jan~Dec)
mmmm	月名を略さないで表示します。(January~December)
q	四半期を数値で表示します。(1~4)
y	1 年のうち何番目の日かを、数値で表示します。(1~366)
yy	年号を 2 桁で表示します。(00~99)
yyyy	年号を 4 桁で表示します。(100~9999)
h	24 時間表記の時刻を、先頭にゼロをつけないで表示します。(0~23)
hh	24 時間表記の時刻を、先頭にゼロをつけて表示します。(00~23)
n	時刻の分を、先頭にゼロをつけないで表示します。(0~59)
nn	時刻の分を、先頭にゼロをつけて表示します。(00~59)
s	時刻の秒を、先頭にゼロをつけないで表示します。(0~59)
ss	時刻の秒を、先頭にゼロをつけて表示します。(00~59)
ttttt	Windows が設定する時刻区切り子フォーマットにしたがい、時刻 (時、分、秒)を表示します。“先頭のゼロ”オプションを選択すると、10:00am/pm 以前の時刻に関しては、先頭にゼロをつけた表記で表示します。Windows では、時刻フォーマットのデフォルトは、h:nn:ss になっています。

AM/PM	12 時間表記で時間を表示し、午前午後を AM/PM (大文字表記)で表示します。
am/pm	12 時間表記で時間を表示し、午前午後を am/pm (小文字表記)で表示します。
A/P	12 時間表記で時間を表示し、午前午後を A/P (大文字表記)で表示します。
a/p	12 時間表記で時間を表示し、午前午後を a/p (小文字表記)で表示します。
AMPM	12 時間表記で時間を表示します。午前時刻は、AM を文字列で、午後時刻は PM を文字列で、それぞれ Windows 内のフォーマット設定にしたがって表示します。AM/PM は、大文字でも小文字でも使えますが、Windows の設定と指定文字列が一致していなければいけません。Windows では、デフォルトは、AM/PM となっています。

注意

数値書式指定子と日付/時間書式指定子の混在

数値書式指定子と日付/時間書式指定子を混在して指定した場合は、エラーとなります。

参照

Left\$, Right\$, Str\$

FmtStr 使用例

```
Function SaveData

    String d$, f$, t$

    ' 月, 日, 時間, 分の形式でファイル名を作成
    d$ = Date$
    t$ = Time$
    d$ = d$ + " " + t$
    FmtStr d$, "mmddhhnn", f$
    f$ = f$ + ".dat"
    WOpen f$ as #30
    Print #30, "data"
    Close #30
End
```

FmtStr\$ 関数

数値式をフォーマットします。

書式

FmtStr\$ (書式指定する式, 書式指定文字列)

パラメーター

書式指定する式	書式指定する数値式または日付/時刻式を指定します。 日付/時刻式は、“yyyy/mm/dd hh:nn:ss”で指定してください。
書式指定文字列	書式指定文字列を指定します。

戻り値

書式指定した文字列を返します。

解説

書式設定文字列にしたがって、書式設定された文字列を返します。

数値書式指定子 文字 解説

None	フォーマットなしでそのまま番号を表示します。
(0)	桁位置を示す桁表示子です。数値またはゼロを表示します。数値式に含まれる 10 進数を、書式指定文字列の“0”のところに表示します。それ以外の場合は、その位置に“0”を表示します。数値の桁数が、書式指定文字列の (小数点の前後いずれかにおける)“0”の数より少ない場合は、先頭ないしは後方の“0”をいくつか表示します。もし、数値の小数点の右側の桁数が、書式指定文字列の小数点の右側にある“0”の数より多い場合は、“0”の数に合わせて数値の桁数を丸めます。また、数値の小数点左側の桁数が、書式指定文字列の小数点左側にある“0”の数より多い場合は、その余分な桁をそのまま表示します。
(#)	桁位置を示す桁表示子です。数値を示すか、何も表示しないかのいずれかを行います。数値式の 10 進数を書式指定文字列の“#”の位置に表示しますが、それ以外の場合は、その位置に何も表示しません。この文字 (#)は、0 桁表示子のような役目をしますが、数値の桁数が書式指定文字列の小数点の左右にある“#”の数と同じか、それより少ない場合は、先頭ないしは後方の“0”は表示されません。
(.)	小数点位置を表示し、小数点の左右にいくつの桁を表示するか設定します。ローカルによっては、カンマで小数点を示すものもあります。書式指定文字列中、この (.)の左に数字しかない場合は、1 より小さい数字が小数点をあたりに始まり、表示されます。先頭のゼロを表示するには、“0”を小数点の左に使います。書式指定されて出力されてくる値で小数点を示す文字は、使用する Windows が認識する数値フォーマットによります。
(,)	数値を 1000 で区切る千桁区切り子です。ローカルによっては、ピリオドを用います。小数点の左に 4 桁以上ある数値で、千ずつ区切るのに使います。標準的な使い方は、書式指定文字列が左右に桁表示子 (0 または #)をとまなう千桁区切り子 (,)を含んでいると指定されます。小数の指定があるなしにかかわらず、小数点のすぐ左に、千桁区切り子が 2 つ並んでいる場合、または単独で使われている場合、「その数値を 1000 で割って、必要ならば四捨五入せよ」という書式指定を意味します。例えば、書式指定文字列 “##0,,” は、1 億を“100”と表示せよ、という意味になります。百万より小さい数値は“0”と表示されます。小数点のすぐ左隣以外の位置に千桁区切り子が 2 つ並んで置かれると、単に千桁を区切るために使われます。書式指定された出力値で実際に千桁区切り子として使われる文字は、使用する Windows が認識する数値フォーマットによります。

日付/時間書式指定子

キャラクター 説明

(:)	時刻区切りです。ローカルによっては、他の文字が用いられることがあります。時刻区切りは、時間値が書式指定されたとき、時、分、秒の値を区切ります。書式指定された出力で実際に使われる文字は、Windows の設定によって異なります。
(/)	日付区切り子です。ローカルによっては、他の文字が用いられることがあります。日付区切り子は、日付値が書式指定されたとき、日、月、年の値を区切ります。書式指定された出力で実際に使われる文字は、Windows の設定によって異なります。
c	日付を dddddd で、時間を tttt で、この順序で表示します。日付シリアル番号に分数部分がないと日付だけを、また、時間情報に整数部分がないと時間情報だけを、表示します。
d	日にちを先頭にゼロをつけないで表示します。(1~31)
dd	日にちを先頭にゼロをつけて表示します。(01~31)
ddd	曜日を略して表示します。(Sun~Sat)
dddd	曜日を略さないで表示します。(Sunday~Saturday)
dddddd	Windows のショートデータ表示設定のフォーマットにしたがって、日付、月、年をすべて表示します。Windows では、ショートデータフォーマットの設定は、デフォルトで、m/d/yy となっています。
dddddd	日付のシリアル数値を、日付、月、年として、Windows のロングデータ表示設定のフォーマットにしたがって表示します。Windows では、ロングデータフォーマットの設定は、デフォルトで mmmm dd, yyyy となっています。
w	曜日を数値で表示します。(1: 日曜日~7: 土曜日)
ww	1 年のうち第何週目かを、数値で表示します。(1~54)
m	月を先頭にゼロをつけない数値で表示します。(1~12) “h”または“hh”の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、“n”または“nn”を使用してください。
mm	月を先頭にゼロをつけた数値で表示します。(01~12) “h”または“hh”の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、“n”または“nn”を使用してください。
mmm	月名を略して表示します。(Jan~Dec)
mmmm	月名を略さないで表示します。(January~December)
q	四半期を数値で表示します。(1~4)
y	1 年のうち何番目の日かを、数値で表示します。(1~366)
yy	年号を 2 桁で表示します。(00~99)
yyyy	年号を 4 桁で表示します。(100~9999)
h	24 時間表記の時刻を、先頭にゼロをつけないで表示します。(0~23)
hh	24 時間表記の時刻を、先頭にゼロをつけて表示します。(00~23)
n	時刻の分を、先頭にゼロをつけないで表示します。(0~59)
nn	時刻の分を、先頭にゼロをつけて表示します。(00~59)
s	時刻の秒を、先頭にゼロをつけないで表示します。(0~59)
ss	時刻の秒を、先頭にゼロをつけて表示します。(00~59)
t t t t t	Windows が設定する時刻区切り子フォーマットにしたがい、時刻 (時、分、秒)を表示します。“先頭のゼロ”オプションを選択すると、10:00am/pm 以前の時刻に関しては、先頭にゼロをつけた表記で表示します。Windows では、時刻フォーマットのデフォルトは、h:nn:ss になっています。

AM/PM	12 時間表記で時間を表示し、午前午後を AM/PM (大文字表記)で表示します。
am/pm	12 時間表記で時間を表示し、午前午後を am/pm (小文字表記)で表示します。
A/P	12 時間表記で時間を表示し、午前午後を A/P (大文字表記)で表示します。
a/p	12 時間表記で時間を表示し、午前午後を a/p (小文字表記)で表示します。
AMPM	12 時間表記で時間を表示します。午前時刻は、AM を文字列で、午後時刻は PM を文字列で、それぞれ Windows 内のフォーマット設定にしたがって表示します。AM/PM は、大文字でも小文字でも使えますが、Windows の設定と指定文字列が一致していなければいけません。Windows では、デフォルトは、AM/PM となっています。

注意

数値書式指定子と日付/時間書式指定子の混在

数値書式指定子と日付/時間書式指定子を混在して指定した場合は、エラーとなります。

参照

Left\$, Right\$, Str\$

FmtStr\$使用例

```
Function SendDateCode

    String d$, f$

    f$ = FmtStr$(10, "000.00")
    OpenCom #1
    Print #1, f$
    CloseCom #1
Fend
```

FolderExists 関数

指定したフォルダーの有無をチェックします。

書式

FolderExists(パス名)

パラメーター

パス名 チェックするフォルダーのパス名を文字列で指定します。ドライブ名を含みます。
パスの詳細は、ChDisk を参照してください。

注意

-
- ディスクが PC の場合に実行可能です。
-

戻り値

フォルダが存在する場合	True
フォルダが存在しない場合	False

参照

FileExists, Mkdir

FolderExists 関数使用例

```
If Not FolderExists ("c:\TEST") Then  
    Mkdir "c:\TEST"  
EndIf
```

For...Next

For...Next 間にある一連のステートメントを、指定の回数だけ、繰り返し実行します。

書式

```
For 変数名 = 初期値 To 終了値 [Step 増分値]
    ステートメント
Next [変数名]
```

パラメーター

変数名	繰り返しデータを代入する変数名を指定します。この変数は通常、整数値ですが、実数変数として定義することも可能です。
初期値	指定した変数にループの最初に代入する数値を指定します。
終了値	ループの終了を表わす値を指定します。この値がみたされると、For...Next ループは終了し、Next 命令の次のステートメントを続けて実行します。
増分値	For...Next ループ中の Next ステートメントが実行されるたびに増分する値を指定します。増分値は負の数にも設定できますが、この場合は、初期値 > 終了値でなければなりません。増分の指定がない場合は、自動的に“1”で増分します。省略可能です。
ステートメント	有効な SPEL+ のステートメントであれば、どれでも For...Next ループ中に挿入することができます。

解説

For...Next は、ループ中にあるステートメントを指定された回数だけ繰り返します。ループの最初は、For ステートメントで、終わりは Next ステートメントです。変数は、ループ内のステートメントが実行される回数を数えます。

初期値の数値は、カウンターの最初の値です。終了値変数と増分値が正しく設定されていれば、負の数値でも設定できます。

終了値は、カウンターの最終値です。この値に達するとループが終了し、プログラム制御は Next 命令の次の命令に移行します。

For ステートメントの次のステートメントは、Next 命令に達するまで、実行されます。カウンター変数 (変数名) は、増分値パラメーターによって指定された値だけ増分していきます。もし増分値が設定されていなければ、カウンターは、“1”ずつ増減します。

そのあと、カウンター変数 (変数名) は、最終値と比較されます。カウンターが最終値よりも少ないか、同じ場合は、For 命令の次のステートメントが再実行されます。カウンター変数 (変数名) が、最終値よりも大きい場合は、実行は、For...Next ループの外に枝分かれし、Next 命令の次の命令に継続します。

注意**負の増分値**

負の値を増分値に指定すると、カウンタ変数は、ループのたびに減少するので、初期値は終了値よりも大きくしてください。

Next の後の変数名は省略できます

Next の後の変数名は省略できますが、ネスティングがある場合などは、変数名を記入しておいたほうが構造がわかりやすくなります。

ループから抜けたときの変数の値は、終了値ではありません

```
Function forsamp1e
  Integer i
  For i = 0 To 3
  Next
  Print i    '4と表示されます
Fend
```

Exit For を使用せず、ネスト構造からループを抜けた場合

Exit For 命令以外のコマンド(Gosub 命令、Goto 命令、Call 命令など)でループから抜けるプログラムを繰り返し実行すると、エラー2020が発生します。ループの途中で抜ける場合は、Exit For 命令で抜けるようにしてください。

空の無限ループ、および無限ループと同様の処理は、できる限り Wait 命令と併用してください

空の For...Next、および同様の処理は、システムに影響を与える可能性があるため、できるだけ行わないでください。コントローラーでは、無限ループタスクの検出を行っており、システムに影響が出るとコントローラーが判断した場合、エラー2556 (過剰なループを検出した)が発生することがあります。ループを必要とする演算や、I/O 待ちを行う場合は、Wait 命令 (Wait 0.1)などをループ処理内で実行し、CPU の占有を回避してください。

参照

Do...Loop

For...Next 使用例

```
Function fornxt
  Integer counter
  For counter = 1 to 10
    Go Pctr
  Next counter

  For counter = 10 to 1 Step -1
    Go Pctr
  Next counter
Fend
```

Force_Calibrate

現在のフォースセンサーの全軸に対してゼロオフセットを設定します。

書式

Force_Calibrate

パラメーター

On | Off トルク制御をオン、またはオフします。

解説

このコマンドは、アプリケーションを開始するとき、センサーごとに呼び出してください。センサーに取りつけられている部品の質量分をキャンセルします。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_Sensor

Force_Calibrate 使用例

Force_Calibrate

Force_ClearTrigger

現在のフォースセンサーのトリガー条件をすべてクリアします。

書式

Force_ClearTrigger

解説

このコマンドは、現在のフォースセンサーのトリガーの全条件をクリアします。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_Sensor, Force_SetTrigger

Force_ClearTrigger 使用例

Force_ClearTrigger

Force_GetForces

すべてのフォースセンサー軸のフォース、トルクを配列で返します。

書式

Force_GetForces 配列()

パラメーター

配列() 上限が 6 の実数配列

戻り値

配列要素は次のように与えられます。

指数	軸	定数
1	X Force	FORCE_XFORCE
2	Y Force	FORCE_YFORCE
3	Z Force	FORCE_ZFORCE
4	X Torque	FORCE_XTORQUE
5	Y Torque	FORCE_YTORQUE
6	Z Torque	FORCE_ZTORQUE

解説

このコマンドは、すべてのフォース、トルク値を一度に読み込むために使用します。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_GetForce

Force_GetForces 使用例

```
Real fValues(6)
Force_GetForces fValues()
```

Force_GetForce 関数

指定する軸のフォースを返します。

書式

Force_GetForce (軸)

パラメーター

軸	軸を表す整数値		
	軸	定数	値
	X Force	FORCE_XFORCE	1
	Y Force	FORCE_YFORCE	2
	Z Force	FORCE_ZFORCE	3
	X Torque	FORCE_XTORQUE	4
	Y Torque	FORCE_YTORQUE	5
	Z Torque	FORCE_ZTORQUE	6

戻り値

実数値を返します。

解説

このコマンドは、一軸について現在のフォース設定を読み込みます。単位は、フォースセンサーの種類ごとに定義されています。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_GetForces

Force_GetForce 関数 使用例

```
Print Force_GetForce(1)
```

Force_Sensor

現在のタスクについて、使用するフォースセンサーを設定します。

書式

Force_Sensor センサー番号

パラメーター

センサー番号 センサー番号を表す整数式

解説

同じシステムで複数のフォースセンサーを使用している場合、他のフォースセンシングコマンドを実行する前に現在のフォースセンサーを設定してください。

システムにセンサーが 1 つしかない場合、初期センサー番号が 1 なので、このコマンドを使用する必要はありません。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_Sensor 関数

Force_Sensor 使用例

Force_Sensor 1

Force_Sensor 関数

現在のタスクについて、使用するフォースセンサーを返します。

書式

Force_Sensor

解説

このコマンドは、現在のタスクに使用するセンサー番号を返します。タスクが開始すると、センサー番号は自動的に 1 になります。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_Sensor

Force_Sensor 関数 使用例

```
var = Force_Sensor
```

Force_SetTrigger

Till コマンドのためのフォーストリガーを設定します。

書式

Force_SetTrigger 軸, しきい値, 比較様式

パラメーター

軸	使用したいフォースセンサー軸を含む整数式		
	軸	定数	値
	X Force	FORCE_XFORCE	1
	Y Force	FORCE_YFORCE	2
	Z Force	FORCE_ZFORCE	3
	X Torque	FORCE_XTORQUE	4
	Y Torque	FORCE_YTORQUE	5
	Z Torque	FORCE_ZTORQUE	6
しきい値	使用したいしきい値を含む実数式 (単位は使用しているセンサーによる)		
比較様式	判定条件	定数	値
	未満	FORCE_LESS	0
	以上	FORCE_GREATER	1

解説

フォースセンサーで動作を停止するには、センサーにトリガーを設定します。動作命令で Till Force を使用してください。

トリガーは複数の軸に設定できます。各軸にこのコマンドを呼び出してください。軸を無効にするには、しきい値を 0 に設定します。

注意

このコマンドは、フォースセンシングオプション(ATI フォースセンサー) がインストールされている場合のみ使用可能です。

エプソン製の力覚センサーを使用する場合は、以下マニュアルの各コマンドを使用してください。

EPSON RC+ 7.0 オプション Force Guide 7.0 SPEL+ ランゲージリファレンス

参照

Force_Calibrate

Force_SetTrigger 使用例

```
'Z 軸にかかる力が-1 未満のとき動作を停止するトリガーを設定する
Force_SetTrigger 3, -1, 0
SpeedS 3
AccelS 5000
Move Place Till Force
```


FreeFile 関数

現在使われていないファイル番号を返し、その番号を予約します。

書式

FreeFile

戻り値

30~63 の整数を返します。

参照

AOpen, BOpen, ROpen, UOpen, WOpen, Close

FreeFile 関数 使用例

```
Integer fileNum, i, j

fileNum = FreeFile
WOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

Function...Fend

1つのファンクションは、Function ステートメントで始まり Fend ステートメントで終わる、実行可能なプログラムの最小単位となります。

書式

```
Function ファンクション名 [(引数リスト)] [As 型(関数)]
    ステートメント
Fend
```

パラメーター

ファンクション名	Function で始まり Fend 命令で終わるステートメントのグループに対する名前を、アルファベット 64 文字以内で指定します。アンダースコアを用いることも可能です。
引数リスト	呼び出されて、ファンクションに渡される引数の変数のリストです。複数の変数の場合は、カンマで区切ります。省略可能です。 引数の書式は下記のとおりです。 [{ByRef ByVal}] 変数名[()] As 型(引数) ByRef 呼び出すファンクションの変数を参照する場合は、ByRef を指定します。この場合、ファンクション内での引数の変更を、呼び出し側の変数に反映させることができます。省略可能です。 ByVal デフォルト設定です。呼び出すファンクションで参照する変数の値の変更をしない場合は、ByVal を指定します。省略可能です。 変数名[()] 引数の変数名で、必須パラメーターです。変数名のつけ方の規則にしたがってください。 配列変数を引数として使用する場合は、ByRef を必ず指定した上で、配列を示す"()"中抜きの括弧を変数の後ろに付加してください。 As 型(引数) 必須パラメーターです。引数の型を宣言してください。
As 型(関数)	戻り値を取得したい場合に付加するパラメーターです。戻り値の型を宣言してください。

戻り値

ファンクション宣言の最後に As で指定されたデータ型です (As 型(関数))。

解説

Function ステートメントは、SPEL+ステートメントグループの始まりを示します。1つのファンクションの終わりは、Fend ステートメントで示します。Function と Fend ステートメントの間にあるステートメントは、すべてそのファンクションの一部とされます。

Function と Fend ステートメントは、その間にあるすべてのステートメントを含む、それ自体が1つのファンクションである、入れ物のようなものと考えられます。複数のファンクションを、1つのプログラムファイルに用いることも可能です。

戻り値を使用したい場合、ファンクション名と同名の変数に値を代入してから、関数を終了させてください。

参照

Call, Fend, Halt, Quit, Return, Xqt

Function...Fend 使用例**<例 1>**

下記は、1つのファイルに3つのファンクションを含む例です。task2とtask3は、mainから呼び出され、mainと同時に実行されます。

```

Function main
  Xqt 2, task2 'タスク2を同時に実行
  Xqt 3, task3 'タスク3を同時に実行
  .
  .
  .
Fend

Function task2
  Do
    On 1
    On 2
    Off 1
    Off 2
  Loop
Fend

Function task3
  Do
    On 10
    Wait 1
    Off 10
  Loop
Fend

```

<例 2>

下記は、周辺装置の圧力制御シーケンスを引数として与え、外部装置に送信したときの結果を戻り値として画面表示する場合の、ファンクション例です。

```

Function main
  Integer iResult
  Real Sequence1(200)
  .
  .
  iResult = PressureControl(ByRef Sequence1()) '引数は配列
  Print "Result:", iResult
  .
Fend

Function PressureControl(ByRef Array1() As Real) As Integer
  .
  (Array1 配列に基づいて、周辺装置を圧力制御)
  .
  PressureControl = 3 '戻り値
  .
  .
Fend

```

GetCurrentUser\$関数

現在の EPSON RC+ ユーザーを返します。

書式

GetCurrentUser\$

戻り値

現在の EPSON RC+ ユーザーのログイン ID を返します。

注意

このコマンドは、セキュリティーオプションが有効になっているときのみ使用可能です。

参照

Login

GetCurrentUser\$ 関数使用例

```
String currUser$  
  
currUser$ = GetCurrentUser$
```

GetRobotInsideBox 関数

進入検出エリアに進入しているロボットを返します。

書式

GetRobotInsideBox(エリア番号)

パラメーター

エリア番号 状態を返す進入検出エリアの番号 (1~15 の整数)を指定します。

戻り値

エリア番号で指定された進入検出エリアに進入しているロボットをビットで返します。

ビット 0 がロボット 1 で以降順番にビット 15 がロボット 16 です。

ロボットで進入検出エリアが設定されていない場合は、該当ビットは常に 0 になります。

例えばロボット 1 とロボット 3 がエリアに進入している場合はビット 0 とビット 2 が On するので 5 を返します。

参照

Box, InsideBox

GetRobotInsideBox 関数使用例

下記は GetRobotInsideBox 関数を使用したプログラムです。

進入検出エリアに 1 台もロボットが進入していない状態を待ちます。

```
Function WaitNoBox

    Wait GetRobotInsideBox(1) = 0
```

進入検出エリアにロボット 2 だけが進入している状態を待ちます。

```
Function WaitInBoxRobot2

    Wait GetRobotInsideBox(1) = &H2
```

下記のプログラムは、動作命令の並列処理内で使用する例です。動作実行中に特定の進入検出エリアに進入した時に I/O をオンします。コントローラーに接続しているロボットが 1 台のケースです。

```
Function Main
    Motor On
    Power High
    Speed 30; Accel 30, 30

    Go P1 !D0; Wait GetRobotInsideBox(1) = 1; On 1!

Fend
```

注意

D0 を必ず記述してください。

GetRobotInsidePlane 関数

進入検出平面に進入しているロボットを返します。

書式

GetRobotInsidePlane(平面番号)

パラメーター

平面番号 状態を返す進入検出平面の番号(1~15の整数)を指定します。

戻り値

平面番号で指定された進入検出平面に進入しているロボットをビットで返します。

ビット0がロボット1で以降順番にビット15がロボット16です。

ロボットで進入検出平面が設定されていない場合は、該当ビットは常に0になります。

例えば、ロボット1とロボット3が平面に進入している場合はビット0とビット2がOnするので5を返します。

参照

InsidePlane, Plane

GetRobotInsidePlane 関数使用例

下記は GetRobotInsidePlane 関数を使用したプログラムです。

進入検出平面に1台もロボットが進入していない状態を待ちます。

```
Function WaitNoPlane  
  
    Wait GetRobotInsidePlane(1) = 0
```

進入検出平面にロボット2だけが進入している状態を待ちます。

```
Function WaitInPlaneRobot2  
  
    Wait GetRobotInsidePlane(1) = &H2
```

下記のプログラムは、動作命令の並列処理内で使用する例です。動作実行中に特定の進入検出平面に進入した時にI/Oをオンします。コントローラーに接続しているロボットが1台のケースです。

```
Function Main  
    Motor On  
    Power High  
    Speed 30; Accel 30, 30  
  
    Go P1 !D0; Wait GetRobotInsidePlane(1) = 1; On 1!  
  
Fend
```

注意

D0を必ず記述してください。

Global

グローバル変数の宣言をします。グローバル変数はどこからでもアクセスできます。

書式

Global [Preserve] データ型 変数名 [(配列変数の最大要素番号)]
[, 変数名[(配列変数の最大要素番号)] ,...]

パラメーター

Preserve Preserveを指定すると、変数の値を保持します。プロジェクトを変更すると、この値はクリアします。省略可能です。

データ型 Boolean, Byte, Double, Int32, Integer, Long, Real, Short, String, UByte, UInt32, UShort のいずれかのデータ型を指定します。

変数名 変数名を 32 文字以内で指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高3次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号1, [最大要素番号2], [最大要素番号3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。

グローバル変数の場合、全要素数は最大100,000です。ただし、String型は最大10,000です。

バックアップ変数 (Global Preserve)の場合、全要素数は最大4,000です。ただし、String型は最大400です。

各最大要素番号は、全要素数が最大値を超えない範囲で指定します。

解説

グローバル変数は、同じプロジェクト内であれば複数のファイルで共有できる変数です。Preserve オプションで宣言されない限り、Run ウィンドウやオペレーターウィンドウからファンクションが起動されるたびにクリアされます。

Preserve オプションで宣言されると、コントローラーの電源を落としても値が保持されます。

保存したグローバル変数は、オプションの RC+ API オプションで用いることも可能です。

グローバル変数名は、プログラム中で見分けやすくするため、下記例のように、接頭部 "g_" で始めてください。

```
Global Long g_PartsCount
```

参照

Boolean, Byte, Double, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Global 使用例

下記は、異なる2つのプログラムファイルです。最初のファイルでは、いくつかのグローバル変数を定義し初期化しています。次のファイルで、それらのグローバル変数を使っています。

FILE1 (MAIN.PRG)

```
Global Integer g_Status
```

```
Global Real g_MaxValue
```

```
Function Main
```

```
    g_Status = 10
```

```
    g_MaxValue = 1.1
```

```
    .
```

```
    .
```

```
Fend
```

```
Function Test
```

```
    Print "status1 =" , g_Status
```

```
    Print "MaxValue =" , g_MaxValue
```

```
    .
```

```
    .
```

```
Fend
```


Go

アームを、現在位置から指定位置まで PTP 動作で動かします。

書式

Go 目標座標 [CP] [LJM [姿勢フラグ選択]] [Till | Find] [! 並列処理 !] [SYNC]

パラメーター

目標座標	ポイントデータで目標位置を指定します。
CP	パスモーションを指定します。省略可能です。
LJM	目標座標を LJM 関数で変換します。省略可能です。
姿勢フラグ選択	LJM 関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
! 並列処理 !	I/O その他のコマンドを動作中に実行させたいときに、並列処理ステートメントを付け加えることができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

Go は、ロボットアームのすべての関節を、同時に PTP 動作で動かします。下記は、Go 命令に対する目標座標を指定する例です。

- 特定のポイント番号で指定する。例: Go P1
- 移動先を具体的な座標値で指定する。例: Go XY(50, 400, 0, 0)
- ポイント番号 + オフセット値で指定する。例: Go P1 +X(50)
- ポイント番号 + それと異なる座標値を明示して指定する。例: Go P1 :X(50)

動作の経路は、各関節がそれぞれ現在のポイントから目標座標までを補間するため、軌跡は予測できません。周辺装置との干渉に十分注意してください。

Go 命令の動作速度は、Speed 命令で設定します。加減速度は、Accel 命令で設定します。

CP パラメーターをつけた場合には、動作の減速開始時に次の動作命令の加速を重ね合わせることが可能です。この場合には、目標座標には位置決めされません。

LJM パラメーターをつけた場合には、現在位置を参照位置として、目標位置を LJM 関数で変換した位置に動作します。

Go LJM(P1, Here,1)を、Go P1 LJM 1 と簡略して記述できます。

このとき元のポイントデータ P1 は変更されません。

LJM パラメーターは垂直 6 軸型ロボット (N シリーズを含む)、および RS シリーズロボットで有効です。

姿勢フラグ選択をデフォルト値で使用する場合は、省略が可能です。

Go P1 LJM

注意

Go と Move の違い

Move と Go は、ともにロボットアームを動作させる命令です。両者の間の大きな違いは、Go が PTP 動作させるのに対し、Move は直線軌道でアームを移動させることです。目標ポイントに到着したときのアームの姿勢が重要であるときは Go 命令を用いますが、動作中のアームの軌道を制御することがより重要である場合は、Move 命令を使います。

Go と Jump の違い

Jump と Go は、ともにロボットアームを PTP 動作で移動させる命令です。しかし、Jump には、Go がない機能が 1 つあります。Jump では、ロボットのハンドをまず LimZ 値まで持ち上げてから、アームを水平移動させ、目標座標の上空にきたところで下降移動させる、という動きをさせることです。この動きのメリットは、障害物をより確実に避けることができる点と、さらに重要なのは、吸着、配置動作による作業のサイクルタイムの向上にあります。

Go に適切な速度と加減速度指示を与える

Go 命令の動作速度と加減速度の設定は、Speed と Accel 命令で行います。非常に重要なことですが、Speed と Accel 命令は Go 命令と同じく、PTP 動作に対する設定を行います。直線および円弧補間動作の速度、加減速度の設定は、SpeedS と AccelS 命令で行います。

Till 修飾子を用いる Go の用法

Till 修飾子を用いることで、Go 命令で指定された目標座標に到着する前に、そこに至る通過点で、ロボットを減速停止する条件を設定することができます。もし Till 条件が成立しない場合は、ロボットはそのまま目標座標まで移動します。Till を用いた Go の用法には、下記の 2 通りがあります。

(1) Go と Till 修飾子

現在の Till 条件が成立したかどうかチェックします。成立すれば、Go が指定した動作の完了を待たずにロボットを途中通過点で減速停止させることで、コマンドの実行は完了します。

(2) Go と Till 修飾子、Sw (入力ビット番号) 修飾子、入力条件

この用法では、あらかじめ設定された Till の既存の設定条件を用いるのではなく、Go 命令と同じ行で Till 条件を新たに設定することができます。設定する条件は、入力の 1 つをチェックするというものですが、そのためには Sw 命令が必要になります。入力の状態がオンかオフかをチェックし、設定した条件に応じてアームを停止させることができます。この機能は、入力条件が成立すると動作が停止させられる「割込み」とほとんど同じ働きをします。もし入力条件がロボットの動作中に 1 度も成立しなければ、アームは指定された目標座標位置に到達します。

Find 修飾子を用いる Go の用法

Find 修飾子を用いることで、Go 命令による動作中にロボットに 1 つの位置を記録するよう条件設定することができます。Find を用いた Go の用法には、下記の 2 通りがあります。

(1) Go と Find 修飾子

現在の Find 条件が成立したかどうかチェックします。成立すれば、現在の位置を特別なポイント、FindPos に保存します。

(2) Go と Find 修飾子、Sw(入力ビット番号) 修飾子、入力条件

この用法では、あらかじめ設定された Find の既存の設定条件を用いるのではなく、Go 命令と同じ行で新たに Find 条件を設定することができます。設定条件は、1 つの入力をチェックすることです。そのためには Sw 命令が必要になります。入力の状態がオンかオフかをチェックし、現在の位置を特別なポイント、FindPos に保存します。

Go 命令では停止前に必ず減速します

Go 命令では、アームが動作の目標座標に停止する前に、必ず減速します。

起こりやすいエラー

可動範囲外にロボットを動作させようとしたとき

目標座標を直接座標で設定する場合、その座標位置がロボットの可動範囲内にあるか、必ず確認してください。ロボットの可動範囲内外が指定されていると、エラーが発生します。

参照

! 並列処理 !, Accel, Find, Jump, Move, Pass, P#= ポイント 指定, Pulse, Speed, Sw, Till

Go 使用例

下記は、P0 から P10 に PTP 動作する簡単な例です。プログラム後半で、アームは、入力ビット 2 がオンになるまで、P2 に向かって PTP 動作します。Go の途中で、入力ビット 2 がオンになると、アームは P2 到着を待たずに減速停止し、次の命令を実行します。

```
Function sample
  Integer i

  Home
  Go P0
  Go P1
  For i = 1 to 10
    Go P(i)
  Next i
  Go P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  EndIf
Fend
```

コマンドウィンドウからの操作例です。

>Go Here +X(50)	'現在の位置から X 方向に 50mm 移動
>Go P1	'P1 に移動
>Go P1 :U(30)	'P1 ただし U=30 の位置へ移動
>Go P1 /L	'左腕姿勢で P1 へ移動
>Go XY(50, 450, 0, 30)	'X=50, Y=450, Z=0, U=30 の位置へ移動

<その他の例>

Till Sw(1) = Off And Sw(2) = On	'入力 1 と 2 を Till 条件として指定
Go P1 Till	'あらかじめ定義した Till 条件を満たした場合は停止
Go P2 Till Sw(2) = On	'入力ビット 2 がオンになった場合は停止
Go P3 Till	'あらかじめ定義した Till 条件を満たした場合は停止

GoSub...Return

GoSub は、プログラム制御をサブルーチンに移行します。サブルーチンの実行が終わると、制御は GoSub 命令の次の行に戻ります。

書式

GoSub {ラベル}

{ラベル:}
ステートメント
Return

パラメーター

ラベル 移行先をラベルで指定します。そのラベルがある行にプログラム実行は移行します。ラベル名は、32 文字以内で指定してください。ただし、最初の文字には数字は使えません。必ずアルファベット文字を用いてください。

解説

GoSub 命令は、指定のラベルにプログラム制御を移します。プログラムは移行先のステートメントを実行し、Return 命令まで、そのまま移行先の行を実行していきます。GoSub 命令は、サブルーチンの実行を行ってから Return によって、GoSub 命令の次の行に戻ります。サブルーチンは必ず Return で終わらせてください。

起こりやすいエラー

存在しないステートメントへの移行

GoSub 命令が存在しないラベルを移行先に指定すると、エラー 3108 が発生します。

GoSub がないのに Return が使われているとき

Return 命令は、GoSub 命令を実行したもとのプログラムへの復帰を行います。もし、GoSub がないのに Return が使われているとエラー 2383 が発生します。GoSub なしで Return が使われても、どこに復帰してよいかわからないので意味を持ちません。

参照

GoTo, OnErr, Return

GoSub 使用例

下記は、GoSub 命令を使って指定ラベルに移り、I/O 命令をいくつか実行して復帰するという簡単な例です。

```
Function main
    Integer var1, var2

    GoSub checkio  'ラベルを使用して GoSub を実行
    On 1
    On 2
    Exit Function

checkio:          'サブルーチンの開始位置
    var1 = In(0)
    var2 = In(1)
    If var1 = 1 And var2 = 1 Then
        On 1
    Else
        Off 1
    EndIf
    Return          'サブルーチンの終了位置
Fend
```

GoTo

GoTo 命令は、プログラム制御を指定のラベルに移します。

書式

GoTo {ラベル}

パラメーター

ラベル ラベルのある行にプログラム実行が移行します。ラベルは、32 文字までで指定しますが、最初の文字に数字は使えません。必ずアルファベットを使用してください。

解説

GoTo 命令は、指定のラベルにプログラム制御を移行します。プログラムは、移行先のステートメント行を実行し、その行以降を実行していきます。

注意

GoTo を多用しすぎる場合

1 つのプログラム中で、GoTo 命令を使い過ぎると、プログラムがわかりにくくなるので注意してください。通常は、できるだけ GoTo 命令は使用しないでください。実際には、どうしても使用しなければならない GoTo がありますが、ソースコードをあちこちに移行するような GoTo ステートメントの使い方は、エラーやその他の問題の原因になります。

参照

GoSub, OnErr

GoTo 使用例

下記は、GoTo 命令を使って、制御を行ラベルに移す簡単なプログラムの例です。

```
Function main

    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print "Input 1 was On, continuing cycle"
    .
    .
    Exit Function

mainAbort:
    Print "Input 1 was OFF, cycle aborted!"
Fend
```

Halt

指定された実行中のタスクを一時停止します。

書式

Halt タスク識別子

パラメーター

タスク識別子	タスク名またはタスク番号を整数値、または式で指定します。 タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。 タスク番号の指定 (整数)
	通常タスク : 1 から 32 バックグラウンドタスク : 65 から 80 トラップタスク : 257 から 267

解説

Halt は、タスク名またはタスク番号で指定された実行中のタスクを一時停止します。停止したところから再開するときは、Resume を使います。タスクの実行を完全に終了するときは、Quit を使います。タスクの状態を表示するには、EPSON RC+ のツールバー上のタスクマネージャのアイコンをクリックし、タスクマネージャを起動します。

Halt は、指定されたタスクが NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、トラップタスク、バックグラウンドタスクの場合も、タスクを一時停止します。ただし、これらのタスクを一時停止させるには十分な検討が必要になります。一般的なケースでは特殊なタスクに対して Halt を実行しないことを推奨します。

参照

Quit, Resume, Xqt

Halt 使用例

下記は、“flicker”という名前のファンクションを Xqt で起動した後、Halt で一時停止し、Resume で再開する例です。

```
Function main
  Xqt flicker          ' flicker タスクを実行

  Do
    Wait 3             ' flicke タスクを 3 秒間実行
    Halt flicker

    Wait 3             ' flicker タスクを 3 秒間一時停止
    Resume flicker

  Loop
Fend

Function flicker
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

Hand

ポイントのハンド(腕系)姿勢を設定します。

書式

(1) Hand ポイント指定 [, Lefty | Righty]

(2) Hand

パラメーター

ポイント指定 P 番号, P(式), ポイントラベルのいずれかを指定します。

Lefty | Righty ハンド(腕系)姿勢を指定します。

結果

2つのパラメーターが両方省略されると、ロボットの現在位置のハンド(腕系)姿勢を表示します。

Lefty | Righty パラメーターが省略されると、指定したポイントのハンド(腕系)姿勢を表示します。

注意

Hand コマンドは、ハンド(エンドエフェクター)を制御するコマンドではありません。

コマンド名	機能
Hand (本コマンド)	マニピュレーターの腕が右腕系か左腕系かを指定します。
Hand_On, Hand_Off, Hand_On 関数, Hand_Off 関数, Hand_TW 関数, Hand_Def 関数, Hand_Type 関数, Hand_Label\$関数, Hand_Number 関数	マニピュレーター先端に取りつける、ハンド(エンドエフェクター)を制御します。

混同しないように注意してください。

ハンド制御コマンドについては、Hand 機能マニュアルを参照してください。

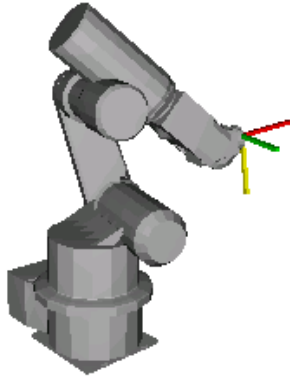
参照

Elbow, Hand 関数, J4Flag, J6Flag, Wrist, J1Flag, J2Flag

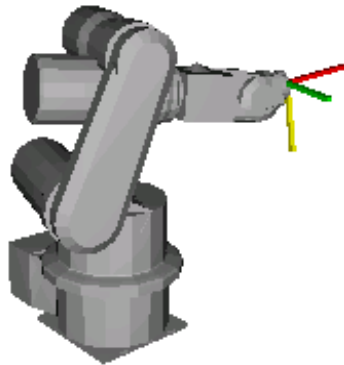
Hand 使用例

```
Hand P0, Lefty  
Hand pick, Righty  
Hand P(myPoint), myHand
```

```
P1 = -364.474, 120.952, 469.384, 72.414, 1.125, -79.991
```



```
Hand P1, Righty  
Go P1
```



```
Hand P1, Lefty  
Go P1
```

Hand 関数

ポイントのハンド(腕系)姿勢を返します。

書式

Hand [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。
省略可能で、省略するとロボットの現在位置のハンド(腕系)姿勢を返します。

戻り値

- 1 Righty (/R)
- 2 Lefty (/L)

注意

Hand 関数は、ハンド(エンドエフェクター)を制御する関数ではありません。

コマンド名	機能
Hand 関数 (本関数)	マニピュレーターの腕が右腕系か左腕系かを指定します。
Hand_On, Hand_Off, Hand_On 関数, Hand_Off 関数, Hand_TW 関数, Hand_Def 関数, Hand_Type 関数, Hand_Label\$関数, Hand_Number 関数	マニピュレーター先端に取りつける、ハンド(エンドエフェクター)を制御します。

混同しないように注意してください。
ハンド制御コマンドについては、Hand 機能マニュアルを参照してください。

参照

Elbow, Wrist, J4Flag, J6Flag, J1Flag, J2Flag

Hand 関数使用例

```
Print Hand(pick)
Print Hand(P1)
Print Hand
Print Hand(P1 + P2)
```

HealthCalcPeriod

部品消耗管理情報における「残月数」を演算する期間の設定と表示をします。

書式

- (1) HealthCalcPeriod 期間設定値
- (2) HealthCalcPeriod

パラメーター

期間設定値 演算する期間を整数値(1~7)で指定します。(単位: 日) デフォルト値は“7”です。

結果

パラメーターを省略すると、現在の HealthCalcPeriod 設定値を表示します。

解説

部品消耗管理情報の残月数は過去の動作状況により定期的に演算されます。HealthCalcPeriod コマンドは、この演算に用いる動作期間の設定と表示をします。

期間を長くすると、期間内の変動によるばらつきの影響を抑えた「残月数」が演算されます。ただし、動作や速度を変更したときに、「残月数」が正しく表示されるまでの時間が長くなります。

HealthCalcPeriod の設定値は、実行したコントローラーで制御されるすべてのロボット、関節、部品に対して適用されます。

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。T/VT シリーズでは常に 1(日)が使用されます。

設定期間

HealthCalcPeriod コマンドで設定する期間は、コントローラーを起動している期間です。実時間と異なることに注意してください。

「残り月数」および「消耗率」のクリア時の演算

EPSON RC+の「部品消耗管理情報」、HealthCtrlReset または HealthRBReset で「残り月数」および「消耗率」をクリアした後に最初に設定期間に達するまでの間は、HealthCalcPeriod の設定値にかかわらず、1 日ごとに残月数の演算を行います。この期間は残月数のばらつきが大きくなります。

参照

HealthCalcPeriod 関数, HealthCtrlInfo, HealthRBInfo, HealthCtrlReset, HealthRBReset

HealthCalcPeriod 使用例

```
> HealthCalcPeriod 3
> HealthCalcPeriod
3
```

HealthCalcPeriod 関数

現在の設定されている部品消耗管理情報における「残月数」を演算する期間を返します。

書式

HealthCalcPeriod

戻り値

演算する期間を整数値で返します。(単位: 日)

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

参照

HealthCalcPeriod

HealthCalcPeriod 関数使用例

演算期間を表示する例です

```
Print "period is", HealthCalcPeriod
```

HealthCtrlAlarmOn 関数

コントローラーに関する指定した種別の部品の部品消耗アラーム状態を返す関数です。

書式

HealthCtrlAlarmOn(部品種別)

パラメーター

部品種別 アラーム状態を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

指定した部品が部品消耗アラームを発生している場合は **True**、発生していない場合は **False** を返します。

部品消耗アラームは、HealthRateCtrlInfo で取得される部品消耗率が 100%を超えた場合に発生します。

参照

HealthCtrlInfo, HealthRateCtrlInfo

HealthCtrlAlarmOn 関数使用例

コントローラーのバッテリー部品消耗アラームが発生しているかを判断する例です。

```
Function PrintAlarm
    If HealthCtrlAlarmOn(HEALTH_CONTROLLER_TYPE_BATTERY) = True Then
        Print "Controller Battery NG"
    Else
        Print "Controller Battery OK"
    EndIf
End
```

HealthCtrlInfo

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を表示します。

書式

HealthCtrlInfo 部品種別

パラメーター

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

解説

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を表示します。

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて演算されます。

注意

コントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- ・稼働時間が HealthCalcPeriod で設定された期間に満たない状況で本コマンドを実行する場合。
- ・長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- ・部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- ・コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーを HealthCalcPeriod で設定された期間の 2 倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

HealthCtrlAlarmOn, HealthRateCtrlInfo

HealthCtrlInfo 使用例

コントローラーのバッテリー交換推奨時期までの残り月数を表示する例です。

```
> HealthCtrlInfo HEALTH_CONTROLLER_TYPE_BATTERY
BATTERY                240.000
>
```

HealthCtrlInfo 関数

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を返す関数です。

書式

HealthCtrlInfo(部品種別)

パラメーター

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

交換推奨時期までの残り月数を実数で返します (単位: 月)。

解説

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて演算されます。

注意

コントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- ・稼働時間が HealthCalcPeriod で設定された期間に満たない状況で本コマンドを実行する場合。
- ・長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- ・部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- ・コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーを HealthCalcPeriod で設定された期間の 2 倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

HealthCtrlAlarmOn, HealthRateCtrlInfo

HealthCtrlInfo 関数使用例

交換推奨時期までの残り月数が 1 か月以下になったらアラームを出力する例です。

```
Function AlarmCheck
    Real month

    month = HealthCtrlInfo(HEALTH_CONTROLLER_TYPE_BATTERY)
    If month < 1 Then
        Print "Alarm ON"
    EndIf
EndFunction
```

HealthCtrlRateOffset

指定した部品種別の消耗率に対してオフセットを設定します。

書式

HealthCtrlRateOffset 部品種別, オフセット値

パラメーター

部品種別	コントローラーに関する部品を整数値 (1)、または以下に示す定数で指定します。		
	定数	値	モード
	HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。
オフセット値	消耗率に対してオフセットする整数値を指定します。(単位: %)		

解説

指定した部品種別に対して、消耗率のオフセットを設定します。

参照

HealthRBAlarmOn, HealthRateRBInfo, HealthRBInfo

HealthCtrlRateOffset 使用例

コントローラーのバッテリーの消耗率に対して 10%加算する例です。

```
> HealthCtrlRateOffset HEALTH_CONTROLLER_TYPE_BATTERY, 10
>
```


HealthCtrlReset

指定した部品種別の交換推奨時期までの残り月数、および消耗率をクリアします。

書式

HealthCtrlReset 部品種別

パラメーター

部品種別 コントローラーに関する部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

解説

指定した部品種別に対して、交換推奨時期までの残り月数、および消耗率をクリアします。
警告も同時に解除します。

参照

HealthCtrlAlarmOn, HealthRateCtrlInfo, HealthCtrlInfo

HealthCtrlReset 使用例

```
> HealthCtrlReset HEALTH_CONTROLLER_TYPE_BATTERY
>
```

HealthCtrlWarningEnable

コントローラーに関する指定した種別の部品の部品消耗アラームの通知を有効または無効にします。

書式

HealthCtrlWarningEnable 部品種別[, On/Off]

パラメーター

部品種別 コントローラーの部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

On/Off On: 部品消耗アラームの通知を有効にします。

Off: 部品消耗アラームの通知を無効にします。

結果

On/Off パラメーターを省略すると、現在の On/Off 設定値が表示されます。

解説

指定した種別の部品の部品消耗アラームが発生した場合、部品消耗アラーム通知を行うかどうかを設定します。

注意

指定した部品の部品消耗アラームの通知を無効にした場合、交換推奨時期を経過しても部品消耗アラームが通知されません。本コマンドによる設定には十分注意して実行してください。

参照

HealthCtrlAlarmOn

HealthCtrlWarningEnable 使用例

コントローラーの、バッテリーの部品消耗アラーム通知を無効にする例です。

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY, Off
```

コントローラーの、バッテリーの部品消耗アラーム通知設定を表示する例です。

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY  
BATTERY Off  
>
```

HealthCtrlWarningEnable 関数

コントローラーに関する指定した種別の部品の部品消耗アラームの通知の設定状態を返す関数です。

書式

HealthCtrlWarningEnable(部品種別)

パラメーター

部品種別 コントローラーの部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

部品消耗アラームの設定値を整数値で返します。

1: On

0: Off

参照

HealthCtrlAlarmOn

HealthCtrlWarningEnable 関数使用例

コントローラーのバッテリーの部品消耗アラームの通知設定状態を表示する例です。

```
Print HealthCtrlWarningEnable(HEALTH_CONTROLLER_TYPE_BATTERY )
```

HealthRateCtrlInfo 関数

コントローラーに関する指定した種別の部品の消耗率を返す関数です。

書式

HealthRateCtrlInfo(部品種別)

パラメーター

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

交換推奨時期を 100%としたときの現在の部品消耗率を実数で返します(単位: %)。

解説

実稼働状況データに基づいて部品消耗率を演算します。

注意

交換推奨時期は、統計に基づいた部品交換を推奨する時期です。
部品消耗率が 100 %に達する前に、交換が必要になる場合もあります。
また、100 %に達した場合でも、即座に使用できなくなるわけではありません。
ただし、100%以降は故障の可能性が増加するため、早めの交換を推奨します。

参照

HealthCtrlAlarmOn, HealthCtrlInfo,

HealthRateCtrlInfo 関数使用例

コントローラーバッテリーの部品消耗率が 90 %以上になったらアラームを出力する例です。

```
Function AlarmCheck
    Real HealthRate

    HealthRate = HealthRateCtrlInfo (HEALTH_CONTROLLER_TYPE_BATTERY)
    If HealthRate > 90 Then
        Print "Alarm ON"
    EndIf
End
```

HealthRateRInfo 関数

ロボットに関する指定した種別の部品の消耗率を返す関数です。

書式

HealthRateRInfo(ロボット番号, 部品種別, 関節番号)

パラメーター

- ロボット番号** 部品消耗率を返すロボット番号を整数値 (1-16) で指定します。
- 部品種別** 部品消耗率を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

- 関節番号** 部品消耗率を返す関節を整数値 (1-9) で指定します。
本コマンドは、付加軸に対しては使用できません。

戻り値

- 交換推奨時期を 100 %としたときの、現在の部品消耗率を実数で返します (単位: %)。
- ロボットの関節が指定した部品を使用していない場合には、-1 を返します。

解説

実稼働状況データに基づいて部品消耗率を演算します。

注意

交換推奨時期は、統計に基づいた部品交換を推奨する時期です。
部品消耗率が 100 %に達する前に、交換が必要になる場合もあります。
また、100 %に達した場合でも、即座に使用できなくなるわけではありません。
ただし、100%以降は故障の可能性が増加するため、早めの交換を推奨します。

参照

HealthRBAAlarmOn, HealthRInfo

HealthRateRInfo 関数使用例

ロボット 1 の第 3 関節の減速機の部品消耗率が 90%以上になったらアラームを出力する例です。

```
Function AlarmCheck
  Real HealthRate

  HealthRate = HealthRateRInfo(1, HEALTH_ROBOT_TYPE_GEAR, 3)
  If HealthRate > 90 Then
    Print "Alarm ON"
  EndIf
End
```

HealthRBAAlarmOn 関数

ロボットに関する指定した種別の部品の部品消耗アラーム状態を返す関数です。

書式

HealthRBAAlarmOn(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号 アラーム状態を返すロボット番号を整数値 (1-16) で指定します。

部品種別 アラーム状態を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 アラーム状態を返す関節を整数値 (1-9) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。本コマンドは、付加軸に対しては使用できません。

戻り値

指定した部品が部品消耗アラームが発生している場合は **True**、発生していない場合は **False** を返します。

部品消耗アラームは、HealthRateRBInfo で取得される部品消耗率が 100%を超えた場合に発生します。

ロボットの関節が指定した部品を使用していない場合には、-1 を返します。

参照

HealthRBInfo, HealthRateRBInfo

HealthRBAAlarmOn 関数使用例

ロボット 1 の第 3 関節のグリス部品消耗アラームが発生しているかを判断する例です。

```
Function PrintAlarm4
  If HealthRBAAlarmOn(1, HEALTH_ROBOT_TYPE_GREASE, 3) = True Then
    Print "Robot1 Joint3 Grease NG"
  Else
    Print "Robot1 Joint3 Grease OK"
  EndIf
Fend
```

HealthRBAAnalysis

あるサイクルのロボット動作に関して、指定した部品種別の使用可能月数をシミュレートし、表示します。

書式

HealthRBAAnalysis ロボット番号, 部品種別[, 関節番号]

パラメーター

ロボット番号 ロボット番号を整数値 (1-16) で指定します。

部品種別 ロボットに関する部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_ALL	0	全部品種別を指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

あるサイクルのロボット動作で、指定した部品種別の使用可能月数をシミュレートします。部品が新品の状態で 24 時間連続稼働した場合に何カ月使用できるか算出し、表示します。過去の使用状況は加味されません。

指定の部品種別が、指定した関節で未実装の場合には、-1 を返します。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

HealthRBStart, HealthRBStop

HealthRBAAnalysis 使用例

スカルロボットの、全関節の全部品の使用可能月数を表示する例です。

```
> HealthRBAAnalysis 1, HEALTH_ROBOT_TYPE_ALL
BELT          -1.000,   -1.000,   38.689,   95.226
GREASE        -1.000,   -1.000,   21.130,   -1.000
MOTOR         240.000,  240.000,  240.000,  240.000
GEAR          240.000,  224.357,   -1.000,   -1.000
BALL_SCREW_SPLINE -1.000,   -1.000,  240.000,   -1.000
>
```

スカラロボットの、全関節の減速機の使用可能月数を表示する例です。

```
> HealthRBAalysis 1, HEALTH_ROBOT_TYPE_GEAR  
GEAR                240.000, 224.357, -1.000, -1.000  
>
```

6 軸ロボットの、第 2 関節のモーターの使用可能月数を表示する例です。

```
> HealthRBAalysis 1, HEALTH_ROBOT_TYPE_MOTOR, 2  
MOTOR              224.357  
>
```


HealthRBAalysis 関数

あるサイクルのロボット動作に関して、指定した部品種別の使用可能月数を返す関数です。

書式

HealthRBAalysis(ロボット番号, 部品種別, 関節番号)

パラメーター

- ロボット番号 ロボット番号を整数値 (1-16)で指定します。
- 部品種別 ロボットに関する部品を整数値 (2-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 関節を整数値 (1-6)で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

- 使用可能月数を実数で返します。
- 指定の部品種別が、指定した関節で未実装の場合には、-1 を返します。

解説

あるサイクルのロボット動作で、指定した部品種別の使用可能月数をシミュレートします。部品が新品の状態で 24 時間連続稼働した場合に何カ月使用できるか算出します。過去の使用状況は加味されません。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む)では、機能しません。

参照

HealthRBStart, HealthRBStop

HealthRBAalysis 関数使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

HealthRBDistance

指定した関節のモーター駆動量 (回転量)を表示します。

書式

HealthRBDistance [ロボット番号] [,関節番号]

パラメーター

ロボット番号	ロボット番号を整数値 (1-16)で指定します。 省略可能です。省略された場合は、カレントロボット番号を使用します。
関節番号	関節を整数値 (1-6)で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節のモーター駆動量 (回転量)を算出し、表示します。過去の使用量は加味されません。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む)では、機能しません。

参照

HealthRBStart, HealthRBStop

HealthRBDistance 使用例

スカラロボットの第 1 関節の駆動量を表示する例です。

```
> HealthRBDistance 1, 1
1.000
>
```

HealthRBDistance 関数

指定した関節のモーター駆動量 (回転量) を返す関数です。

書式

HealthRBDistance([ロボット番号,] 関節番号)

パラメーター

ロボット番号	ロボット番号を整数値 (1-16) で指定します。 省略可能です。省略された場合は、カレントロボット番号を使用します。
関節番号	関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

駆動量を実数で返します。

解説

HealthRBStart から HealthRBStop までのロボット動作に対して、指定した関節のモーター駆動量 (回転数) を返します。過去の使用量は加味されません。

注意

- Auto モードでは、機能しません。
 - ドライラン (仮想コントローラー含む) では、機能しません。
-

参照

HealthRBStart, HealthRBStop

HealthRBDistance 関数使用例

```
Function RobotPartAnalysis
  Real healthDistance

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthDistance = HealthRBDistance(1,1)
  Print "Distance =", Str$(healthDistance)
Fend
```

HealthRBIInfo

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を表示します。

書式

HealthRBIInfo ロボット番号, 部品種別[, 関節番号]

パラメーター

ロボット番号 交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16)で指定します。

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (0-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_ALL	0	全部品種別を指定します。
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 交換推奨時期までの残り月数を返す関節を整数値 (1-9)で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を表示します。

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて演算されます。

ロボットの関節が指定した部品を使用していない場合には、-1を表示します。

注意

コントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間が HealthCalcPeriod で設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーを HealthCalcPeriod で設定された期間の 2 倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

HealthRBAAlarmOn, HealthRateRBIInfo

HealthRInfo 使用例

ロボット 1 の、全関節の全部品の交換推奨時期までの残り月数を表示する例です。

```
> HealthRInfo 1, HEALTH_ROBOT_TYPE_ALL
BATTERY          240.000
BELT              -1.000,   -1.000,   38.689,   95.226
GREASE            -1.000,   -1.000,   21.130,   -1.000
MOTOR            240.000,  240.000,  240.000,  240.000
GEAR              240.000,  224.357,   -1.000,   -1.000
BALL_SCREW_SPLINE -1.000,   -1.000,  240.000,   -1.000
>
```

ロボット 1 の、全関節の減速機の交換推奨時期までの残り月数を表示する例です。

```
> HealthRInfo 1, HEALTH_ROBOT_TYPE_GEAR
GEAR              240.000,  224.357,   -1.000,   -1.000
>
```

ロボット 1 の、第 2 関節のモーターの交換推奨時期までの残り月数を表示する例です。

```
> HealthRInfo 1, HEALTH_ROBOT_TYPE_MOTOR, 2
MOTOR             224.357
>
```

HealthRBIInfo 関数

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を返す関数です。

書式

HealthRBIInfo(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号 交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16) で指定します。

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 交換推奨時期までの残り月数を返す関節を整数値 (1-9) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。本コマンドは、付加軸に対しては使用できません。

戻り値

交換推奨時期までの残り月数を実数で返します (単位: 月)。
ロボットの関節が指定した部品を使用していない場合には、-1 を返します。

解説

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて演算されます。

注意

コントローラーを HealthCalcPeriod で設定された期間稼働することによって取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間が HealthCalcPeriod で設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーを HealthCalcPeriod で設定された期間の 2 倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

HealthRBAAlarmOn, HealthRateRBIInfo

HealthRInfo 関数使用例

ロボット 1 の第 3 関節ボールねじスプラインの交換推奨時期までの残り月数が 1 か月以下になった場合にアラームを出力する例です。

```
Function AlarmCheck
  Real month

  month = HealthRInfo(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  If month < 1 Then
    Print "Alarm ON"
  EndIf
Fend
```


HealthRBRateOffset

指定した部品種別の消耗率に対してオフセットを設定します。

書式

HealthRBRateOffset ロボット番号, 部品種別, 関節番号, オフセット値

パラメーター

ロボット番号 ロボット番号を整数値 (1-16) で指定します。

部品種別 ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 関節を整数値 (1-6) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にもオフセットを設定します。本コマンドは、付加軸に対しては使用できません。

オフセット値 消耗率に対してオフセットする整数値を指定します。(単位: %)

解説

指定した部品種別、関節に対して、消耗率のオフセットを設定します。

参照

HealthRBAAlarmOn, HealthRateRBInfo, HealthRBInfo

HealthRBRateOffset 使用例

ロボット 1 の第 1 関節の減速機の消耗率に対して 10 % 加算する例です。

```
> HealthRBRateOffset 1,HEALTH_ROBOT_TYPE_GEAR,1,10
>
```

HealthRBReset

指定した部品種別の交換推奨時期までの残り月数、および消耗率をクリアします。

書式

HealthRBReset ロボット番号, 部品種別, 関節番号

パラメーター

ロボット番号 ロボット番号を整数値 (1-16) で指定します。

部品種別 ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号 交換推奨時期までの残り月数を返す関節を整数値 (1-6) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にもクリアします。本コマンドは、付加軸に対しては使用できません。

解説

指定した部品種別、関節に対して、交換推奨時期までの残り月数、および消耗率をクリアします。

警告も同時に解除します。

参照

HealthRBArmOn, HealthRateRBInfo, HealthRBInfo

HealthRBReset 使用例

```
> HealthRBReset 1, HEALTH_ROBOT_TYPE_GEAR, 1
>
```

HealthRBSpeed

指定した関節の速度の平均値を表示します。

書式

HealthRBSpeed [ロボット番号] [, 関節番号]

パラメーター

ロボット番号	ロボット番号を整数値 (1-16) で指定します。 省略可能です。省略された場合は、カレントロボット番号を使用します。
関節番号	関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStart から HealthRBStop までのロボット動作に対して、指定した関節の速度の絶対値の平均を表示します。結果は 0~1 までの実数値で表されます。最大平均速度値は “1” になります。平均値が 0.001 以下の値になる場合は “0” になります。

注意

-
- Auto モードでは、機能しません。
 - ドライラン (仮想コントローラー含む) では、機能しません。
-

参照

HealthRBStart, HealthRBStop, AveSpeed

HealthRBSpeed 使用例

スカルロボットの第 1 関節の速度を表示する例です。

```
> HealthRBSpeed 1, 1
0.100
>
```

HealthRBSpeed 関数

指定した関節の、速度の絶対値の平均値を返します。

書式

HealthRBSpeed ([ロボット番号,] 関節番号)

パラメーター

- ロボット番号** ロボット番号を整数値 (1-16) で指定します。
省略可能です。省略された場合は、カレントロボット番号を使用します。
- 関節番号** 関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

0~1 までの実数値で返します。

解説

HealthRBStart から HealthRBStop までのロボット動作に対して、指定した関節の速度の絶対値の平均値を返します。結果は 0~1 までの実数値で表されます。最大平均速度は 1 になります。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

HealthRBStart, HealthRBStop, AveSpeed

HealthRBSpeed 関数使用例

```
Function RobotPartAnalysis
  Real healthSpeed

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthSpeed = HealthRBSpeed(1,1)
  Print "AveSpeed =", Str$(healthSpeed)
End
```

HealthRBStart

あるサイクルのロボット動作に対する部品の使用可能月数と要素の演算を開始します。

書式

HealthRBStart ロボット番号

パラメーター

ロボット番号 ロボット番号を整数値 (1-16) で指定します。

解説

指定されたロボット番号に対する部品の使用可能月数と要素 (トルク, 速度, 駆動量) の演算を開始します。
演算を開始している状態で、再度実行すると、前回の演算結果は初期化されます。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

HealthRBAnalysis, HealthRBStop, HealthRBTRQ, HealthRBSpeed, HealthRBDistance

HealthRBStart 使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
End
```

HealthRBStop

あるサイクルのロボット動作に対する部品の使用可能月数と要素の演算を終了します。

書式

HealthRBStop ロボット番号

パラメーター

ロボット番号 ロボット番号を整数値 (1-16) で指定します。

解説

指定されたロボット番号に対する部品の使用可能月数と要素 (トルク, 速度, 駆動量) の演算を終了します。

注意

- Auto モードでは、機能しません。
 - ドライラン (仮想コントローラー含む) では、機能しません。
 - 演算開始して 1 時間過ぎると、自動的に終了します。
自動的に終了後、このコマンドを実行するとエラーになります。
 - HealthRBStart コマンドを実行せずに、このコマンドを実行するとエラーになります。
 - HealthRBStop 実行後に HealthRBStart を実行せずに、再度 HealthRBStop を実行した場合にもエラーになります。
-

参照

HealthRBAnalysis, HealthRBStart, HealthRBTRQ, HealthRBSpeed, HealthRBDistance

HealthRBStop 使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

HealthRBTRQ

指定した関節の寿命影響トルク値を表示します。

書式

HealthRBTRQ [ロボット番号] [, 関節番号]

パラメーター

ロボット番号	ロボット番号を整数値 (1-16) で指定します。 省略可能です。省略された場合は、カレントロボット番号を使用します。
関節番号	関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStart から HealthRBStop までのロボット動作に対して、指定した関節の寿命影響トルク値を表示します。結果は 0~1 までの実数値で表されます。最大トルク値は “1” になります。

注意

-
- Auto モードでは、機能しません。
 - ドライラン (仮想コントローラー含む) では、機能しません。
-

参照

HealthRBStart, HealthRBStop, ATRQ

HealthRBTRQ 使用例

スカルロボットの第 1 関節の寿命影響トルク値を表示する例です。

```
> HealthRBTRQ 1, 1
0.020
>
```

HealthRBTRQ 関数

指定した関節の寿命影響トルク値を返します。

書式

HealthRBTRQ ([ロボット番号,] 関節番号)

パラメーター

ロボット番号	ロボット番号を整数値 (1-16) で指定します。 省略可能です。省略された場合は、カレントロボット番号を使用します。
関節番号	関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

0~1 までの実数値で返します。

解説

HealthRBStart から HealthRBStop までのロボット動作に対して、指定した関節の寿命影響トルク値を返します。結果は 0~1 までの実数値で表わされます。最大実効トルクは 1 になります。

注意

- Auto モードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

HealthRBStart, HealthRBStop, ATRQ

HealthRBTRQ 関数使用例

```
Function RobotPartAnalysis
  Real healthTRQ

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthTRQ = HealthRBTRQ(1,1)
  Print "Torque =", Str$(healthTRQ)
End
```


HealthRBWarningEnable

ロボットに関する指定した種別の部品の部品消耗アラームの通知を有効または無効にします。

書式

HealthRBWarningEnable ロボット番号, 部品種別[, On/Off]

パラメーター

- ロボット番号 ロボット番号を整数値 (1-16)で指定します。
- 部品種別 ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

- On/Off On: 部品消耗アラームの通知を有効にします。
Off: 部品消耗アラームの通知を無効にします。

結果

On/Off パラメーターを省略すると、現在の On/Off 設定値が表示されます。

解説

指定した種別の部品の部品消耗アラームが発生した場合、部品消耗アラーム通知を行うかどうかを設定します。

注意

指定した部品の部品消耗アラームの通知を無効にした場合、交換推奨時期を経過してもワーニングが通知されません。本コマンドによる設定には十分注意して実行してください。

参照

HealthRBAAlarmOn

HealthRBWarningEnable 使用例

スカラロボット 1 の、グリス部品の部品消耗アラーム通知を無効にする例です。

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE, Off
```

スカラロボット 1 の、グリス部品の部品消耗アラーム通知設定を表示する例です。

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE
GREASE Off
>
```

HealthRBWarningEnable 関数

ロボットに関する指定した種別の部品の部品消耗アラームの通知の設定状態を返す関数です。

書式

HealthRBWarningEnable(ロボット番号, 部品種別)

パラメーター

ロボット番号 交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16) で指定します。

部品種別 交換推奨時期までの残り月数を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

戻り値

部品消耗アラームの設定値を整数値で返します。

1: On

0: Off

参照

HealthRBAAlarmOn

HealthRBWarningEnable 関数使用例

スカルロボット 1 のグリス部品の部品消耗アラームの通知設定状態を表示する例です。

```
Print HealthRBWarningEnable(1, HEALTH_ROBOT_TYPE_GREASE )
```

Here

現在位置をロボット座標として教示します。

書式

Here 座標

パラメーター

座標 P 番号, P(式), ポイントラベルのいずれかを指定します。

注意

Here ステートメントと並列処理

Here ステートメントと並列処理を 1 つの動作命令内で混在させることはできません。

```
Go Here :Z(0) ! D10; MemOn 1 !
```

上記のような使い方はできません。

```
P999 = Here
```

```
Go P999 Here :Z(0) ! D10; MemOn 1 !
```

上記のようなプログラムに変更してください。

Here ステートメントとマルチタスク

Here ステートメントを Xqt などで行ったマルチタスク関数内で実行したとき、メインタスク内で Move, Go などによってロボットが動作中である場合、エラー停止します。

CurPos を使うことで、動作中の現在位置が取得できます。

例

```
Function Xqt_PrintHere
Do
    Print CurPOS
    Wait 0.1
Loop
Fend
Function main
Xqt 10, Xqt_PrintHere
Go P0
Fend
```

参照

Here 関数, CurPos

Here 使用例

```
Here P1
Here pick
```

Here 関数

ロボットの現在位置を返します。

書式

Here

戻り値

ロボットの現在位置を返します。

解説

Here 関数は使用中のマニピュレーターの現在位置を取得するために使います。

参照

Here 関数

Here 関数使用例

```
P1 = Here
```

Hex\$関数

16 進数表現の数値を文字化して返します。

書式

Hex\$(数値)

パラメーター

数値 整数値を指定します。

戻り値

16 進数表現の数値を ASCII 値の文字列として返します。

解説

Hex\$関数は、16 進数表現の数値を文字化して返します。文字は 0~9, A~F で構成します。Hex\$は Stat 関数の結果確認に使用します。

参照

Str\$, Stat, Val

Hex\$ 関数使用例

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

Hofs

エンコーダー原点からソフトウェア原点への補正パルスの設定と表示をします。

書式

- (1) Hofs 第 1 関節設定値, 第 2 関節設定値, 第 3 関節設定値, 第 4 関節設定値
[, 第 5 関節設定値, 第 6 関節設定値] [, 第 7 関節設定値] [, 第 8 関節設定値, 第 9 関節設定値]
- (2) Hofs

パラメーター

第 1 関節設定値	第 1 関節補正パルス値 (整数)を、式または数値で指定します。
第 2 関節設定値	第 2 関節補正パルス値 (整数)を、式または数値で指定します。
第 3 関節設定値	第 3 関節補正パルス値 (整数)を、式または数値で指定します。
第 4 関節設定値	第 4 関節補正パルス値 (整数)を、式または数値で指定します。
第 5 関節設定値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。第 5 関節補正パルス値 (整数)を、式または数値で指定します。
第 6 関節設定値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。第 6 関節補正パルス値 (整数)を、式または数値で指定します。
第 7 関節設定値	ジョイント型 7 軸ロボット用のパラメーターです。第 7 関節補正パルス値 (整数)を、式または数値で指定します。
第 8 関節設定値	付加軸 S 関節用のパラメーターです。第 8 関節 (付加軸 S)補正パルス値 (整数)を、式または数値で指定します。
第 9 関節設定値	付加軸 T 関節用のパラメーターです。第 9 関節 (付加軸 T)補正パルス値 (整数)を、式または数値で指定します。

戻り値

パラメーターの指定がないときは、現在の Hofs 設定値を表示します。

解説

Hofs は、原点補正パルス値を表示または設定します。Hofs は、エンコーダー原点 (Z 相)からメカ原点までのオフセット値を設定します。

ロボット動作制御は、各関節に搭載されたエンコーダーの原点に基づいていますが、エンコーダー原点がロボットのメカ原点と必ずしも一致するわけではありません。そこで、メカ原点と一致するエンコーダーの位置をソフトウェア上の原点とするために、Hofs により補正パルス量を設定します。

注意

Hofs 値は、必要な時以外、絶対に変更しないでください

Hofs 値は工場出荷時に精密に設定されています。不必要にこの値を変更すると、位置決めエラーや思わぬ動作の原因になり危険です。必要な時以外は、Hofs 値を絶対に変更しないようにしてください。

JointAccuracy のリセット(関節精度補正対応機種のみ)

関節精度補正対応機種では、Hofs により原点補正パルス値の設定を行うと、変更があった関節に対しては、JointAccuracy で設定した関節精度補正の補正值が“0”にリセットされます。JointAccuracy で設定した補正值をリセットしたくない場合は、HofsJointAccuracy を使用してください。

Hofs 値を自動的に演算するには

Hofs 値を自動的に演算するためには、アームをキャリブレーションしたい位置まで動かし、Calib を実行します。そうすると、CalPls パルス値およびキャリブレーション位置パルス値に基づき、コントローラーが自動的に Hofs 値を演算します。

Hofs の保存と復元

Hofs は、[システム設定]メニュー-[システム設定]ダイアログ-[ロボット]-[キャリブレーション]にある[保存]および[読み込み]を使って保存および復元することができます。

本コマンド実行後は、安全機能マネージャーを起動してください (Safety基板を搭載したコントローラーのみ)

Safety基板を搭載したコントローラーでは、コントローラーのHofs値と安全機能を実現するSafety基板のHofs値が一致する必要があります。

本コマンドを実行すると、コントローラーのHofs値のみが変更され、Safety基板設定と差異が生じるため、ワーニングが発生します。

このため、本コマンド実行後は、Safety基板設定を更新するために、安全機能マネージャーを起動してください。

詳細は、以下のマニュアルを参照してください。

ロボットコントローラー 安全機能マニュアル

参照

Calib, CalPls, JointAccuracy, HofsJointAccuracy, Home, Hordr, MCal, SysConfig

Hofs 使用例

下記は、モニターウィンドウでの簡単な使用例です。第 1 関節原点補正值を-545 に、第 2 関節原点補正值を 514 に、第 3 関節と第 4 関節の原点補正值を 0 に設定し、その後、現在の原点補正值を表示します。

```
> hofs -545, 514, 0, 0

> hofs
-545, 514, 0, 0
>
```


Hofs 関数

各関節のエンコーダー原点からソフトウェア原点への補正パルス値を返す関数です。

書式

Hofs(関節番号)

パラメーター

関節番号 Hofs 値を求める関節番号 (整数)を、式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

指定した関節の補正パルス値 (整数値、単位: パルス)を返します。

参照

Calib, CalPIs, Home, Hordr, MCal, SysConfig

Hofs 関数使用例

下記は Hofs 関数を使ったプログラムの例です。

```
Function DisplayHofs
  Integer i

  Print "Hofs settings:"
  For i = 1 To 4
    Print "Joint ", i, " = ", Hofs(i)
  Next i
End
```

HofsJointAccuracy

関節精度補正の補正値を変更せずに、エンコーダー原点からソフトウェア原点への補正パルスの設定と表示をします。

書式

- (1) HofsJointAccuracy 第 1 関節設定値, 第 2 関節設定値, 第 3 関節設定値, 第 4 関節設定値
[, 第 5 関節設定値, 第 6 関節設定値] [, 第 7 関節設定値] [, 第 8 関節設定値, 第 9 関節設定値]
(2) HofsJointAccuracy

パラメーター

第 1 関節設定値	第 1 関節補正パルス値 (整数)を、式または数値で指定します。
第 2 関節設定値	第 2 関節補正パルス値 (整数)を、式または数値で指定します。
第 3 関節設定値	第 3 関節補正パルス値 (整数)を、式または数値で指定します。
第 4 関節設定値	第 4 関節補正パルス値 (整数)を、式または数値で指定します。
第 5 関節設定値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。 第 5 関節補正パルス値 (整数)を、式または数値で指定します。
第 6 関節設定値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。 第 6 関節補正パルス値 (整数)を、式または数値で指定します。
第 7 関節設定値	ジョイント型 7 軸ロボット用のパラメーターです。 第 7 関節補正パルス値 (整数)を、式または数値で指定します。
第 8 関節設定値	付加軸 S 関節用のパラメーターです。 第 8 関節 (付加軸 S)補正パルス値 (整数)を、式または数値で指定します。
第 9 関節設定値	付加軸 T 関節用のパラメーターです。 第 9 関節 (付加軸 T)補正パルス値 (整数)を、式または数値で指定します。

戻り値

パラメーターの指定がないときは、現在の Hofs 設定値を表示します。

解説

HofsJointAccuracy は、関節精度補正の補正値を変更せずに、原点補正パルス値を表示、または設定します。関節精度補正対応機種については、マニピュレーターマニュアルを参照してください。

ロボット動作制御は、各関節に搭載されたエンコーダーの原点に基づいていますが、エンコーダー原点がロボットのメカ原点と必ずしも一致するわけではありません。そこで、メカ原点と一致するエンコーダーの位置をソフトウェア上の原点とするために、Hofs により補正パルス量を設定します。

Hofs は各関節について変更がある場合は、それぞれの JointAccuracy で設定した関節精度補正の補正値を“0”にリセットしますが、HofsJointAccuracy ではリセットを行いません。関節精度補正の補正値を“0”にリセットしたくない場合は、HofsJointAccuracy を使用してください。

注意**Hofs 値は、必要な時以外、絶対に変更しないでください**

Hofs 値は、工場出荷時に精密に設定されています。不必要にこの値を変更すると、位置決めエラーや思わぬ動作の原因になり危険です。必要な時以外は、Hofs 値を絶対に変更しないようにしてください。

参照

JointAccuracy, Hofs

HofsJointAccuracy 使用例

下記は、コマンドウィンドウでの簡単な使用例です。第 1 関節原点補正値を“-545”に、第 2 関節原点補正値を“514”に、第 3 関節と第 4 関節の原点補正値を“0”に設定し、その後、現在の原点補正値を表示します。HofsJointAccuracy の使用前後で関節精度補正の補正値は変化していません。

```
> JointAccuracy 1
1000, 420, 100, 240
> HofsJointAccuracy -545, 514, 0, 0

> HofsJointAccuracy
-545, 514, 0, 0
> JointAccuracy 1
1000, 420, 100, 240
>
```

Home

ロボットアームをユーザー定義のホーム位置まで移動します。

書式

Home

解説

HomeSet で指定されたホーム位置 (待機姿勢) まで、Hordr で設定された関節順序にしたがって、低速の PTP 動作で移動します。

水平多関節型ロボット (RS シリーズを含む) では、通常、第 3 関節が 1 番最初に HomeSet 位置まで戻ってから、第 1 関節, 第 2 関節, 第 4 関節が、同時にそれぞれの HomeSet 座標位置まで戻ります。ホーム位置に戻る関節の順番は、Hordr 命令で変更することができます。

注意

Home 状態の出力

ロボットがホーム位置 (待機姿勢) にあるとき、コントローラーの Home 出力はオンになります。

起こりやすいエラー

HomeSet 値を設定しないで Home を実行した場合

HomeSet 値を設定しないで Home を実行すると、エラー 2228 が発生します。

参照

HomeClr, HomeDef, Hordr, HomeSet

Home 使用例

Home 命令は、下記のようなプログラムで使用できます。

```
Function InitRobot
  Reset
  If Motor = Off Then
    Motor On
  EndIf
  Home
Fend
```

あるいは、コマンドウィンドウから下記のように発行します。

```
> home
>
```

HomeClr 関数

ホーム位置の設定をクリアします。

書式

HomeClr

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

HomeDef, HomeSet

HomeClr 使用例

HomeClr 関数は、下記のようなプログラムで使用できます。

```
Function ClearHome  
  
    If HomeDef = True Then  
        HomeClr  
    EndIf  
End
```

HomeDef 関数

ホーム位置が設定されたかどうか返します。

書式

HomeDef

戻り値

設定された場合は“True”、設定されていない場合は“False”を返す。

参照

HomeClr, HomeSet

HomeClr 使用例

HomeDef 関数は、下記のようなプログラムで使用できます。

```
Function DisplayHomeSet

    Integer i

    If HomeDef = False Then
        Print "Home is not defined"
    Else
        Print "Home values:"
        For i = 1 To 4
            Print "J", i, " = ", HomeSet(i)
        Next i
    EndIf
End
```

HomeSet

ホーム位置 (待機姿勢)の姿勢パルスの設定と表示をします。

書式

(1) HomeSet 第 1 関節パルス値, 第 2 関節パルス値, 第 3 関節パルス値, 第 4 関節パルス値 [, 第 5 関節パルス値, 第 6 関節パルス値] [, 第 7 関節パルス値] [, 第 8 関節パルス値, 第 9 関節パルス値]

(2) HomeSet

パラメーター

第 1 関節パルス値	第 1 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 2 関節パルス値	第 2 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 3 関節パルス値	第 3 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 4 関節パルス値	第 4 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 5 関節パルス値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。第 5 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 6 関節パルス値	垂直 6 軸型ロボット (N シリーズを含む)用のパラメーターです。第 6 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 7 関節パルス値	ジョイント型 7 軸ロボット用のパラメーターです。第 7 関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 8 関節パルス値	付加軸 S 関節用のパラメーターです。第 8 関節 (付加軸 S)ホームエンコーダーパルス値 (整数)を、式または数値で指定します。
第 9 関節パルス値	付加軸 T 関節用のパラメーターです。第 9 関節 (付加軸 T)ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

結果

パラメーターを省略すると現在のホームパルス値を表示します。

解説

ユーザーは、各関節に対するパルス値を指定し、ホーム (待機)位置を設定することができます。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

起こりやすいエラー

HomeSet 値を設定せずに Home を実行したとき

HomeSet 値を設定せずに Home を実行すると、エラー 2228 が発生します。

HomeSet 値を設定せずに HomeSet 値を表示させようとしたとき

HomeSet 値を設定せずにホーム位置のパルス値を表示しようとすると、エラー 2228 が発生します。

参照

Home, HomeClr, HomeDef, Hordr, Pls

HomeSet 使用例

下記はコマンドウィンドウからの操作例です。

```
> homeset 0,0,0,0   'ホーム位置を 0,0,0,0 に設定
> homeset
  0  0
  0  0

> home               'ロボットがホーム位置へ移動
```

Pls 関数を使って、アームの現在位置をホーム位置と設定します。

```
> homeset Pls(1), Pls(2), Pls(3), Pls(4)
```


HomeSet 関数

指定された関節のホーム位置 (待機姿勢) のパルス値を返す関数です。

書式

HomeSet(関節番号)

パラメーター

関節番号 HomeSet 値を求める関節番号を、式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

指定関節のホームパルス値を返します。関節番号に“0”を指定すると、HomeSet 値の設定がある場合は、“1”を、設定がない場合は“0”を返します。

参照

HomeSet

HomeSet 関数使用例

下記は、HomeSet 関数を使ったプログラムの例です。

```
Function DisplayHomeSet

    Integer i

    If HomeSet(0) = 0 Then
        Print "HomeSet is not defined"
    Else
        Print "HomeSet values:"
        For i = 1 To 4
            Print "J", i, " = ", HomeSet(i)
        Next i
    EndIf
Fend
```

Hordr

Home 実行時の各関節の動作順序の指定と表示をします。

書式

(1) Hordr 設定値 1, 設定値 2, 設定値 3, 設定値 4 [, 設定値 5] [, 設定値 6] [, 設定値 7] [, 設定値 8] [, 設定値 9]

(2) Hordr

パラメーター

設定値 1	第 1 ステップで復帰する関節をビットパターンで指定します。
設定値 2	第 2 ステップで復帰する関節をビットパターンで指定します。
設定値 3	第 3 ステップで復帰する関節をビットパターンで指定します。
設定値 4	第 4 ステップで復帰する関節をビットパターンで指定します。
設定値 5	第 5 ステップで復帰する関節をビットパターンで指定します。
設定値 6	第 6 ステップで復帰する関節をビットパターンで指定します。
設定値 7	第 7 ステップで復帰する関節をビットパターンで指定します。
設定値 8	第 8 ステップで復帰する関節をビットパターンで指定します。
設定値 9	第 9 ステップで復帰する関節をビットパターンで指定します。

結果

パラメーターを省略すると現在の復帰順序設定を表示します。

解説

Hordr は、Home コマンド実行時の関節の復帰順序を設定します。どの関節が何番目にホーム位置 (待機姿勢) へ移動するかが決まります。

Hordr 命令は、Home 実行時の各関節の復帰順序を、ユーザーが変更できるようにします。復帰順序は、ロボットの型によって、4 つ、6 つ、あるいは 9 つのステップに分かれて設定されます。ユーザーは Hordr により各ステップで復帰する関節を指定することができます。各ステップで復帰する関節は、複数を指定することも可能です。理論的には全関節が同時に復帰することもあり得ます。しかし、水平多関節型ロボット (RS シリーズを含む 4 軸ロボット) の場合、一般的には、第 3 関節を最初にステップ 1 で移動させ、他の関節をそれ以降のステップで復帰させることをお勧めします。

Hordr 命令では、各ステップに対しビットパターンが指定されなければなりません。各関節に対し、ビットパターンが定められています。あるステップで、ビットが“1”であれば、対応する関節がホーム位置 (待機姿勢) へ移動します。ビットを“0”にクリアすると、対応する関節は、そのステップにおいてはホーム位置 (待機姿勢) へ移動しません。各関節に対するビットパターンは次のように割り当てられています。

ビットパターン表

関節名:	第 1 関節	第 2 関節	第 3 関節	第 4 関節	第 5 関節	第 6 関節
ビット番号:	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5
2 進数表記:	&B0001	&B0010	&B0100	&B1000	&B10000	&B100000
関節名:	第 7 関節	第 8 関節	第 9 関節			
ビット番号:	bit 6	bit 7	bit 8			
2 進数表記:	&B1000000	&B10000000	&B100000000			

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

Home, HomeSet

Hordr 使用例

下記はコマンドウィンドウからの水平多関節型ロボット (RS シリーズを含む 4 軸ロボット) の操作例です。

この例では、ロボットアームのホーム位置への復帰順序を 2 進数で次のように設定します

第 1 ステップで第 3 関節、第 2 ステップで第 1 関節、第 3 ステップで第 2 関節、第 4 ステップで第 4 関節がホーム位置へ復帰します。

```
>hordr  &B0100, &B0001, &B0010, &B1000
```

この例では、ロボットアームのホーム位置への復帰順序を 10 進数で次のように設定します

第 1 ステップで第 3 関節、第 2 ステップで第 1 関節、第 2 関節、第 4 関節が同時にホーム位置へ復帰します。

```
>hordr  4, 11, 0, 0
```

次の例では、現在の復帰順序を 10 進数で表示しています。

```
>hordr
4, 11, 0, 0
>
```

Hordr 関数

Home 実行時の、指定された各関節の動作順序を返します。

書式

Hordr(設定値番号)

パラメーター

設定値番号 Hordr 命令の動作順序を表わす整数値を指定します。

戻り値

指定した動作順序の Horder 設定値の数値を返します。

参照

Home, HomeSet

Hordr 関数使用例

```
Integer a  
a = Hordr(1)
```

Hour

コントローラーの積算通電時間の表示をします。

書式

Hour

解説

コントローラーの電源を投入し、積算時間を表示します。(積算通電時間)
積算時間の単位は時間です。

参照

Time

Hour 使用例

次の例は、コマンドウィンドウから実行します。

```
> hour  
2560  
>
```

Hour 関数

コントローラーの積算通電時間を返します。

書式

Hour

戻り値

コントローラーの積算通電時間を実数値で返します。

参照

Time

Hour 関数使用例

```
Print "Number of controller operating hours: ", Hour
```

If...Then...Else...EndIf

指定された条件分岐にしたがって命令を実行します。

書式

```
(1) If 条件式 Then
    ステートメント T1
    .
    .
    [Elseif 条件式 Then]
        ステートメント T1
    .
    .
    [Else]
        ステートメント F1
    .
    .
EndIf
```

```
(2) If 条件式 Then ステートメント T1 [:ステートメント T2...] [Else ステートメント F1 [:ステートメント F2...]]
```

パラメーター

条件式	真偽値 (True/False の値)が戻る有効な条件式です。真 (True)の場合は“0”以外の数値を、偽 (False)の場合は、“0”を返します。(下記の条件式例を参照してください。)
ステートメント T1	条件式の値が真 (True)のとき、つまり、条件が満たされたときに実行されます。(複数のステートメントを If...Then...Else 形式のブロックの中に記述できます。)
ステートメント F1	条件式の値が偽 (False)のとき、つまり、条件が満たされなかったときに実行されます。(複数のステートメントを If...Then...Else 形式のブロックの中に記述できます。)

解説

- (1) If...Then...Else は、条件が満たされたときにステートメント T1 以降を実行します。条件が満たされなかったときは、ステートメント F1 以降を実行します。If...Then...Else 命令の Else 部分は省略可能です。Else 部分を省略し、条件が満たされなかったときは、EndIf 以降のステートメントが実行されます。If...Then...Else ステートメントのブロックは、Else の有無にかかわらず、EndIf で閉じてください。
- (2) If...Then...Else は、ブロック以外の形式で用いることも可能です。If...Then...Else のステートメントをすべて同じ行に記述する方法です。この方法で If...Then...Else を用いる場合には、EndIf は必要ではありません。この行で記述されている条件式が満たされれば (条件式に対して真値 (True)であれば)、Then と Else 間のステートメントが実行されます。条件式が満たされなければ (条件式に対して偽値 (False)であれば)、Else に続くステートメントが実行されます。If...Then...Else の Else 部分は必須ではありません。条件式が満たされない場合 (条件式に対して偽値 (False)の場合)で Else キーワードがなければ、制御はプログラム中の次のステートメントに移行します。

注意**条件式サンプル**

a = b	: a と b が等しい
a < b	: a が b より小さい
a >= b	: a が b より大きいまたは等しい
a <> b	: a と b が等しくない
a > b	: a が b より大きい
a <= b	: a が b より小さいまたは等しい

論理演算子 And, Or, Xor など也可以使用できます。

条件式に True を使用する場合

定数 True の値は-1 ですが、Boolean 型なので、他の型の変数との比較条件に使用する場合は注意が必要です。

```
Function main
  Integer i
  i = 3
  If i = True Then
    Print "i=TRUE"
  EndIf
Fend
```

上記プログラムを実行すると “i=TRUE” が表示されます。

これは、Boolean 型を含む条件判定は、“0”か“非 0”かで行われるためです。

“i”の値が“0”でない場合は条件成立と判定され表示が行われます。

参照

Else, Select...Case, Do...Loop

If/Then/Else 使用例**<1 行の If...Then...Else>**

下記例は、特定の出力のオンオフを決定するために、入力をチェックする簡単な例です。このようなタスクは、常に動かしておく I/O タスクなどに使用します。

```
Function main
  Do
    If Sw(0) = 1 Then On 1 Else Off 1
  Loop
Fend
```


<ブロック形式の If...Then...Else>

下記は、いくつかの入力をチェックして、その入力の状態を出力する簡単な例です。

```
If Sw(0) = 1 Then Print "Input0 ON" Else Print "Input0 OFF"
,
If Sw(1) = 1 Then
  If Sw(2) = 1 Then
    Print "Input1 On and Input2 ON"
  Else
    Print "Input1 On and Input2 OFF"
  EndIf
Else
  If Sw(2) = 1 Then
    Print "Input1 Off and Input2 ON"
  Else
    Print "Input1 Off and Input2 OFF"
  EndIf
EndIf
```

<その他の書式の例>

```
If x = 10 And y = 3 Then GoTo 50
If test <= 10 Then Print "Test Failed"
If Sw(0) = 1 Or Sw(1) = 1 Then Print "Everything OK"
```

ImportPoints

現在選択中のロボットのプロジェクトにポイントファイルをインポートします。

書式

ImportPoints ソースパス,ファイル名 [, ロボット番号]

パラメーター

ソースパス	カレントプロジェクトにインポートしたい特定の軌道やファイルを表す文字列式 拡張子は、“.pts”、または “.pnt” (EPSON RC+ 3.x / 4.x 形式)です。 詳細は、ChDisk を参照してください。
ファイル名	カレントロボットのカレントプロジェクトへインポートさせたい特定のファイルを表す文字列式 拡張子は pts です。パスを指定することはできません。また、ChDisk などの影響を受けません。詳細は、ChDisk を参照してください。
ロボット番号	どのロボットがポイントファイルに関連するのかを指定する整数式 省略可能です。 ロボット番号=0 の場合、ポイントファイルは共通ポイントファイルとしてインポートされます。省略された場合は、カレントロボット番号を使用します。

解説

ImportPoints は、現在選択中のプロジェクトにポイントファイルをコピーして、そのポイントファイルを現在選択中のロボットのファイルに追加します。追加されたポイントファイルはコンパイルされ、LoadPoints コマンドで読み込み可能になります。現在選択中のロボットに同じファイルが既に存在していた場合は上書きされ、再コンパイルされます。

ポイントデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、ImportPoints を実行するとコンパクトフラッシュへの書き込みが発生します。あまりにも頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。必要な場合にのみ ImportPoints を実行することを推奨します。

起こりやすいエラー

ファイルが存在しない場合

指定したソースパスが存在しない場合、エラーが発生します。

指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。

指定したファイルが現在のロボットのファイルでない場合

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。

参照

LoadPoints, Robot, SavePoints

ImportPoints 使用例

```
Function main
  Robot 1
  ImportPoints "c:\mypoints\model1.pts", "robot1.pts"
  LoadPoints "robot1.pts"
End
```

In 関数

指定された入力ポートの状態をバイト単位で返します。バイトポートは、8 個の入力ビットで構成されます。

書式

In(バイトポート番号)

パラメーター

バイトポート番号 I/O のバイトポートを指定します。

戻り値

0~255 の整数を返します。戻り値は 8 ビットで、各ビットが入力ビット 1 個に対応します。

解説

In では、8 個の入力ビットの値を同時に見ることができます。In 命令は、8 個の I/O ビットの状態を 1 つの変数の値として保存したり、Wait 命令とともに用いることで、「2 個以上の I/O ビットの状態が特定の条件に一致するまで待機させる」といった使い方ができます。

1 度に 8 個の入力ビットをチェックすることができるので、戻り値は、0~255 の範囲の整数値になります。下記表で、戻り値の各整数値が、それぞれの入力ビットにどのように対応しているか参照してください。

入力ビット表 (バイトポート 0 使用時)

戻り値	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

入力ビット表 (バイトポート 2 使用時)

戻り値	23	22	21	20	19	18	17	16
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

参照

InBCD, MemIn, MemOff, MemOn, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

In 関数使用例

下記のプログラム例は、入力ビット 20, 21, 22, 23 がセンサーデバイスに接続されており、アプリケーションは、その各々から“準備 OK”を示すオン信号が戻ってから始動する、という仮定に基づいて記述されています。プログラム例では、バイトポート 2 の 8 個の入力ビットの状態を取得し、入力ビット 20, 21, 22, 23 がすべてオンであることを確認してから次に進んでいます。もし、いずれかの入力ビットがオンでないと(いずれかで“1”が返されると)、エラーメッセージが表示され、タスクは停止します。

この場合、入力ビット 20, 21, 22, 23 がすべてオンであるためには、In(2)の戻り値は、240 以上でなくてはなりません。それを確認するために、変数“var1”を数値 239 と比較しています。(ここでは、入力ビット 16, 17, 18, 19 については意識せず、戻り値が 240 から 255 内の値であれば、プログラムを進めています。)

```
Function main
  Integer var1
  var1 = In(2)  'バイトポート2の8個の入力ビットの状態を取得
  If var1 > 239 Then
    Go P1
    Go P2
    '他の動作コマンドをここで実行
    '.
    '.
  Else
    Print "Error in initialization!"
    Print "Sensory Inputs not ready for cycle start"
    Print "Please check inputs 20,21,22 and 23 for"
    Print "proper state for cycle start and then"
    Print "start program again"
  EndIf
Fend
```

コマンドウィンドウからは、入力を設定することはできませんが、確認することはできます。下記の例では、入力ビット 1, 5, 15 がオンであることを前提にしています。その他の入力はすべてオフになっています。

```
> print In(0)
34
> print In(1)
128
> print In(2)
0
```

InBCD 関数

入力ポートを 8 ビットごとのグループとして、入力ポートの状態を BCD で返す関数です。

書式

InBCD(ポート番号)

パラメーター

ポート番号 I/O の入力バイトを指定します。

戻り値

入力ポート (0~99) の状態を BCD (0~9) で返します。

解説

InBCD は、BCD で同時に 8 個の入力ラインを読みます。InBCD 命令のパラメーター“ポート番号”で、どのビットグループの状態を読むか指定します。例えば、ポート番号=0 のとき 0~7 の入力ビットの状態を、ポート番号=1 のとき 8~15 の入力ビットの状態を読む、ということになります。

8 個の入力ビットの状態は、BCD で返されます。戻り値は、0~99 の 1 または 2 桁の数値です。1 桁目 (10 の位) は、ポート番号で指定した 8 個の出力ビットのうち上位 4 ビットに対応します。2 桁目 (1 の位) は、ポート番号で指定した 8 個の出力ビットのうち下位 4 ビットに対応します。

各桁における BCD の有効値は 0~99 なので、I/O のすべての組み合わせを満たすことができません。下記表でポート番号=0 のとき、可能な I/O 状態の組み合わせと戻り値を、一部示しています。

入力設定 (入力番号)

戻り値	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

BCD では、10 進数しか指定することができません。このことは、特定のポートですべての入力ビットが同時にオンであるようなとき、BCD を使う InBCD 命令では有効な値を求めることは不可能だということを意味します。InBCD では最大戻り値は、“99”です。上記の表からもわかるように、“99”が戻り値のときには、すべての入力がオンにはなりません。戻り値“99”のときオンの入力ビットは、0, 3, 4, 7 で、その他のビットはすべてオフになっています。

注意

InBCD と In の違い

SPEL+では、InBCD と In 命令はいくつか重要な点で異なりますので、その違いを下記に示します。

- InBCD 命令は、8 個の入力ビットについての戻り値は BCD で指定されます。BCD では &HA, &HB, &HC, &HD, &HE, &HF などの値が使えないので、8 個の入力ビットの全組み合わせを満たすことができません。
 - In 命令は、InBCD 命令と非常に似た機能をしますが、8 個の入力ビットすべてについて戻り値を返すことができます。(In 命令の戻り値 0~255 に対して、InBCD の戻り値は 0~99 です。)したがって 1 グループ 8 個の入力ビット全組み合わせを満たすことができます。
-

参照

In, MemOff, MemOn, MemOut, MemSw, Off, On, OpBCD, Oport, Out, Sw, Wait

InBCD 使用例

コマンドウィンドウからの簡単な操作例です。

入力ビット 0, 4, 10, 16, 17, 18 がオンで、他の入力ビットはオフだと仮定しています。

```
> Print InBCD (0)
11
> Print InBCD (1)
04
> Print InBCD (2)
07
>
```

Inertia

ロボットの負荷イナーシャと偏心量を設定します。

書式

Inertia [負荷イナーシャ [, 偏心量]]

Inertia

パラメーター

負荷イナーシャ	ハンドおよびワークを含む、先端関節の中心周りの慣性モーメント (イナーシャ) (実数、単位: kgm^2) を、数値または式で指定します。省略可能 (負荷イナーシャのみを省略することは不可能) です。
偏心量	ハンドおよびワークの重心の先端関節中心からの距離 (実数、単位: mm) を、数値または式で指定します。省略可能です。

結果

パラメーターを省略すると、現在設定されているイナーシャパラメーターを表示します。

[偏心量] を省略すると、入力された [負荷イナーシャ] が設定され、デフォルト値の [偏心量] が設定されます。

[負荷イナーシャ] のみを省略することはできません。

イナーシャ値や偏心量に、実際より小さい値を指定すると、過剰な加速値および減速値が設定され、マニピュレーターに損傷を与えることがありますので注意してください。また、エラーの発生や衝撃の原因となり、機能が発揮できないだけでなく、部品の寿命が低下したり、ベルトの歯飛びによる位置ズレが発生したりする可能性があります。

解説

先端関節周りの慣性モーメントを指定するとき、Inertia ステートメントを使います。これにより先端関節の加減速およびサーボゲインを適正に補正することができます。また、偏心量パラメーターで、先端関節中心からハンドとワークの重心までの距離を指定することができます。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

“負荷, イナーシャ, 偏心/オフセット測定ユーティリティ” による設定も可能です。

詳細は、以下のマニュアルに記載しています。

EPSON RC+ 7.0 ユーザーズガイド 6.18.12 負荷, イナーシャ, 偏心/オフセット測定ユーティリティ

注意

Inertia の設定値は、電源オフでも変更されません

Inertia の設定値は、一度設定するとコントローラーに記憶されます。電源をオフしても変更されません。何も設定しない場合は、前回設定された値のままになります。

参照

Inertia 関数

ハンドについては、Hand 機能マニュアルを参照してください。

Inertia 使用例

Inertia 0.02, 1

Inertia 関数

イナーシャのパラメーター値を返します。

書式

Inertia(パラメーター番号)

パラメーター

パラメーター番号 下記の整数値を指定します。

- 0: ロボットがイナーシャパラメーターをサポートするときは“1”、サポートしないときは“0”を返します。
- 1: 負荷イナーシャを返します。(単位: kgm²)
- 2: 偏心量を返します。(単位: mm)

戻り値

指定した設定を実数値で返します。

参照

Inertia

ハンドについては、Hand 機能マニュアルを参照してください。

Inertia 関数使用例

```
Real loadInertia, eccentricity  
  
loadInertia = Inertia(1)  
eccentricity = Inertia(2)
```


InPos 関数

ロボットの位置決め状態を返します。

書式

InPos

戻り値

位置決めが完了した場合	True
ロボット動作中	False

参照

CurPos, FindPos, WaitPos

InPos 関数使用例

```
Function main

  P0 = XY(0, -100, 0, 0)
  P1 = XY(0, 100, 0, 0)

  Xqt MonitorPosition
  Do
    Jump P0
    Wait .5
    Jump P1
    Wait .5
  Loop

Fend

Function MonitorPosition

  Boolean oldInPos, pos

  Do
    Pos = InPos
    If pos <> oldInPos Then
      Print "InPos = ", pos
    EndIf
    oldInPos = pos
  Loop

Fend
```

Input

ディスプレイデバイスから入力を受け取り、変数に保存します。

書式

Input 変数名 [, 変数名, 変数名, ...]

パラメーター

変数名 変数名を指定します。複数の変数を指定するときは、“,”で区切ります。この場合の“,”をデリミターと呼びます。

解説

ディスプレイデバイスからデータを受け取り、指定された変数に代入します。

命令の実行時には、ディスプレイデバイスにプロンプト“?”が表示されます。データを入力後、キーボードでリターンキーを押します。

注意

数値入力の規則

数値入力をする場合、デリミター以外の非数値データがあると、その非数値データと、それ以降のデータは切り捨てられます。

文字列入力の規則

文字列に代入するときは、数字とアルファベットが文字として扱われます。

その他の Input 命令に関する規則

- 複数の変数を代入先に指定するときは、それぞれに代入する数値データは、必ずデリミター“,”で区切ります。
- 数値変数と文字列変数を指定することができますが、入力データ型は代入先の変数型に適合していなければなりません。

起こりやすいエラー

指定された変数の数と入力データの数が合わないとき

複数の変数を指定すると、入力データの数は指定された変数の数と一致しなければなりません。命令で指定された変数の数とキーボードから受け取った数値データの数が合わないとき、エラー2505になります。

参照

Input #, Line Input, Line Input #, Print, String

Input 使用例

下記は、簡単な Input ステートメントの例です。

```
Function InputNumbers
  Integer A, B, C

  Print "Please enter 1 number"
  Input A
  Print "Please enter 2 numbers separated by a comma"
  Input B, C
  Print "A = ", A
  Print "B = ", B, "C = ", C
Fend
```

上記のプログラムを走らせると、次のセッションが行われます。

```
Please enter 1 number
?-10000
Please enter 2 numbers separated by a comma
?25.1, -10000
A = -10000
B = 25 C = -10000
```

Input

ファイル、通信ポート、データベース、またはデバイスから、文字列か数値をデータ入力し、変数に保存します。

書式

Input #ポート番号, 変数名 [, 変数名, 変数名,...]

パラメーター

ポート番号	ファイル、通信ポート、データベース、またはデバイスを示す ID 番号です。 ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。 通信ポート番号は、OpenCom (RS-232C)または OpenNet (TCP/IP)ステートメントで指定した番号です。 データベース番号は、OpenDB ステートメントで指定した番号です。 デバイス ID は以下の数値です。 21 RC+ 24 TP (TP1 のみ) 20 TP3
変数名	データを受け取る変数名を指定します。

解説

Input # 命令は、ポート番号で指定したデバイスから数値または文字列データを受け取り、そのデータを指定する変数に入力します。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

数値入力の規則

数値入力をする際、デリミター以外の非数値データがあると、その非数値データと、それ以降のデータは切り捨てられます。

文字列入力の規則

文字列に代入するときは、数字とアルファベットが文字として扱われます。

最大データ長

本命令が一度にあつかえる最大データ長は、256 Byte です。

ただし、データベースが対象の場合は、4096 Byte が最大データ長になります。

通信ポート(TCP/IP)が対象の場合は、1024 Byte が最大データ長になります。

その他の Input # 命令に関する規則

- 複数の変数を代入先に指定するときは、それぞれに代入する数値データは、必ずデリミター (“;”)または空白 (“ ”)で区切ります。
- 複数の文字列変数、および数値変数と文字列変数の両方を指定するときには、数値データは、必ずデリミター (“;”)、または空白 (“ ”)で区切り、文字列データは必ずデリミター (“;”)で区切る必要があります。
- 入力データ型は代入先の変数型に適合していなければなりません。

コントローラー間で、通信ポートを使って文字列変数、数値変数をやりとりする例

送信側 (どちらのパターンでも OK です。)

```
Print #PortNum, "$Status,", InData, OutData
```

```
Print #PortNum, "$Status", ",", InData, OutData
```

受信側

```
Input #PortNum, Response$, InData, OutData
```

起こりやすいエラー

指定された変数の数と入力データの数合わないとき

命令で指定された変数の数とデバイスから受け取った数値データの数合わないとき、エラー2505 になります。

参照

Input, Line Input, Line Input #, Print #, Read, ReadBin

Input # 使用例

下記は Input # ステートメントを使った簡単な例です。

```
Function GetData
  Integer A
  String B$

  OpenCom #1
  Print #1, "Send"
  Input #1, A    'ポート#1 から数値を取得
  Input #1, B$   'ポート#1 から文字列を取得
  CloseCom #1
End
```

InputBox

入力ダイアログボックスを表示します。ユーザーがテキストに入力するか、またはボタンをクリックするまで待ち、入力した内容を返します。

書式

InputBox プロンプト, タイトル, デフォルト, 入力文字列

パラメーター

プロンプト	ダイアログボックスに表示するメッセージの文字列
タイトル	ダイアログボックスのタイトルバーに表示する文字列
デフォルト	テキストボックスにデフォルトで表示する文字列 デフォルト値を設定しない場合は、空白を設定します。 (“ ”)
入力文字列	ユーザーが入力した文字列をセットする文字列型の変数 ユーザーがキャンセルをクリックした場合は、この変数に “@” がセットされます。

解説

InputBox はダイアログボックスを表示し、ユーザーが<OK>ボタン、または<キャンセル>ボタンをクリックするまで待機します。パラメーターの入力文字列に、ユーザーが入力した文字列がセットされます。

参照

MsgBox

InputBox 使用例

このプログラムは InputBox ステートメントの使用例です。

```
Function GetPartName$ As String
    String prompt$, title$, answer$

    prompt$ = "Enter " + Chr$(34) + "part name" + Chr$(34) + ":"
    title$ = "サンプル"
    InputBox prompt$, title$, "", answer$

    If answer$ <> "@" Then
        GetPartName$ = answer$
    EndIf
End
```

次の図は前記のプログラム例の結果です。



制限事項

パラメーターのプロンプト、タイトル、およびデフォルトに、半角カンマ“,”が含まれる場合、文字列が正しく表示できません。

半角カンマを含まない文字列をご使用ください。

InReal 関数

2 ワード (32 ビット) の入力データを 32 ビット浮動小数点データ (IEEE754 準拠) として読み込みます。

書式

InReal(ワードポート番号)

パラメーター

ワードポート番号 I/O 入力ワードポートを指定します。

戻り値

入力ポートの状態を Real 型実数で返します。

解説

ワードポート番号で指定した入力ワードポートから 2 入力ワードの値を IEEE754 の Real 型値として取得します。ワードポート番号パラメーターには、入力ワードラベルを使用できます。
本関数は、Wait 命令、および Till, Find, Sense の条件として使用できません。

参照

In, InW, InBCD, Out, OutW, OpBCD, OutReal

InW 関数使用例

```
Real realVal  
  
realVal = InReal(32)
```

InsideBox 関数

進入検出エリアの検出状態を返します。

書式

InsideBox(エリア番号 [, ロボット番号 | All])

パラメーター

- エリア番号** 状態を返す進入検出エリアの番号 (1~15 の整数)を指定します。
- ロボット番号** 検出するロボットの番号を整数値で指定します。
 ロボット番号を省略した場合、現在選択されているロボットが対象となります。
 All を指定した場合は、1 台のロボットでも進入している場合は True を返します。

戻り値

指定した進入検出エリアにロボットの先端位置が進入している場合は True、進入していない場合は False を返します。

参照

Box, BoxClr, BoxDef, GetRobotInsideBox, InsidePlane

注意

EPSON RC+ 5.0 では、Wait 命令と組み合わせて InsideBox 関数の結果を待つことが可能でしたが、EPSON RC+ 6.0, RC+ 7.0 では Wait 命令と組み合わせることはできません。このような場合 InsideBox 関数の代わりに、GetRobotInsideBox 関数を使用してください。

対応表

RC+ バージョン	ロボット コントローラー	Wait	Till, Find, Sense, Trap	Print など、左に 属さないコマンド/ 分岐判定処理	GetRobotInsideBox 関数の利用
RC+ 7.0	RC700 シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC90 シリーズ	不可	不可	可能	すべて可能
RC+ 6.0	RC620	不可	不可	可能	すべて可能
RC+ 5.0	RC90 シリーズ	可能	不可	可能	不可

不可: 利用不可能な組み合わせ

可能: 利用可能な組み合わせ

すべて可能: Wait, Till, Find, Sense, Trap, Print などの表示、分岐判定処理に利用可能

InsideBox 関数使用例

下記のプログラム例は、エリア 3 番にロボット 1 が進入しているかを判断する例です。

```
Function PrintInsideBox
  If InsideBox(3,1) = True Then
    Print "Inside Box3"
  Else
    Print "Outside Box3"
  EndIf
Fend
```

InsidePlane 関数

進入検出平面の検出状態を返します。

書式

InsidePlane(平面番号[, ロボット番号 | All])

パラメーター

平面番号 状態を返す進入検出平面の番号 (1~15 の整数)を指定します。
ロボット番号 検出するロボットの番号を整数値で指定します。
 ロボット番号を省略した場合、現在選択されているロボットが対象となります。
 All を指定した場合は、1 台のロボットでも進入している場合は **True** を返します。

戻り値

指定した進入検出平面にロボットの手先位置が進入している場合は“True”、進入していない場合は“False”を返します。

参照

InsideBox, GetRobotInsidePlane, Plane, PlaneClr, PlaneDef

注意

EPSON RC+ 5.0 では、Wait 命令と組みあわせて InsidePlane 関数の結果を待つことが可能でしたが、EPSON RC+ 6.0, RC+ 7.0 では、Wait 命令と組みあわせることはできません。このような場合 InsidePlane 関数の代わりに、GetRobotInsidePlane 関数を使用してください。

対応表

RC+ バージョン	ロボットコントローラー	Wait	Till, Find, Sense, Trap	Print など、左に属さないコマンド / 分岐判定処理	GetRobotInsidePlane 関数の利用
RC+ 7.0	RC700 シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC90 シリーズ	不可	不可	可能	すべて可能
RC+ 6.0	RC620	不可	不可	可能	すべて可能
RC+ 5.0	RC90 シリーズ	可能	不可	可能	不可

不可: 利用不可能な組み合わせ

可能: 利用可能な組み合わせ

すべて可能: Wait, Till, Find, Sense, Trap, Print などの表示、分岐判定処理に利用可能

InsidePlane 関数使用例

下記のプログラム例は、平面 3 番にロボット 1 が進入しているかを判断する例です。

```
Function PrintInsidePlane
  If InsidePlane(3,1) = True Then
    Print "Inside Plane3"
  Else
    Print "Outside Plane3"
  EndIf
Fend
```

InStr 関数

文字列からの文字列検索をし、見つかった場所を返します。

書式

InStr(文字列, 検索文字列)

パラメーター

文字列 検索される文字列を指定します。

検索文字列 検索する文字列を指定します。

戻り値

検索する文字列が見つかった場合はその場所を、見つからなかった場合は-1 を返します。

参照

Mid\$

InStr 関数使用例

```
Integer pos
```

```
pos = InStr("abc", "b")
```

Int 関数

実数値を整数値に変換します。実数値の小数点以下を切り捨てた値を返します。

書式

Int(数値)

パラメーター

数値 実数値を指定します。

戻り値

パラメーターで設定した実数値を整数値に変換して返します。

解説

Int(数値) は実数値の小数点以下を切り捨てた値を返します。

注意

1 より小さい数値の場合 (負の値の場合)

パラメーターが 1 より小さい値の場合は、負の方向に切り捨てた値を返します。(数値が-1.35 の場合は、-2 を返します。)

参照

Abs, Atan, Atan2, Cos, Mod, Not, Sgn, Sin, Sqr, Str\$, Tan, Val

Int 関数使用例

次の例はコマンドウィンドウで実行します。

```
> Print Int(5.1)
5
> Print Int(0.2)
0
> Print Int(-5.1)
-6
>
```

Int32

Int32 型の変数を宣言します。(4 バイト整数型変数)

書式

Int32 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Int32 は整数型変数の宣言をします。整数型変数の範囲は-2147483648~2147483647 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Int32 使用例

次は Int32 を使って整数型の変数を宣言するプログラム例です。

```
Function int32test
    Int32 A(10)          'Int32 型の 1 次元配列
    Int32 B(10, 10)      'Int32 型の 2 次元配列
    Int32 C(5, 5, 5)     'Int32 型の 3 次元配列
    Int32 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

Int64

Int64 型の変数を宣言します。(8 バイト整数型変数)

書式

Int64 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Int64 は整数型変数の宣言をします。整数型変数の範囲は-9223372036854775808~9223372036854775807 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Integer, Long, Real, Short, String, UByte, UInt32, UShort, UInt64

Int64 使用例

次は Int64 を使って整数型の変数を宣言するプログラム例です。

```
Function int64test
    Int64 A(10)           'Int64 型の 1 次元配列
    Int64 B(10, 10)       'Int64 型の 2 次元配列
    Int64 C(5, 5, 5)      'Int64 型の 3 次元配列
    Int64 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

Integer

Integer 型の変数を宣言します。(2 バイト整数型変数)

書式

Integer 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Integer は整数型変数の宣言をします。整数型変数の範囲は-32768~32767 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Long, Real, Short, String, UByte, UInt32, UInt64, UShort

Integer 使用例

次は Integer を使って整数型の変数を宣言するプログラム例です。

```
Function inttest
    Integer A(10)           'Integer 型の 1 次元配列
    Integer B(10, 10)       'Integer 型の 2 次元配列
    Integer C(5, 5, 5)      'Integer 型の 3 次元配列
    Integer var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```


InW 関数

入力ポートの状態をワード単位で返します。ワードポートは 16 個の入力ビットで構成されます。

書式

InW(ワードポート番号)

パラメーター

ワードポート番号 I/O のワードポートを指定します。

戻り値

入力ポートの状態 (0~65535 の Long 型整数)を返します。

注意

リアルタイム I/O の入力ビットを含むワードポートに関する規則

ワードポート=1, 3, 17, 19 は、入力ポートの状態を 0~255 の整数で返します。
リアルタイム I/O の入力ビットは、反映されません。

参照

In, Out, OutW

InW 関数使用例

```
Long word0  
  
word0 = InW(0)
```

IODef 関数

指定した入出力のビット、バイト、ワードまたは I/O ラベルが定義されているかを返します。

書式

IODef (IO 型, IO 幅, ポート番号)

IODef (IO ラベル)

パラメーター

IO 型 I/O の型を表す整数値

0 – 入力

1 – 出力

2 – メモリー

IO 幅 ポートの幅を表す整数値: 1 (ビット), 8 (バイト), または 16 (ワード)

ポート番号 ラベルを返すビット、バイト、またはワードを表す整数値

IO ラベル 文字列で標準 I/O または、メモリー I/O ラベルを指定する文字列式です。

戻り値

指定した入出力のビット、バイト、ワード、または I/O ラベルが定義されている場合は“True”、それ以外は“False”を返します。

参照

IOLabel\$, IONumber

IODef 関数使用例

```
Integer i

For i = 0 To 15
  If IODef( 0, 1, i) = TRUE Then
    Print "Port " , i, " is defined"
  Else
    Print "Port " , i, " is undefined"
  EndIf
Next i
```

IOLabel\$ 関数

指定した入出力のビット、バイト、ワードの I/O ラベルを返します。

書式

IOLabel\$(IO 型, IO 幅, ポート番号)

パラメーター

IO 型	I/O の型を表す整数値 0 – 入力 1 – 出力 2 – メモリー
IO 幅	ポートの幅を表す整数値: 1 (ビット), 8 (バイト), 16 (ワード)
ポート番号	ラベルを返すビット、バイト、またはワードを表す整数値

戻り値

ラベルを含む文字列を返します。

参照

PLabel\$, IONumber, IODef

IOLabel\$ 関数使用例

```
Integer i

For i = 0 To 15
    Print "Input ", i, ": ", IOLabel$(0, 1, i)
Next i
```

IONumber 関数

指定した I/O ラベルの I/O ポート番号を返します。

書式

IONumber(IO ラベル)

パラメーター

IO ラベル 文字列で標準 I/O またはメモリー I/O ラベルを指定する文字列式です。

戻り値

指定した I/O ラベルの I/O ポート番号 (ビット, バイト, ワード) を返します。指定した I/O ラベルが存在しない場合は、エラーが発生します。

参照

IOLabel\$, IODef

IONumber 関数使用例

```
Integer IObit
```

```
IObit = IONumber ("myIO")
```

```
IObit = IONumber ("Station" + Str$(station) + "InCycle")
```

J1Angle

ポイントの J1Angle 属性を設定します。

書式

- (1) J1Angle ポイント指定 [, 設定値]
- (2) J1Angle

パラメーター

- ポイント指定** P 番号, P(式), ポイントラベルのいずれかを指定します。
- 設定値** 設定値を実数値で指定します。省略可能です。

結果

J1Angle 属性は、RS シリーズと N シリーズのロボットでのみ使われます。
設定値パラメーターを省略すると、指定ポイントに対する J1Angle 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J1Angle 値が表示されます。

RS シリーズ: ポイントの X 座標値が“0”で、かつ Y 座標値も“0”となる特異姿勢の第 1 関節角度値を指定します。特異姿勢とならないポイントでは、J1Angle 属性の値は意味を持ちません。

N シリーズ: 第 1 関節, 第 4 関節, 第 6 関節の軸心が直線上にある場合、第 1 関節と第 6 関節の軸心が直線上にある場合、または第 1 関節と第 4 関節の軸心が直線上にある場合の、第 1 関節角度値を指定します。特異姿勢とならないポイントでは、J1Angle 属性の値は意味を持ちません。

参照

Hand, J1Angle 関数, J1Flag, J2Flag, J4Angle, J4Angle 関数

J1Angle 使用例

```
J1Angle P0, 10.0
J1Angle P(mypoint), 0.0
```

J1Angle 関数

ポイントの J1Angle 属性を返します。

書式

J1Angle [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。
省略可能で、省略するとロボットの現在位置の J1Angle 設定を返します。

戻り値

J1Angle 属性は、RS シリーズと N シリーズのロボットでのみ使われます。

RS シリーズ: ポイントの X 座標値が“0”で、かつ Y 座標値も“0”となる特異姿勢の第 1 関節角度値を実数値で返します。

N シリーズ: 第 1 関節, 第 4 関節, 第 6 関節の軸心が直線状にある場合、第 1 関節と第 6 関節の軸心が直線状にある場合、または第 1 関節と第 4 関節の軸心が直線状にある場合の第 1 関節角度値を実数値で返します。

参照

Hand, J1Angle, J1Flag, J2Flag, J4Angle, J4Angle 関数

J1Angle 関数使用例

```
Print J1Angle (pick)
Print J1Angle (P1)
Print J1Angle
```

J1Flag

ポイントの J1Flag 属性を設定します。

書式

- (1) J1Flag ポイント指定 [, 設定値]
- (2) J1Flag

パラメーター

- ポイント指定** P 番号, P(式), ポイントラベルのいずれかを指定します。
- 設定値** 下記のいずれかを、整数または式で指定します。省略可能です。
- RS シリーズの場合
- 0 (/J1F0) J1 の範囲: -90 ~ +270 (単位: 度)
 - 1 (/J1F1) J1 の範囲: -270 ~ -90 または、+270 ~ +450 (単位: 度)
- C8, C12 シリーズの場合
- 0 (/J1F0) J1 の範囲: -180 ~ +180 (単位: 度)
 - 1 (/J1F1) J1 の範囲: -240 ~ -180 または、+180 ~ +240 (単位: 度)

結果

J1Flag 属性は、1 つのポイントに対する第 1 関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対する J1Flag 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J1Flag 値が表示されます。

参照

Hand, J1Flag 関数, J2Flag

J1Flag 使用例

```
J1Flag P0, 1
J1Flag P(my point), 0
```

J1Flag 関数

ポイントの J1Flag 属性を返します。

書式

J1Flag [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。

省略可能で、省略するとロボットの現在位置の J1Flag 設定を返します。

戻り値

0 /J1F0

1 /J1F1

参照

Hand, J1Flag, J2Flag

J1Flag 関数使用例

```
Print J1Flag(pick)
Print J1Flag(P1)
Print J1Flag
Print J1Flag(Pallet(1, 1))
```


J2Flag

ポイントの J2Flag 属性を設定します。

書式

- (1) J2Flag ポイント指定 [, 設定値]
- (2) J2Flag

パラメーター

- ポイント指定** P 番号, P(式), ポイントラベルのいずれかを指定します。
- 設定値** 下記のいずれかを、整数または式で指定します。省略可能です。
- 0 (/J2F0) J2 の範囲: -180 ~ +180 (単位: 度)
- 1 (/J2F1) J2 の範囲: -360 ~ -180 または、+180 ~ +360 (単位: 度)

結果

J2Flag 属性は、1 つのポイントに対する第 2 関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対する J2Flag 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J2Flag 値が表示されます。

参照

Hand, J1Flag, J2Flag 関数

J2Flag 使用例

```
J2Flag P0, 1
J2Flag P(mypoint), 0
```

J2Flag 関数

ポイントの J2Flag 属性を返します。

書式

J2Flag [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。

省略可能で、省略するとロボットの現在位置の J2Flag 設定を返します。

戻り値

0 /J2F0

1 /J2F1

参照

Hand, J1Flag, J2Flag

J2Flag 関数使用例

```
Print J2Flag (pick)
Print J2Flag (P1)
Print J2Flag
Print J2Flag (P1 + P2)
```

J4Angle

ポイントの J4Angle 属性を設定します。

書式

- (1) J4Angle ポイント指定 [, 設定値]
- (2) J4Angle

パラメーター

- ポイント指定 P 番号, P(式), ポイントラベルのいずれかを指定します。
- 設定値 設定値を実数値で指定します。省略可能です。

結果

J4Angle 属性は、N シリーズのロボットでのみ使われます。

第 4 関節と第 6 関節の軸心が直線状にある場合の、第 4 関節角度値を指定します。

特異姿勢にならないポイントでは、J4Angle 属性の値は意味を持ちません。

設定値パラメーターを省略すると、指定ポイントに対する J4Angle 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J4Angle 値が表示されます。

参照

Hand, J1Angle, J1Angle 関数, J4Angle 関数

注意

下記のように J4Flag と J4Angle の両方を使用した場合、J4Angle が優先されます

J4Angle P0,0
J4Flag P0,1

J4Angle 使用例

```
J4Angle P0, 10.0  
J4Angle P(my point), 0.0
```

J4Angle 関数

ポイントの J4Angle 属性を返します。

書式

J4Angle [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。

省略可能で、省略するとロボットの現在位置の J4Angle 設定を返します。

戻り値

第 4 関節と第 6 関節の軸心が直線上にある場合の、第 4 関節角度値を実数値で返します。

J4Angle 属性は、N シリーズのロボットでのみ使われます。

参照

Hand, J1Angle, J1Angle 関数, J4Angle

J4Angle 関数使用例

```
Print J4Angle(pick)
Print J4Angle(P1)
Print J4Angle
```

J4Flag

ポイントの J4Flag 属性を設定します。

書式

- (1) J4Flag ポイント指定 [, 設定値]
- (2) J4Flag

パラメーター

- ポイント指定** P 番号, P(式), ポイントラベルのいずれかを指定します。
- 設定値** 下記のいずれかを、整数または式で指定します。省略可能です。
- 0 (/J4F0) J4 の範囲: -180 ~ +180 (単位: 度)
- 1 (/J4F1) J4 の範囲: -360 ~ -180 または、+180 ~ +360 (単位: 度)

結果

J4Flag 属性は、1 つのポイントに対する第 4 関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対する J4Flag 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J4Flag 値が表示されます。

参照

Elbow, Hand, J4Flag 関数, J6Flag, Wrist

J4Flag 使用例

```
J4Flag P0, 1
J4Flag P(my point), 0
```

J4Flag 関数

ポイントの J4Flag 属性を返します。

書式

J4Flag [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。
省略可能で、省略するとロボットの現在位置の J4Flag 設定を返します。

戻り値

0 /J4F0
1 /J4F1

参照

Elbow, Hand, Wrist, J4Flag, J6Flag

J4Flag 関数使用例

```
Print J4Flag(pick)
Print J4Flag(P1)
Print J4Flag
Print J4Flag(Pallet(1, 1))
```

J6Flag

ポイントの J6Flag 属性を設定します。

書式

- (1) J6Flag ポイント指定 [, 設定値]
- (2) J6Flag

パラメーター

ポイント指定 P 番号, P(式), ポイントラベルのいずれかを指定します。

設定値 整数または式を指定します。

範囲: 0 から 127 (/J6F0 ~ /J6F127)

指定ポイントに対する J6 の範囲は、次のとおりです。

$(-180 * (\text{設定値} + 1) < J6 \leq -180 * \text{設定値})$

$(180 * \text{設定値} < J6 \leq 180 * (\text{設定値} + 1))$

結果

J6Flag 属性は、1 つのポイントに対する第 6 関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対する J6Flag 値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置の J6Flag 値が表示されます。

参照

Elbow, Hand, J4Flag, J6Flag 関数, Wrist

注意

J6Flag の範囲は機種によって異なります

C4 : 0 から 127 (/J6F0 ~ /J6F127)

C8, C12 : 0 から 81 (/J6F0 ~ /J6F81)

N2 : 0 から 40 (/J6F0 ~ /J6F40)

N6 : 0 から 61 (/J6F0 ~ /J6F61)

J6Flag 使用例

```
J6Flag P0, 1
```

```
J6Flag P(my point), 0
```

J6Flag 関数

ポイントの J6Flag 属性を返します。

書式

J6Flag [(ポイント指定)]

パラメーター

ポイント指定 ポイント式で指定します。

省略可能で、省略するとロボットの現在位置の J6Flag 設定を返します。

戻り値

0 ~ 127 /J6F0 ~ /J6F127

参照

Elbow, Hand, Wrist, J4Flag, J6Flag

J6Flag 関数使用例

```
Print J6Flag (pick)
Print J6Flag (P1)
Print J6Flag
Print J6Flag (P1 + P2)
```


JA 関数

関節の角度から計算したロボット座標を返します。

書式

JA(第 1 関節位置, 第 2 関節位置, 第 3 関節位置, 第 4 関節位置[, 第 5 関節位置, 第 6 関節位置] [, 第 7 関節位置] [, 第 8 関節位置, 第 9 関節位置])

パラメーター

第 1 関節位置~第 9 関節位置

関節の角度 (単位: deg) を実数値で指定します。直動関節の場合は (単位: mm) で指定します。

第 5 関節位置, 第 6 関節位置は、垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボット専用のパラメーターです。

第 7 関節位置は、ジョイント型 7 軸ロボット専用のパラメーターです。

第 8 関節位置, 第 9 関節位置は、付加軸 S、および T 関節専用のパラメーターです。

補足

動作範囲外の角度を指定した場合、動作範囲外エラーが発生します。

結果

指定した関節位置で示すロボット座標を返します。

解説

関節角度でロボット座標を指定する場合に、JA 関数を使用します。

JA 関数により返されるポイントがロボットの特異姿勢となる場合、そのポイントに対する動作命令を実行しても、JA 関数の引数として与えた関節角度に一致するとは限りません。JA 関数で指定する関節角度に動作したい場合は、ロボットの特異姿勢を避ける必要があります。

例)

```
> go ja(0,0,0,90,0,-90)
> where
WORLD: X:    0.000 mm Y:  655.000 mm Z:  675.000 mm U:    0.000 deg
V: -90.000 deg W: -90.000 deg
JOINT: 1:    0.000 deg 2:    0.000 deg 3:    0.000 deg 4:    0.000 deg
5:    0.000 deg 6:    0.000 deg
PULSE: 1:    0 pls 2:    0 pls 3:    0 pls 4:    0 pls
5:    0 pls 6:    0 pls
> go ja(0,0,0,90,0.001,-90)
> where
WORLD: X:   -0.004 mm Y:  655.000 mm Z:  675.000 mm U:    0.000 deg
V: -90.000 deg W: -89.999 deg
JOINT: 1:    0.000 deg 2:    0.000 deg 3:    0.000 deg 4:   90.000 deg
5:    0.001 deg 6: -90.000 deg
PULSE: 1:    0 pls 2:    0 pls 3:    0 pls 4: 2621440 pls
5:    29 pls 6: -1638400 pls
```

参照

AglToPls, XY

JA 関数使用例

```
P10 = JA(60, 30, -50, 45)
GO JA(135, 90, -50, 90)
P3 = JA(0, 0, 0, 0, 0, 0)
```

Joint

現在のロボットの位置を、関節の座標系で表示します。

書式

Joint

参照

Pulse, Where

Joint 使用例

```
>joint  
JOINT: 1: -6.905 deg 2: 23.437 deg 3: -1.999 mm 4: -16.529 deg  
>
```

JointAccuracy

関節精度補正の補正值の設定と表示をします。

書式

- (1) JointAccuracy 関節番号, 第 1 設定値, 第 2 設定値, 第 3 設定値, 第 4 設定値
- (2) JointAccuracy 関節番号

パラメーター

関節番号	関節番号を指定します。
第 1 設定値	第 1 設定値 (整数)を、数値で指定します。範囲は 0 から 2000 までです。
第 2 設定値	第 2 設定値 (整数)を、数値で指定します。範囲は 0 から 999 までです。
第 3 設定値	第 3 設定値 (整数)を、数値で指定します。範囲は 0 から 2000 までです。
第 4 設定値	第 4 設定値 (整数)を、数値で指定します。範囲は 0 から 999 までです。

マニピュレーターごとに関節精度補正が有効な関節は異なり、有効な関節以外ではエラーになります。有効な関節については、マニピュレーターマニュアルを参照してください。

戻り値

- (2)の場合は、関節番号に対応する現在の関節精度補正の補正值を表示します。

解説

JointAccuracy は、指定した関節の補正值を設定します。適切に補正值を設定することにより、軌跡精度をあげることができます。

注意

JointAccuracy は、必要な時以外、絶対に変更しないでください

JointAccuracy は、工場出荷時に精密に設定されています。不必要にこの値を変更すると軌跡精度の悪化を引き起こします。JointAccuracy はキャリブレーションウィザードを実行した場合に自動的に設定されるため、必要な時以外は JointAccuracy を絶対に変更しないようにしてください。

Calib と Hofs の実行

JointAccuracy が設定された状態で Calib, Hofs コマンドを実行すると、変更があった関節について関節精度補正の補正值が “0” になります。JointAccuracy の補正值を変更せずに、Hofs の値を変更するには、HofsJointAccuracy を実行してください。

参照

HofsJointAccuracy, Calib, Hofs

JointAccuracy 使用例

以下は、コマンドウィンドウでの簡単な使用例です。第 1 関節に対して、第 1 設定値 “1000”、第 2 設定値 “420”、第 3 設定値 “100”、第 4 設定値 “240”として関節精度補正の補正値を設定しています。その後、現在の第 1 関節の関節精度補正の補正値を表示します。

```
> JointAccuracy 1, 1000, 420, 100, 240
```

```
> JointAccuracy 1  
1000, 420, 100, 240  
>
```

JointAccuracy 関数

関節精度補正の補正值の表示をします。

書式

JointAccuracy(関節番号, 設定番号)

パラメーター

関節番号 関節番号を指定します。

設定番号 表示する補正值を以下の定数または整数値(1～4)で指定します。

定数	値
JAC_PARAM1	1: 第 1 設定値
JAC_PARAM2	2: 第 2 設定値
JAC_PARAM3	3: 第 3 設定値
JAC_PARAM4	4: 第 4 設定値

戻り値

指定した関節の設定番号に対応する関節精度補正の補正值(整数値)を返します。

参照

JointAccuracy

JointAccuracy 関数使用例

下記は JointAccuracy 関数を使ったプログラムの例です。

```
Function DisplayJointAccuracy(joint As Integer)
    Integer i

    Print "Joint ", joint, ", JointAccuracy settings:"
    For i = 1 To 4
        Print "Param ", i, " = ", JointAccuracy(joint, i)
    Next i
End
```

JRange

指定関節の許容動作エリアをパルス値で設定します。

書式

JRange 関節番号, 下限パルス値, 上限パルス値

パラメーター

関節番号	JRange が指定する関節番号を 1~9 の整数値で設定します。 付加軸の S 軸は 8、T 軸は 9 になります。
下限パルス値	指定関節の動作範囲の下限パルス値を、式または数値で指定します。
上限パルス値	指定関節の動作範囲の上限パルス値を、式または数値で指定します。

解説

指定関節の動作範囲を上限パルス値と下限パルス値で設定します。**Range** コマンドでは全関節の動作範囲の指定が必要ですが、**JRange** コマンドでは、1 関節ごとに動作エリアを設定するので、パラメーターの数が少なくてすみます。設定した動作エリアを確認するには、**Range** コマンドを使います。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

下限パルス値 ≤ 上限パルス値を守ってください

JRange コマンドの下限パルス値は、上限パルス値を上回らないように設定してください。下限パルス値が上限パルス値を上回るとエラーになり、動作コマンドの実行ができなくなります。

Jrange の設定値の変更

JRange で設定された値は、一度設定されると、**Range** または **JRange** コマンドで変更しない限り保持されます。電源オフでは、JRange で設定した制限値は変更されません。

動作エリアの上下限

各マニピュレーターの型によって動作エリアの上限パルス値は異なりますので、エリアの上限パルス値の設定については、各マニピュレーターマニュアルを参照してください。

参照

Range, JRange 関数

JRange 使用例

下記は、コマンドウィンドウからの操作例です。

- > **JRange** 2, -6000, 7000 ' 第 2 関節の動作範囲を定義
- > **JRange** 1, 0, 7000 ' 第 1 関節の動作範囲を定義

JRange 関数

指定した関節の許容動作エリアパルス値 (レンジ設定値)を返す関数です。

書式

JRange(関節番号, パラメーター番号)

パラメーター

関節番号 参照する関節番号 (1~9 の整数)を、式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

パラメーター番号 下記の 2 つの値のいずれかを、整数値で指定します。

1: 下限パルス値を指定

2: 上限パルス値を指定

戻り値

指定関節のレンジ設定値 (整数値、単位: パルス)を返します。

参照

Range, JRange

JRange 関数使用例

```
Long i, oldRanges(3, 1)

For i = 0 To 3
    oldRanges(i, 0) = JRange(i + 1, 1)
    oldRanges(i, 1) = JRange(i + 1, 2)
Next i
```


JS 関数

Jump Sense, Jump3 Sense, Jump3CP Sense, JumpTLZ Sense 実行後、Sense 入力が成立したかどうかを返す関数です。

書式

JS

戻り値

“True”または“False”を返します。

True : Sense 入力条件が成立してアームが目標座標上方で停止した場合、JS は“True”を返します。

False : アームが Jump 命令で設定された目標位置に到達し正常な動作を完了すると、JS は、“False”を返します。

解説

JS は、Jump や Sense 命令とセットで用います。JS 命令の目的は、Jump 命令が指示した動作中に (Sense で設定した)入力条件が成立したかどうかを返します。入力条件が成立すると JS は“True”を返します。入力条件が成立せず、アームが目標位置に到達すると JS は、“False”を返します。

JS は、単に状態を確認する命令で、動作を発生させたり動作中に何かの入力をチェックしたりする機能はありません。動作を指示するには Jump 命令を用い、場合によって、その Jump 命令によって起こる動作中に特定の入力をチェックするために、Sense 命令を用います。

注意

JS は直前に使用した Jump 命令, Jump3 命令, Jump3CP, JumpTLZ 命令とのみ連携します

JS は、直前に使用した Jump 命令の入力チェック (Sense 命令によって指示されます)のみを行います。次の Jump 命令が起動すると、JS 命令は、その新しい Jump 命令についての状態を返します。最初の Jump 命令に対する状態データは、返すことはできません。状態チェックが必要な Jump 命令には、必ずその直後に JS チェックを行うようにします。

参照

JT, Jump, Jump3, Jump3CP, JumpTLZ, Sense

JS 関数使用例

```
Function SearchSensor As Boolean
    Sense Sw(5) = On

    Jump P0
    Jump P1 Sense
    If JS = TRUE Then
        Print "Sensor was found"
        SearchSensor = TRUE
    EndIf
End
```

JT 関数

直前の Jump, Jump3, Jump3CP, JumpTLZ 動作の結果を返す関数です。

書式

JT

戻り値

下記の形式の Long 型数値のビットをセット、またはクリアします。

Bit 0	上昇動作開始時、または上昇動作量 0 の場合は 1
Bit 1	水平動作開始時、または水平移動量 0 の場合は 1
Bit 2	下降動作開始時、または下降動作量 0 の場合は 1
Bit 16	上昇動作完了時、または上昇動作量 0 の場合は 1
Bit 17	水平動作完了時、または水平動作量 0 の場合は 1
Bit 18	下降動作完了時、または下降動作量 0 の場合は 1

解説

直前に実行された Jump 命令が、Sense, Till, Abort などにより指定された動作停止条件を成立したかどうかを調べることができます。

参照

JS, Jump, Jump3, Jump3CP, JumpTLZ, Sense, Till

JT 関数使用例

```
Function SearchTill As Boolean

    Till Sw(5) = On

    Jump P0
    Jump P1 Till
    If JT And 4 Then
        Print "Motion stopped during descent"
        SearchTill = TRUE
    EndIf
End
```

JTran

原点復帰を必要としない 1 関節のみの PTP 動作を行います。

書式

JTran 関節番号, 移動量

パラメーター

関節番号	動作する関節番号を整数値で指定します。 付加軸の S 軸は 8、T 軸は 9 になります。
移動量	実数を指定します。回転関節では、(単位: deg)、直動関節では、(単位: mm)で指定します。

解説

JTran は、現在位置から、指定した関節のみを指定量動作させます。

参照

Go, Jump, Move, PTran

JTran 使用例

```
JTran 1, 20
```

Jump

アームを、現在位置から指定位置まで、ゲートモーション (まず垂直方向に上昇し、それから水平移動し、最後に垂直方向に下降する、門型動作)により PTP 動作します。

書式

Jump 目標座標 [C アーチ番号] [LimZ [Z 座標値]] [CP]
[Sense|Till|Find]] [!並列処理] [SYNC]

パラメーター

目標座標	ポイントデータで目標位置を指定します。
アーチ番号	アーチ番号は、Jump におけるアーチモーションの形状に、どの Arch テーブルの設定を使うか指定します。アーチ番号は、頭にアルファベット文字 “C” をつけてください。有効値は、C0~C7 です。省略可能です。
Z 座標値	Jump 命令による動作中に第 3 関節が移動し得る最大値 (Z 制限値)、あるいは、Jump による Z 成分の高さシーリング値と考えてください。有効な第 3 関節の座標値ならば、どの値でもかまいません。省略可能です。
CP	パスモーションを指定します。省略可能です。
Sense、Till または Find	Sense、Till または Find 式を記述します。省略可能です。 Sense Till Find Sense Sw(式) = {On Off} Till Sw(式) = {On Off} Find Sw(式) = {On Off}
!並列処理!	Jump 命令による動作中に、I/O や、その他のコマンドを実行するための並列処理ステートメントを追加することができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

Jump 命令は、いわゆる「アーチモーション (アーチ型動作)」によってアームを現在位置から目標座標まで移動させます。いわば、3 つの動きを 1 回で行うステートメントと考えることができます。例えば、アーチ番号が定義されていれば、1 回の Jump 命令で次の 3 つの動作を行います。

- 1) まず、第 3 関節だけが、Jump コマンド中のアーチ番号が演算する Z 軸の高さまで動作します。
- 2) 次に、アームは、LimZ が指定する Z 制限位置に達するまで Z 軸方向に上昇しながら、目標座標に水平移動します。それから、第 1 関節、第 2 関節、第 4 関節のそれぞれの動作を行いながら Z 軸方向に下降を始めます。アームは最終的な X、Y、U 座標位置が得られるまで動作します。
- 3) アームは、ターゲットの Z 座標位置が得られるまで Z 軸方向のみの移動を行い、目標座標が得られたところで Jump 命令が終了します。

目標座標 (移動の目的位置)は、Jump 命令の中で指定することができないので、Jump 命令が実行される前にティーチングしておく必要があります。Jump 移動の加速と減速は、Accel によって行います。また、移動の速度は、Speed によって行います。

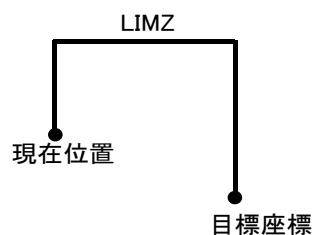
垂直 6 軸型ロボット (N シリーズを含む)に対して Jump を実行することはできません。Jump3 を使用してください。

CP について

CP パラメーターをつけた場合には、動作の減速開始時に次の動作命令の加速を重ね合わせることが可能です。この場合には、目標座標には位置決めされません。

アーチ番号について

Jump のアーチの型は、Jump コマンド中で指定されるアーチ番号によって変更することができます。これによって、第 1 関節、第 2 関節、第 4 関節の各関節が動く前に、どれだけ Z 軸方向に動かしておきたいか決めることができます。Jump 命令で有効に使えるアーチ番号値は、C0~C7 までの値です。C0~C6 までの値に対する Arch テーブル値の設定は、ユーザーが Arch コマンドによって定義することができるようになっています。しかし、C7 は常に、「ゲートモーション」を定義しています。「ゲートモーション」とは、ロボットが、第 1 関節、第 2 関節、第 4 関節の各関節を動かす前に、第 3 関節だけを先に、LimZ で定義される座標位置まで移動させる動きのことです。この「ゲートモーション」では、LimZ が定義する Z 制限値まで移動してから初めて、第 1 関節、第 2 関節、第 4 関節の各関節の動作が始まります。第 1 関節、第 2 関節、第 4 関節の各関節が、それぞれの最終目標座標位置まで移動すると、第 3 関節が、目標座標で定義される最終 Z 座標位置を目指して下降します。次の図は「ゲートモーション」の動きを表わします。



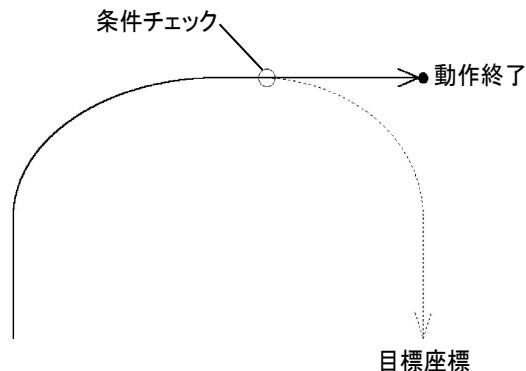
LimZ について

LimZ Z 座標値は、現在設定されているローカル座標系の水平移動面における Z 座標の最高値を指定します。指定されたアーチ設定によっては、LimZ 値に達する前に、第 1 関節、第 2 関節、第 4 関節の各関節が動き始めることもあるかも知れませんが、LimZ 値は、常に、その移動における Z 座標方向の上限値を定義しています。もし、LimZ パラメーターを省略すると、その直前 (最後) に LimZ によって指定された最高値が準用されます。

LimZ によって指定される高さ方向の制限値は、ローカルロボット座標系での Z 座標値です。つまり、それは、Arm や Tool 座標の Z 座標値ではありません。したがって、作業高の異なるツールやハンドを使用する場合は、十分注意し、必要な処置を行ってください。

Sense について

省略できるパラメーターの 1 つです。Sense は、第 3 関節が最終の下降動作を行う前に、入力条件またはメモリ I/O 条件をチェックしたいときなどに使います。問題がなければ、ロボットのアームは、目標座標の上(最終的に第 3 関節の動作を残すだけになっている位置)に停止して、このコマンド (Jump) の実行が終了したことになります。ただし、Sense で指定した条件を検知しても、ロボットのアームは、すぐには停止しないので注意してください。



JS や Stat などのコマンドを組み合わせ、Sense 条件が成立してロボットのアームが目標座標の前で停止したか、あるいは、Sense 条件が成立せずロボットのアームはそのまま目標座標に停止したか、を確認することができます。

Till について

オプションの Till を使って、Jump を実行する前に、ロボットを減速して停止させる条件を設定することができます。これは、I/O 入力のうち 1 つ、あるいは、メモリ I/O の 1 つをチェックする、という条件を含むことで設定できます。これには、Sw または MemSw 関数を使います。あらかじめ設定された条件にしたがって、入力がオンかオフかをチェックして、アームを減速し、停止させることができます。

Stat 関数を使うことにより、Till 条件が成立してこのコマンドが実行されたか、あるいは、Till 条件が成立しないでロボットが目標座標に停止したかを確認することができます。

注意

Jump は垂直 6 軸型ロボット (N シリーズを含む)には使用できません。

Jump3 または Jump3CP を使用してください。

アーチ番号 パラメーターを省略する場合

アーチ番号パラメーターの設定を省略すると、Jump コマンド実行時のデフォルト Arch 値は、C7 となります。上記でも説明されているとおり、Arch 値が C7 の場合、「ゲートモーション」移動(上記参照)になります。

Jump と Jump3, Jump3CP の違い

Jump3 と Jump3CP は、垂直 6 軸型ロボット (N シリーズを含む)で使用することができますが、Jump は使用することができません。水平多関節型ロボット (RS シリーズを含む)で、Z 軸方向に上昇と下降を行う場合は、Jump を使用すると動作時間が短縮できます。Jump3 の接近動作や退避動作は、Z 軸以外の方向に行うこともできます。

Jump と Go の違い

Jump と Go の、最も重要な違いは、Go の場合、全関節の動きがシンクロしており、各関節は同時に動き始め、同時に停止します。Jump では、動作の始めと終わりに垂直方向だけの第 3 関節移動があります。デバイスを吸着、配置というような作業にはこのコマンドをお勧めします。

Jump における減速停止

Jump では、アームは必ず減速しながら目標座標に停止します。

Jump における適正速度と加速指示

Jump 動作におけるロボットの速度と加減速は、それぞれ Speed と Accel よって設定します。Speed と Accel は、Jump, Go など、あくまでもポイントからポイントへ動作する場合についてのみ設定できる命令で

あることに注意してください。例えば、Move や Arc のような直線および円弧補間動作を行う命令の場合は、SpeedS や AccelS の命令を使用してください。

また、Jump の場合は、第 3 関節の上昇移動、第 4 関節の回転を含む水平動作、そして第 3 関節の下降について、速度と加減速を別々に設定することができます。

Jump の Pass 機能

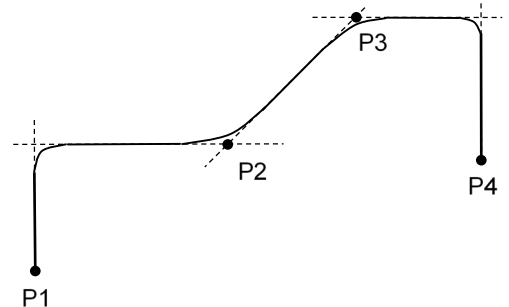
下降動作量が 0 の Jump に CP パラメーターをつけた場合、その Jump の水平動作が減速停止することなく、直後に続く PTP 動作と滑らかに接続できます。

また、直前の PTP 動作命令に CP パラメーターをつけた場合、上昇動作量が 0 の Jump は、その PTP 動作が減速停止することなく、Jump の水平動作と滑らかに接続できます。

これは、通常の Jump の水平動作 (1 つの PTP 動作) を、いくつかの PTP 動作が滑らかに接続する動作に置きかえたい場合に便利です。

(例)

```
Go P1
Jump P2 :Z(-50) C0 LimZ -50 CP
Go P3 :Z(0) CP
Jump P4 C0 LimZ 0
```



アーチ使用時の重要事項

アーチモーションは、第 3 関節の上昇または下降動作と、横方向動作の合成を軌跡制御上で行うため、実際の軌跡を保証できるものではありません。動作速度やアームの動き方によってその軌跡は変化します。実際の軌跡は、作業で使用する実速度と姿勢で確認してください。

- 同じ位置で、同じ[C アーチ番号]を持つ Jump 命令を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- 低速動作時より高速動作の方が、垂直上昇量が増え、垂直下降量が減る傾向にあります。期待した垂直下降距離が出ない場合は、速度や減速度を下げるか、下降距離を長めに設定してください。
- 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。アームの動き方による軌跡の変化はさまざまですが、一般的な水平多関節型ロボットを例にとれば、第 1 アームを大きく動かすほど垂直上昇量が増え、垂直下降量が減る傾向が強くなります。期待した垂直下降距離が出ない場合は、速度や減速度を下げるか、下降距離を長めに設定してください。

起こりやすいエラー

LimZ 値の設定が低すぎる場合

第 3 関節のアーム位置が、LimZ の設定値よりも高い位置にある状態で、Jump が実行されると、エラー 4005 になります。

参照

Accel, Arc, Arch, Go, JS, JT, LimZ, P# = ポイント指定, Pulse, Sense, Speed, Stat, Till

Jump 使用例

次の例は、ポイント P0 から P1 へ、単純な PTP 動作をしたあと、P0 へ **Jump** で戻るものです。プログラム後半で、アームは **Jump** を実行し、入力ビット 4 がオンになれば下降動作を行い、P1 に移動します。入力ビット 4 がオンの場合、下降動作を行いません。

```
Function jumptest
  Home
  Go P0
  Go P1
  Sense Sw(4) = On
  Jump P0 LimZ -10
  Jump P1 LimZ -10 Sense '入力 4 をチェック
  If Js(0) = 1 Then
    Print "Input #4 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P1."
  Else
    Print "The move to P1 completed successfully."
    Print "Input #4 never came on during the move."
  EndIf
Fend
```

> **Jump** P10+X50 C0 LimZ-20 Sense !D50;On 0;D80;On 1!

コマンドウィンドウからの操作例です。

> Jump P0	'P0 にジャンプ動作します
> Jump P0 C0	'P0 にアーチ番号 C0 で設定されたアーチモーションでジャンプ動作します
> Jump P0 LimZ -10	'P0 に、Z 制限値-10mm まででジャンプ動作します
> Jump P0 !D0; On 1; D50; Off 1!	'P0 にジャンプ動作します。動作の移動量が 50%になるまでは出力の 1 ビット目を On し、50%以降は 1 ビット目を Off します。

入力の 1 ビット目が On になったら、**Jump** 命令を停止して次の処理に進みます。

```
Function main
  (中略)
  Till Sw(1) = On
  Jump P0 C0 CP Till
  (中略)
Fend
```


Jump3, Jump3CP

アームを3次元ゲート動作で移動します。

Jump3 は、2つの CP 動作と1つの PTP 動作の組み合わせです。

Jump3CP は、3つの CP 動作の組み合わせです。

書式

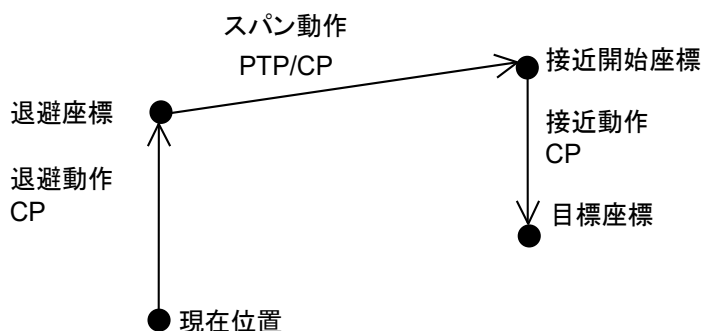
- (1) Jump3 退避座標, 接近開始座標, 目標座標 [, C アーチ番号] [, CP] [, LJM [, 姿勢フラグ選択]]
[, Sense | Till | Find] [, !並列処理!] [, SYNC]
- (2) Jump3CP 退避座標, 接近開始座標, 目標座標 [, ROT] [, C アーチ番号] [, CP] [, LJM [, 姿勢フラグ選択]] [, Sense | Till | Find] [, !並列処理!] [, SYNC]

パラメーター

退避座標	現在位置より上の退避点をポイント指定します。
接近開始座標	目標座標より上の接近開始点をポイント指定します。
目標座標	動作の到達する目標座標をポイント指定します。
ROT	ツール姿勢変化を優先させて、動作の速度、加減速度を決めます。省略可能です。
アーチ番号	アーチ番号は、Jump3 命令のアーチ型動作を決める、アーチテーブルを指定します。アーチ番号の頭には、必ず大文字「C」をつけてください。(有効値は C0~C7 です。) アーチ番号は省略可能です。
CP	パスモーションを指定します。省略可能です。
LJM	待避座標, 接近座標, 目標座標を LJM 関数で変換します。省略可能です。
姿勢フラグ選択	LJM 関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。
Sense Till Find	Sense, Till, Find のいずれかの式を記述します。省略可能です。 Sense Till Find Sense Sw(式) = {On Off} Till Sw(式) = {On Off} Find Sw(式) = {On Off}
!並列処理!	Jump3、Jump3CP 命令に並列処理ステートメントを追加し、動作中に I/O やその他のコマンドを実行できます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

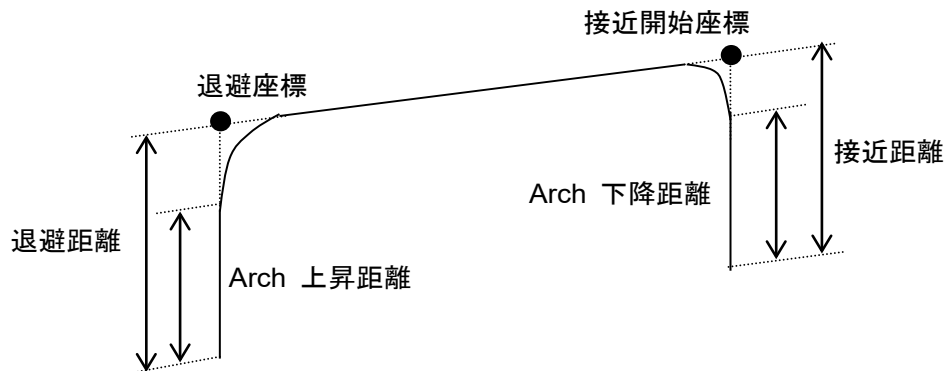
アームを現在位置から目標座標まで3次元ゲート動作で移動します。3次元ゲート動作は、退避動作、スパン動作、そして接近動作から成り立っています。現在位置から退避座標までの退避動作は CP 動作です。退避座標から接近開始座標までのスパン動作は、Jump3 の場合は PTP 動作で、Jump3CP の場合は CP 動作です。接近開始座標から目標座標までの接近動作は、CP 動作になります。



アーチ動作は、アーチ番号の設定によって行います。

Jump3 および Jump3CP のアーチ動作は下図に示すとおりです。

退避距離は Arch 上昇距離よりも長く、接近距離は Arch 下降距離よりも長くしてください。



Jump3CP の速度と加減速度はそれぞれ SpeedS と AccelS の設定値を使用します。速度と加減速度の関係については、“注意”の“Jump3, Jump3CP を CP とともに用いる”を参照してください。ただし、ROT 修飾パラメーターを指定しているときの速度と加減速度はそれぞれ SpeedR と AccelR の設定値を使用します。この場合、SpeedS と AccelS の設定値は無効となります。

通常、移動距離が 0 で、姿勢関節のみが動作するような場合にはエラーとなります。ROT 修飾パラメーターを付けてツール姿勢変化の加減速を優先することでエラーなく動作が可能になります。ROT 修飾パラメーターを付けたとき、姿勢が変化なく、移動距離が 0 でない場合にはエラーとなります。

また、移動距離に対してツール姿勢変化速度が大きすぎる場合や、指定された回転速度がマニピュレーターの限度を越えている場合もエラーとなります。その場合は指定速度を落とすか、または ROT 修飾パラメーターをつけて姿勢変化の加減速度を優先するようにしてください。

注意

LimZ は、Jump3 および Jump3CP には影響しません。

スパン動作は、必ずしも座標系の Z 軸に垂直とは限らないため、LimZ は、Jump3 および Jump3CP には影響しません。

Jump3 のスパン動作は PTP 動作です。

PTP 動作はその軌跡が予測しにくいいため、ロボット本体や周辺装置との干渉に十分注意してください。

Jump3, Jump3CP を CP とともに用いる

CP パラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。CP 指定なしの Jump3 命令, Jump3CP 命令では、アームは必ず減速して、指定された目標座標に停止します。

Jump3 の Pass 機能

接近動作量が 0 の Jump3 に CP パラメーターをつけた場合、その Jump3 のスパン動作が減速停止することなく、直後に続く PTP 動作と滑らかに接続できます。

また、直前の PTP 動作命令に CP パラメーターをつけた場合、退避動作量が 0 の Jump3 は、その PTP 動作が減速停止することなく、Jump3 のスパン動作と滑らかに接続できます。

これは、通常の Jump3 のスパン動作 (1 つの PTP 動作) を、いくつかの PTP 動作が滑らかに接続する動作に置きかえたい場合に便利です。

Jump3CP の Pass 機能

接近動作量が 0 の Jump3CP に CP パラ미터をつけた場合、その Jump3CP のスパン動作が減速停止することなく、直後に続く CP 動作と滑らかに接続できます。

また、直前の CP 動作命令に CP パラ미터をつけた場合、退避動作量が 0 の Jump3CP は、その CP 動作が減速停止することなく、Jump3CP のスパン動作と滑らかに接続できます。

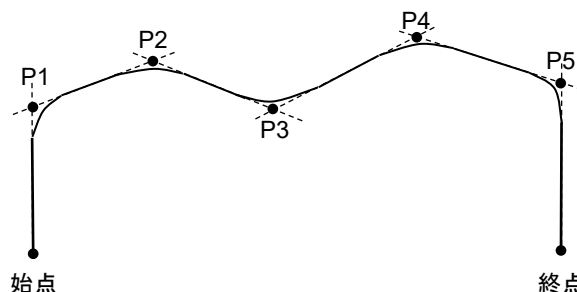
これは、通常の Jump3CP のスパン動作 (1 つの CP 動作) を、いくつかの CP 動作が滑らかに接続する動作に置きかえたい場合に便利です。

(例 1)

```
Jump3 P1, P2, P2 CP
Go P3, P4 CP
Jump3 P4, P5, P5+t1z (50)
```

(例 2)

```
Jump3CP P1, P2, P2 CP
Move P3, P4 CP
Jump3CP P4, P5, P5+t1z (50)
```



Jump3, Jump3CP を LJM とともに用いる

LJM パラ미터を使うと、LJM 関数を使用するプログラムを簡略化できます。

例えば、

```
P11 = LJM(P1, Here, 2)
P12 = LJM(P2, P11, 2)
P13 = LJM(P3, P12, 2)
Jump3 P11, P12, P13
```

という 4 行のプログラムを

```
Jump3 P1, P2, P3 LJM 2
```

という 1 行のプログラムに置き換えることができます。

LJM パラ미터は垂直 6 軸型ロボット (N シリーズを含む)、および RS シリーズロボットで有効です。

Jump3CP はスパン動作が直線 (CP) 動作のため途中で手首姿勢を切り替えることができません。そのため手首姿勢の切り替えの可能性がある LJM 関数の姿勢フラグ選択 (LJM 1) は使用しないでください。

アーチ使用時の重要事項

アーチモーションは、動作の合成を軌跡制御上で行うため、実際の軌跡を保証できるものではありません。動作速度やアームの動き方によってその軌跡は変化します。実際の軌跡は、作業で使用する実速度と姿勢で確認してください。

- 同じ位置で、同じ[C アーチ番号]を持つ Jump3 命令を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- 低速動作時より高速動作の方が、合成なしの退避移動量が増え、合成なしの接近移動量が減る傾向にあります。期待した移動距離が出ない場合は、速度や減速度を下げるか、接近距離を長めに設定してください。
- 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。

起こりやすいエラー

退避動作 (接近動作)とスパン動作で、主に動作する関節が同じ場合

Jump3, Jump3CP コマンドでアーチモーションを実行中に、異常加速度エラーが発生することがあります。これは、退避動作 (接近動作)とスパン動作で、主に動作する関節が同じ場合に特に顕著となります。このような場合は、Jump3 の場合は Accel 命令で、Jump3CP の場合は AccelS 命令でスパン動作の加減速度を下げることで回避してください。また、動作姿勢によっては、AccelS 命令で退避動作 (接近動作)の加減速度を下げることで有効な場合もあります。

参照

Accel, Arc, Arch, Go, JS, JT, P# = ポイント指定, Pulse, Sense, Speed, Stat, Till

Jump3 使用例

' スカラロボットの Jump のように動作する垂直 6 軸型ロボット (N シリーズを含む)の動作
Jump3 Here :Z(100), P3 :Z(100), P3

' Z ツール座標を使用した退避動作と接近動作
Jump3 Here -TLZ(100), P3 -TLZ(100), P3

' Z ベース座標を使用した退避動作と Z ツール座標を使用した接近動作
Jump3 Here +Z(100), P3 -TLZ(100), P3

Tool1 で退避動作を行い、Tool2 で接近動作を行う例

```
Arch 0,20,20
Tool 1
Go P1

P2 = P1 -TLZ(100)
Tool 2
Jump3 P2, P3-TLZ(100), P3 C0
```

JumpTLZ

アームを3次元ゲート動作で移動します。

JumpTLZは、2つのCP動作と1つのPTP動作の組み合わせです。

書式

JumpTLZ 目標座標, TLZ 方向移動量[, C アーチ番号] [, CP] [, LJM [, 姿勢フラグ選択]] [, Sense | Till | Find] [, !並列処理] [, SYNC]

パラメーター

目標座標	動作の到達する目標座標をポイント指定します。
TLZ 方向移動量	Tool座標のZ方向に移動する量を指定します。単位は mm です。 現在設定されている Tool 番号の Tool 座標を使用します。
アーチ番号	アーチ番号は、JumpTLZ 命令のアーチ型動作を決める、アーチテーブルを指定します。アーチ番号の頭には、必ず大文字「C」をつけてください。(有効値はC0からC7です。) アーチ番号は省略可能です。
CP	パスモーションを指定します。省略可能です。
LJM	目標座標をLJM関数で変換します。省略可能です。
姿勢フラグ選択	LJM関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。
Sense Till Find	Sense, Till, Findのいずれかの式を記述します。省略可能です。 Sense Till Find Sense Sw(式) = {On Off} Till Sw(式) = {On Off} Find Sw(式) = {On Off}
!並列処理!	Jump3, Jump3CP 命令に並列処理ステートメントを追加し、動作中にI/Oやその他のコマンドを実行できます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

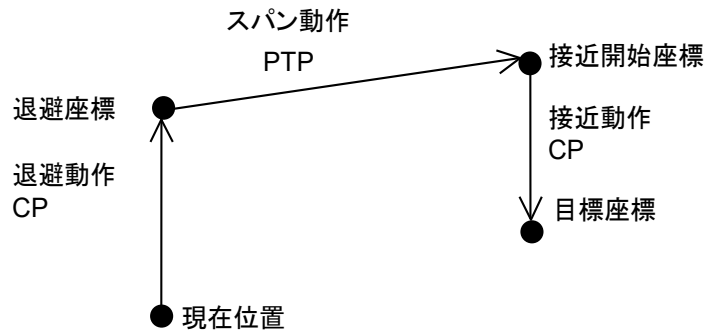
解説

アームを現在位置から目標座標まで 3 次元ゲート動作で移動します。3 次元ゲート動作は、退避動作、スパン動作、そして接近動作から成り立っています。現在位置から退避座標までの退避動作は CP 動作です。退避座標から接近開始座標までのスパン動作は PTP 動作になります。

退避座標は、現在位置から Tool 座標の Z 方向に TLZ 方向移動量だけ移動した位置になります。

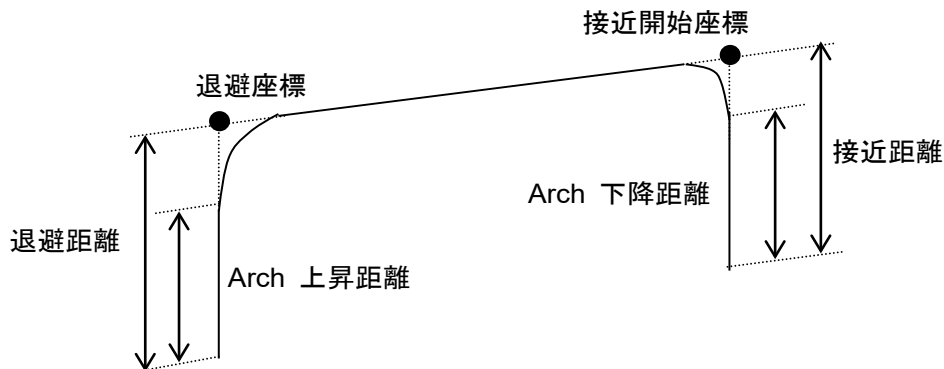
退避座標のロボットの姿勢は、現在位置と同じになります。(特異点や特異点近傍を通過した場合のみ、ロボットの姿勢が同じにならないことがあります。)

接近開始座標は、退避座標から Tool 座標の X, Y 方向に目標座標までの移動量だけ移動した位置になります。退避座標の U, V, W の座標とロボットの姿勢は、目標位置と同じになります。(特異点や特異点近傍を通過した場合のみ、ロボットの姿勢が同じにならないことがあります。)



アーチ動作は、アーチ番号の設定によって行います。

退避距離は Arch 上昇距離よりも長く、接近距離は Arch 下降距離よりも長くしてください。



注意

LimZ は、JumpTLZ には影響しません。

スパン動作は、必ずしも座標系の Z 軸に垂直とは限らないため、LimZ は、JumpTLZ には影響しません。

JumpTLZ のスパン動作は PTP 動作です。

PTP 動作はその軌跡が予測しにくいいため、ロボット本体や周辺装置との干渉に十分注意してください。

JumpTLZ と Jump3 の違い

JumpTLZ と Jump3 は下記の点で異なります。

JumpTLZ:

退避座標は、現在位置から Tool 座標の Z 方向に移動した位置以外にすることはできません。

接近座標は、目標座標から Tool 座標の Z 方向以外に移動させることはできません。

また、接近距離を指定することはできません。

退避座標、接近座標、目標位置で異なる Tool 座標を選択することはできません。

(Tool1 で退避動作を行い、Tool2 で接近動作を行うことはできません)

Jump3:

退避座標の位置を任意に指定できます
接近座標の位置を任意に指定できます
退避座標,接近座標,目標位置で異なる Tool 座標を選択することができます。
(Tool1 で退避動作を行い、Tool2 で接近動作を行うことが可能です)

JumpTLZ が使用できる機種

N シリーズのみ使用可能です。

アーチ使用時の重要事項

アーチモーションは、動作の合成を軌跡制御上で行うため、実際の軌跡を保証できるものではありません。動作速度やアームの動き方によってその軌跡は変化します。実際の軌跡は、作業で使用する実速度と姿勢で確認してください。

- ・ 同じ位置で、同じ[C アーチ番号]を持つ JumpTLZ 命令を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- ・ 低速動作時より高速動作の方が、合成なしの退避移動量が増え、合成なしの接近移動量が減る傾向にあります。期待した移動距離が出ない場合は、速度や減速度を下げるか、接近距離を長めに設定してください。
- ・ 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。

起こりやすいエラー

退避動作 (接近動作)とスパン動作で、主に動作する関節が同じ場合

JumpTLZ コマンドでアーチモーションを実行中に、異常加速度エラーが発生することがあります。これは、退避動作 (接近動作)とスパン動作で、主に動作する関節が同じ場合に特に顕著となります。このような場合は、Accel 命令でスパン動作の加減速度を下げることで回避してください。また、動作姿勢によっては、AccelS 命令で退避動作 (接近動作)の加減速度を下げることも有効な場合があります。

参照

Accel, Arc, Arch, Go, JS, JT, P# = ポイント指定, Pulse, Sense, Speed, Stat, Till

JumpTLZ 使用例

現在位置から Tool 座標の Z 方向に 100 mm 上昇し、目的地 (P0)に移動させる場合:

```
JumpTLZ P0, -100
```

LatchEnable

R-I/O 入力によるロボット位置のラッチ機能を有効、または無効にします。

書式

LatchEnable { On | Off }

パラメーター

On | Off On : ロボット位置のラッチ機能を有効にします。
 Off : ロボット位置のラッチ機能を無効にします。

結果

パラメーターが省略されているときは、現在のラッチ機能の有効、無効を表示します。

解説

R-I/O に接続したトリガー入力信号によるロボット位置のラッチ機能を有効、または無効にします。ラッチ機能を有効にしてから、最初のトリガー入力でロボットの位置がラッチされます。繰り返しロボット位置をラッチする場合は、一度 **LatchEnable Off** でラッチ機能を解除し、再び **LatchEnable On** を実行します。繰り返し使用する場合、各コマンドの処理時間を考慮すると、約 60 ms 以上の間隔が必要となります。
LatchEnable 自体の実行時間は無視できます。

注意

ラッチ機能を有効にする前に、**SetLatch** でトリガー入力ポートとトリガー信号論理の設定をしてください。

参照

LatchPos 関数, LatchState 関数, SetLatch

LatchEnable 使用例

```
Function main
    SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
    LatchEnable On           'ラッチ機能有効
    Go P1
    Wait LatchState = True   'トリガー待ち
    Print LatchPos           'ラッチ位置表示
    LatchEnable Off        'ラッチ機能無効
Fend
```


LatchState 関数

R-I/O によるロボット位置のラッチ状態を返す関数です。

書式

LatchState

戻り値

ロボット位置のラッチが完了している場合は“True”、完了していない場合は“False”を返します。

ラッチ完了を確認したあと、LatchPos 関数でラッチ位置情報を取得します。

SetLatch で連続ラッチ回数を指定した場合、指定した回数のラッチが全て完了すると“True”を返します。

参照

LatchEnable, LatchPos 関数, SetLatch, Wait

LatchState 関数使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
  LatchEnable On           'ラッチ機能有効
  Go P1
  Wait LatchState = True   'トリガー待ち
  Print LatchPos           'ラッチ位置表示
  LatchEnable Off         'ラッチ機能無効
Fend
```

LatchPos 関数

R-I/O 入力信号でラッチしたロボットの位置を返す関数です。

書式

LatchPos ([WithToolArm | WithoutToolArm], ラッチ番号)

パラメーター

WithToolArm | WithoutToolArm

関数呼び出し時に、Tool, Arm 設定に基づいた位置を返すか、Tool 0, Arm 0 の位置を返すかを指定します。パラメーターは省略可能ですが、ラッチ番号を指定する場合は省略しないでください。省略すると、WithToolArm が設定されます。

定数	値
WithToolArm	0
WithoutToolArm	1

WithToolArm

値 0 の定数です。

WithoutToolArm

関数呼び出し時の Tool, Arm 設定に基づいた位置を返します。

値 1 の定数です。

ラッチ番号

Tool, Arm 設定に関わらず、Tool 0, Arm 0 の位置を返します。

LatchEnable On 以降、何回目の R-I/O 入力信号でラッチしたポイントデータを返すかを指定します。

1, 2, 3, 4 が指定可能です。

SetLatch で連続ラッチ回数を指定することで、LatchEnable On 以降、最大 4 回までの R-I/O 入力信号によるポイントデータのラッチが可能です。

パラメーターは省略可能です。省略すると 1 回目の R-I/O 入力信号でラッチしたポイントデータを返します。

戻り値

R-I/O 入力信号でラッチしたロボットの位置をポイントデータで返します。

LatchPos 関数の実行には約 15 msec の処理時間がかかります。

パラメーターが WithToolArm の場合、関数呼び出し時の Tool, Arm 設定に基づいた位置を返します。

パラメーターが WithoutToolArm の場合、Tool, Arm 設定に関わらず、Tool 0, Arm 0 の位置を返します。

参照

LatchEnable, LatchState 関数, SetLatch

LatchPos 関数使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE, 4
  LatchEnable On           'ラッチ機能有効
  Go P1
  Wait LatchState = True   'トリガー待ち
  Print LatchPos (WithoutToolArm, 1)      'ラッチ位置 1 表示
  Print LatchPos (WithoutToolArm, 2)      'ラッチ位置 2 表示
  Print LatchPos (WithoutToolArm, 3)      'ラッチ位置 3 表示
  Print LatchPos (WithoutToolArm, 4)      'ラッチ位置 4 表示
  LatchEnable Off         'ラッチ機能無効
Fend
```

パラメーターを省略する場合の例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
  LatchEnable On           'ラッチ機能有効
  Go P1
  Wait LatchState = True   'トリガー待ち
  Print LatchPos           'ラッチ位置 1 表示
  LatchEnable Off         'ラッチ機能無効
Fend
```

LatchPos の戻り値をポイントデータに代入する例

```
P2 = LatchPos           'ラッチ位置 1 を代入。パラメーター省略

P2 = LatchPos (WithoutToolArm, 1)  'ラッチ位置 1 を代入
P3 = LatchPos (WithoutToolArm, 2)  'ラッチ位置 2 を代入
P4 = LatchPos (WithoutToolArm, 3)  'ラッチ位置 3 を代入
P5 = LatchPos (WithoutToolArm, 4)  'ラッチ位置 4 を代入
```

LCase\$ 関数

文字列を小文字にして返します。

書式

LCase\$(文字列)

パラメーター

文字列 小文字にする文字列を指定します。

戻り値

小文字に変換した文字列を返します。

参照

LTrim\$, Trim\$, RTrim\$, UCase\$

LCase\$ 関数使用例

```
str$ = "Data"  
str$ = LCase$(str$)    ' str$ = "data"
```

Left\$ 関数

文字列の左から、指定の文字列を取り出す関数です。

書式

Left\$(文字列, 文字数)

パラメーター

文字列	指定の文字列が左からコピーされる元の文字列です。
文字数	文字列の左からコピーする文字数を直接数値で指定します。

戻り値

指定した文字列の左から、指定の文字数をとって返します。

解説

Left\$は、指定した文字列の左から文字数の数だけ文字を取り出して返します。Left\$は、指定文字列内の文字数だけ文字を返すことができます。

参照

Asc, Chr\$, InStr, Len, Mid\$, Right\$, Space\$, Str\$, Val

Left\$ 関数使用例

下記は、文字列を解析するプログラムの例です。

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(DataIn$, Len(DataIn$) - pos))

End
```

コマンドウィンドウで Left\$ 命令を使う例です。

```
> Print Left$("ABCDEFGH", 2)
AB

> Print Left$("ABC", 3)
ABC
```

Len 関数

文字列の文字数を返します。

書式

Len(文字列)

パラメーター

文字列 文字列式を指定します。

戻り値

Len 命令の引数として与えられた文字列の文字数を、整数値で返します。

解説

Len は、指定した文字列の文字数を整数値 (0~255) で返します。(文字列の字数制限範囲は 0~255 です。)

参照

Asc, Chr\$, InStr, Left\$, Mid\$, Right\$, Space\$, Str\$, Val

Len 関数使用例

下記は、文字列を解析するプログラムの例です。

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(DataIn$, Len(DataIn$) - pos))

End
```

他にコマンドウィンドウで Len 命令を使う例です。

```
> ? len("ABCDEFGH")
7

> ? len("ABC")
3

> ? len("")
0
>
```

LimitTorque

ハイパワーモード時のトルク上限値を設定、および設定値を返す関数

書式

- (1) LimitTorque 全関節ハイパワートルク上限値
 (2) LimitTorque 第 1 関節ハイパワートルク上限値, 第 2 関節ハイパワートルク上限値, 第 3 関節ハイパワートルク上限値, 第 4 関節ハイパワートルク上限値
 (3) LimitTorque 第 1 関節ハイパワートルク上限値, 第 2 関節ハイパワートルク上限値, 第 3 関節ハイパワートルク上限値, 第 4 関節ハイパワートルク上限値, 第 5 関節ハイパワートルク上限値, 第 6 関節ハイパワートルク上限値
 (4) LimitTorque

パラメーター

- 全関節ハイパワートルク上限値 すべての関節のハイパワートルク上限値を、各関節の瞬時最大トルクに対する割合を表す整数値で指定します。
- 第 n 関節ハイパワートルク上限値 第 n 関節のハイパワートルク上限値を、第 n 関節の瞬時最大トルクに対する割合を表す整数値で指定します。

結果

パラメーターが省略されているときは、現在の LimitTorque 値を表示します。

解説

ハイパーモード時のトルク上限値を制限します。通常は最大トルクが設定されており、本設定値を変更する必要はありませんが、例えば、周辺機器との干渉によるロボットや装置のダメージを少なくするため、特定の動作に必要なトルク以上のトルクを発生しないように制限したい場合に有効です。上限値は、特定の動作をした時のピークトルクを PTRQ で測定し、その値に変動分 (10%程度) を考慮した余裕を加えた値とします。

本コマンドでローパワーモード時のトルク上限値を下回るハイパワートルク上限値を設定することはできません。各機種、関節に対して設定できる最小値はそれぞれ異なります。設定後に設定値を表示し、実際に設定された上限値を確認してください。

下記の場合、LimitTorque はデフォルト値に戻ります。

コントローラー起動時 Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

注意

低すぎる LimitTorque 設定

LimitTorque は、特定の動作に対して、設定された加減速度で動作するために必要なトルクの大きさにかかわらず、設定されたトルク制限値を上限値とし、トルクを制限します。その結果、特定の動作が設定した上限値より大きなトルクを必要とする場合、ロボットは適切に動作をすることができず、振動的な動作をしたり、異音が発生したり、位置偏差エラーおよびオーバーランが発生したりすることが考えられます。必ず PTRQ を計測してから、トルク制限機能を使用してください。万一、上記の状態が発生した場合は、トルクが不足しているため、トルク上限値を大きくし、正常に動作するように調整してください。

参照

LimitTorque 関数, Power, PTrq, RealTorque

LimitTorque 使用例

第 1 関節の最大トルクを 80%に制限して動作する例です。

```
Function main
  Motor On
  Power high
  Speed 100; Accel 100,100
  LimitTorque 80,100,100,100    '第 1 関節の最大トルクを 80%に制限する
  Jump P1                        'Jump 動作を実行する
Fend
```


LimitTorque 関数

LimitTorque 命令の設定値を返します。

書式

LimitTorque(関節番号)

パラメーター

関節番号	1~9 の整数で指定します。 付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

LimitTorque 命令の設定値を整数値で返します。

参照

LimitTorque

LimitTorque 関数使用例

```
Print LimitTorque(1) ' 第 1 関節の LimitTorque 値を表示
```

LimitTorqueLP

ローパワーモード時のトルク上限値を設定、および設定値を返す関数

書式

- (1) LimitTorqueLP 全関節ローパワートルク上限値
- (2) LimitTorqueLP 第 1 関節ローパワートルク上限値, 第 2 関節ローパワートルク上限値, 第 3 関節ローパワートルク上限値, 第 4 関節ローパワートルク上限値
- (3) LimitTorqueLP 第 1 関節ローパワートルク上限値, 第 2 関節ローパワートルク上限値, 第 3 関節ローパワートルク上限値, 第 4 関節ローパワートルク上限値, 第 5 関節ローパワートルク上限値, 第 6 関節ローパワートルク上限値
- (4) LimitTorqueLP

パラメーター

全関節ローパワートルク上限値	すべての関節のローパワートルク上限値を、各関節の瞬時最大トルクに対する割合を表す整数値で指定します。
第 n 関節ローパワートルク上限値	第 n 関節のローパワートルク上限値を、第 n 関節の瞬時最大トルクに対する割合を表す整数値で指定します。

結果

パラメーターが省略されているときは、現在の LimitTorqueLP 値を表示します。
本コマンドにより値が変更されていない場合は、デフォルト値が表示されます。

解説

ローパワーモード時のトルク上限値を制限します。デフォルトでは、ローパワー動作に対して必要なトルクが上限値として設定されており (値は機種や軸によって異なる。15~60%程度)、本設定値を変更する必要はありませんが、例えば、周辺機器との衝突によるロボットや装置のダメージを少なくするため、衝突のない正常動作に必要なトルクを大きく超えるトルクが発生しないように制限したい場合に有効です。上限値は、正常動作をした場合のピークトルクを PTRQ で測定し、その値に変動分 (4 割を推奨) を考慮したマージンを加えた値とします。同じ値を異なるロボットに適用する場合は、さらに 1~2 割のマージンを加えてください。

PTRQ の値は、ローパワー時のデフォルトの最大トルクを 1.0 として表示します。例えば、変更前のデフォルト値が 27% で、PTRQ での測定値が 0.43 となった場合、 $27\% \times 0.43 \times 1.4 = 16.25$ となるので、整数で繰り上げて 17 を設定します。

本コマンドに、5% を下回る値や、デフォルト値を上回る値を設定することはできません。その場合、5 以下の設定値は 5 に、デフォルト値を超えた設定値はデフォルト値に丸められます。例えば、“LimitTorqueLP 100” とすると、デフォルト値は必ず 100 以下なので、全軸デフォルト値の設定に戻ります。設定後に設定値を表示し、実際に設定された上限値を確認してください。

コントローラーを再起動するまで、LimitTorqueLP の設定値は有効となります。

注意**低すぎる LimitTorqueLP 設定**

LimitTorqueLP は、特定の動作に対して、設定された加減速度で動作するために必要なトルクの大きさにかかわらず、設定されたトルク制限値を上限値とし、トルクを制限します。その結果、特定の動作が設定した上限値より大きなトルクを必要とする場合、ロボットは適切に動作をすることができず、位置偏差エラーが発生することが考えられます。

必ずローパワー状態での PTRQ を計測してから、トルク制限機能を使用してください。上記の状態が発生した場合は、トルクが不足しているため、トルク上限値を大きくし、正常に動作するように調整してください。

参照

LimitTorqueLP 関数, PTRQ

LimitTorqueLP 使用例

第 1 関節の最大トルクを 10% に制限して動作する例です。

```
Function main
Motor On
Power low
LimitTorqueLP 10,27,31,42 '第 1 関節の最大トルクを 10%に制限する
                              'その他の軸はデフォルトの値を設定
Go P1                        'Go 動作を実行する
End
```

LimitTorqueLP 関数

LimitTorqueLP 命令の設定値を返します。

書式

LimitTorqueLP(関節番号)

パラメーター

関節番号 1~9 の整数で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

LimitTorqueLP 命令の設定値を整数値で返します。

参照

LimitTorqueLP

LimitTorqueLP 関数使用例

`Print LimitTorqueLP(1)` '第 1 関節の LimitTorqueLP 値を表示

LimitTorqueStop

ハイパワーモード時にトルク上限に達した場合にロボットを停止する機能をオン、またはオフします。

書式

- (1) LimitTorqueStop 状態
- (2) LimitTorqueStop 状態, 関節番号
- (3) LimitTorqueStop

パラメーター

状態	On: トルク上限での停止機能を有効にします。 Off: トルク上限での停止機能を無効にします。
関節番号	1 から 6 までの関節番号を指定します。 (スカルロボットの場合、関節番号は 1 から 4 まで)

結果

パラメーターが省略されているときは、現在の LimitTorqueStop 状態を表示します。

解説

ハイパワー動作でのトルク上限での停止機能を有効にします。トルク上限 (デフォルトは 100%) で、ロボットを即時停止させます。LimitTorque によるトルク制限機能と併用することにより、ハイパワーモード時の衝突や接触による、ロボットや装置のダメージを低減する効果が得られます。

全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オフです。

コントローラー起動時にデフォルト値に戻りますが、それ以外の場合、本コマンドで明示的に設定しない限り、状態は変化しません。

トルク上限に達した場合、エラー 5040 “ハイパワー状態でモータートルク出力が異常”のメッセージを出し、ロボットを停止します。

参照

LimitTorque, LimitTorque 関数

LimitTorqueStop 使用例

第 1 関節の最大トルクを 30%に制限し、即時停止させる例です。

```
Function main
Motor On
Power high
Speed 20
Accel 20,20
LimitTorque 30,100,100,100 '第 1 関節の最大トルクを 30%に制限する
LimitTorqueStop On, 1      '第 1 関節は最大トルクで即停止する
Go P1                       'Go 動作を実行する
End
```

LimitTorqueStop 関数

LimitTorqueStop 命令の設定値を返します。

書式

LimitTorqueStop(関節番号)

パラメーター

関節番号 1~6 の整数で指定します。

戻り値

LimitTorqueStop 命令の設定値を整数値で返します。

0 = オフ

1 = オン

参照

LimitTorqueStop

LimitTorqueStop 関数使用例

Print **LimitTorqueStop**(1) '第1関節の LimitTorqueStop 値を表示

LimitTorqueStopLP

ローパワーモード時にトルク上限に達した場合にロボットを停止する機能をオン、またはオフします。

書式

- (1) LimitTorqueStopLP 状態
- (2) LimitTorqueStopLP 状態, 関節番号
- (3) LimitTorqueStopLP

パラメーター

状態	On: トルク上限での停止機能を有効にします。 Off: トルク上限での停止機能を無効にします。
関節番号	1 から 6 までの関節番号を指定します。 (スカルロボットの場合、関節番号は 1 から 4 まで)

結果

パラメーターが省略されているときは、現在の LimitTorqueStopLP 状態を表示します。

解説

ローパワー動作でのトルク上限での停止機能を有効にします。トルク上限で、ロボットを即時停止させます。LimitTorqueLP によるトルク制限機能と併用することにより、ローパワーモード時の衝突や接触による、ロボットや装置のダメージを低減する効果が得られます。
全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オフです。
コントローラー起動時にデフォルト値に戻りますが、それ以外の場合、本コマンドで明示的に設定しない限り、状態は変化しません。
トルク上限に達した場合、エラー 5041 “ローパワー状態でモータートルク出力が異常”のメッセージを出力し、ロボットを停止します。

参照

LimitTorqueLP, LimitTorqueLP 関数

LimitTorqueStopLP 使用例

第 3 関節の最大トルクを 15%に制限し、即時停止させる例です。

```
Function main
Motor On
Power low
LimitTorqueLP 20,27,15,42      '第 3 関節の最大トルクを 15%に制限する
                                'その他の軸はデフォルトの値を設定
LimitTorqueStopLP On, 3        '第 3 関節は最大トルクで即停止する
Go P1                          'Go 動作を実行する
End
```

LimitTorqueStopLP 関数

LimitTorqueStopLP 命令の設定値を返します。

書式

LimitTorqueStopLP(関節番号)

パラメーター

関節番号 1~6 の整数で指定します。

戻り値

LimitTorqueStopLP 命令の設定値を整数値で返します。

0 = オフ

1 = オン

参照

LimitTorqueStopLP

LimitTorqueStopLP 関数使用例

Print **LimitTorqueStopLP**(3) '第3関節の LimitTorqueStopLP 値を表示

LimZ

Jump コマンドのときの第 3 関節の高さ (Z 座標値) の初期値を設定します。

書式

- (1) LimZ Z 座標値
- (2) LimZ

パラメーター

Z 座標値 第 3 関節の動作範囲内にある座標の値を指定します。

結果

パラメーターが省略されているときは、現在の LimZ 値を表示します。

解説

Jump 命令実行時には、ロボットアームが第 3 関節 (Z 軸) 方向で上昇し、次に X-Y 平面で移動し、最後に第 3 関節 (Z 軸) 方向に下降しますが、LimZ は、そのときアームが第 3 関節 (Z 軸) 方向に動作する高さの上限を設定します。LimZ は、Jump 命令時の第 3 関節の動作範囲の最高座標のデフォルト値を設定します。Jump 命令時に特定の LimZ 値が設定されていなければ、最後に設定された LimZ 値が使われます。

注意

LimZ 値を 0 にリセットする

コントローラーの再起動、SFree, SLock, Motor On などの命令で、いずれも LimZ 値は、0 に初期化されます。

LimZ 値は Arm、Tool、または Local 座標には使えません

LimZ の高さ制限値は、ロボット座標の Z 座標値です。Arm、Tool、または Local 座標の Z 座標値ではありません。したがって、高さの異なるハンドまたは手先ツールを使うときには、注意してください。

LimZ は、Jump3 および Jump3CP には影響しません。

スパン動作は、必ずしも座標系の Z 軸に垂直とは限らないため、LimZ は、Jump3 および Jump3CP には影響しません。

参照

Jump

LimZ 使用例

Jump 操作時に LimZ を使う例です。

```
Function main
    LimZ -10          'LimZ のデフォルト値を設定
    Jump P1           'Jump 実行時-10 で水平移動
    Jump P2 LimZ -20  'Jump 実行時-20 で水平移動
    Jump P3           'Jump 実行時-10 で水平移動
End
```

LimZ 関数

LimZ 命令の設定値を返します。

書式

LimZ

戻り値

LimZ 命令の設定値を実数値で返します。

参照

LimZ

LimZ 関数使用例

```
Real savLimz

savLimz = LimZ
LimZ -25
Go pick
LimZ savLimz
```

LimZMargin

LimZ 設定値より高い位置で動作開始したときのエラー検出のマージンを設定、および設定値を返す。

書式

- (1) LimZMargin LimZ マージン
- (2) LimZMargin

パラメーター

LimZ マージン LimZ エラー検出のマージン値を指定します。

結果

パラメーターが省略されているときは、現在の LimZMargin 値を表示します。

解説

Jump 命令実行時には、LimZ で設定した高さに第 3 関節(Z 軸)が上昇しますが、Jump 動作開始時に LimZ 位置より高い位置に第 3 関節 (Z 軸)ある場合、エラーが発生します。LimZMargin はこのエラー検出に対してマージン値を設定します。デフォルトでは 0.02 mm が設定されています。

注意

LimZMargin 値をデフォルト値にリセットする

コントローラーの再起動、SFree, SLock, Motor On などの命令で、いずれも LimZMargin 値は、デフォルト値に初期化されます。

参照

LimZMargin 関数, LimZ

LimZMargin 使用例

Jump 操作時に LimZMargin を使う例です。

```
Function main
    LimZ -10                'LimZ のデフォルト値を設定
    LimZMargin 0.03        'LimZ のエラー検出マージンに 0.03 mm を設定
    Jump P1                 'Jump 実行時-10 で水平移動
    Jump P2 LimZ -20        'Jump 実行時-20 で水平移動
    Jump P3                 'Jump 実行時-10 で水平移動
End
```

LimZMargin 関数

LimZMargin 命令の設定値を返します。

書式

LimZMargin

戻り値

LimZMargin 命令の設定値を実数値で返します。

参照

LimZMargin, LimZ

LimZMargin 関数使用例

```
Real savLimzMargin

savLimzMargin = LimZMargin
LimZMargin 0.03
Jump pick
LimZ savLimzMargin
```

Line Input

入力データを 1 行読んで、そのデータを文字列変数に代入します。

書式

Line Input 文字列変数名\$

パラメーター

文字列変数名\$ 文字列変数名を指定します。(文字列変数は、最後に\$をつけてください。)

解説

Line Input はディスプレイデバイスからの 1 行入力データを読んで、Line Input 命令の文字列変数に代入します。Line Input 命令がユーザーからデータを受け取れる状態になると、ディスプレイデバイスにプロンプト"?" が表示されます。このプロンプトの後に入力されたデータ行が、文字列の値として代入されます。入力データを入力した後は、[ENTER]キーを押してください。

参照

Input, Input #, Line Input#, ParseStr

Line Input 使用例

Line Input の使用例です。

```
Function Main
  String A$
  Line Input A$ ' 入力データを 1 行読んで A$ に代入
  Print A$
End
```

上記プログラムを実行すると、下記のセッションが行われます。

```
?A, B, C
A, B, C
```

Line Input

ファイル、通信ポート、データベース、デバイスから、1 行データを読み込みます。

書式

Line Input #ポート番号, 文字列変数名\$

パラメーター

ポート番号	ファイル、通信ポート、データベース、デバイスを示す ID 番号です。 ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。 通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。 データベース番号は、OpenDB ステートメントで指定した番号です。 デバイス ID は以下の数値です。 21 RC+ 24 TP (TP1 のみ) 20 TP3
文字列変数名\$	文字列変数名を指定します。(文字列変数は、最後に\$をつけてください。)

解説

Line Input # は、ポート番号で指定したデバイスから 1 行データを読み、文字列変数名\$で指定された変数に代入します。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

参照

Input, Input #, Line Input

Line Input # 使用例

下記例では、通信ポート 1 から文字列データを受け取り、文字列変数 A\$ に代入します。

```
Function lintest
  String a$
  Print #1, "Please input string to be sent to robot"
  Line Input #1, a$
  Print "Value entered = ", a$
Fend
```

LJM 関数

指定ポイントに対して参照ポイントから見て最も関節移動量が少なくなるように姿勢フラグを変換したポイントデータを返します。

書式

LJM (ポイント指定 [, 参照ポイント指定 [, 姿勢フラグ選択]])

パラメーター

ポイント指定 対象とするポイントデータを指定します。

参照ポイント指定 基準にするポイントデータを指定します。参照ポイント指定を省略した場合は現在位置 (Here)を参照ポイントとします。

姿勢フラグ選択

垂直 6 軸型

1: J4 軸が最短移動量となるように、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag、J1Flag によって変換します。姿勢フラグ選択を省略したときのデフォルトです。

2: J4Flag および J6Flag によって変換します。

3: J5 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag、J1Flag によって変換します。

4: J6 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag、J1Flag によって変換します。

“姿勢フラグ 選択”	腕 姿勢	肘 姿勢	手首 姿勢	J1Flag	J4Flag	J6Flag	移動量が 最短となる軸の 優先順
1	—	—	○	○	○	○	J4
2	—	—	—	○	○	○	—
3	—	—	○	○	○	○	J5
4	—	—	○	○	○	○	J6

Note: “—” は、“参照ポイント指定”で指定した姿勢と同じ姿勢です。

RS シリーズ

1: 腕姿勢 (Hand フラグ)、J1Flag および J2Flag によって変換します。“姿勢フラグ選択”を省略したときのデフォルトです。

2: 腕姿勢 (Hand フラグ)、J1Flag および J2Flag によって変換します。“姿勢フラグ選択”が変換されるときに U 軸が動作範囲外となるエラーを防ぎます。

N2 シリーズ

1: J1, J5 軸の優先順で、関節移動量が少なくなる姿勢に変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。肘姿勢 (Elbow フラグ)は、必ず上肘姿勢になります。“姿勢フラグ選択”を省略したときのデフォルトです。

2: J1, J4 軸の優先順で、関節移動量が少なくなる姿勢に変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。肘姿勢 (Elbow フラグ)は、必ず上肘姿勢になります。

3: J4 軸が最短移動量となるように、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。

4: J4Flag および J6Flag によって変換します。

5: “参照ポイント指定”で指定した姿勢と異なる腕姿勢 (Hand フラグ)に変更し、J5 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。また、肘姿勢 (Elbow フラグ)は、必ず上肘姿勢になります。

6: “参照ポイント指定”で指定した姿勢と異なる腕姿勢 (Hand フラグ)に変更し、J4 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。また、肘姿勢 (Elbow フラグ)は、必ず上肘姿勢になります。

7: 肘姿勢 (Elbow フラグ)を下肘姿勢にし、J1, J5 軸の優先順で最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。

8: 肘姿勢 (Elbow フラグ)を下肘姿勢にし、J1, J4 軸の優先順で最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。変換対象となる姿勢は、腕姿勢 (Hand フラグ)、肘姿勢 (Elbow フラグ)、手首姿勢 (Wrist フラグ)、J4Flag および J6Flag です。

“姿勢フラグ 選択”	腕 姿勢	肘 姿勢	手首 姿勢	J4Flag	J6Flag	移動量が 最短となる軸の 優先順
1	○	*1	○	○	○	J1>J5
2	○	*1	○	○	○	J1>J4
3	-	-	○	○	○	J4
4	-	-	-	○	○	-
5	*2	*1	○	○	○	J5
6	*2	*1	○	○	○	J4
7	○	*3	○	○	○	J1>J5
8	○	*3	○	○	○	J1>J4

Note: “-” は、“参照ポイント指定”で指定した姿勢と同じ姿勢です。

*1: 上肘姿勢

*2: “参照ポイント指定”で指定した姿勢と腕姿勢が異なります。

*3: 下肘姿勢

N6 シリーズ

1: J4 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。“姿勢フラグ選択”を省略したときのデフォルトです。

2: J4Flag および J6Flag によって変換します。

3: J5 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。

4: J6 軸が最短移動量となるように手首姿勢 (Wrist フラグ)、J4Flag および J6Flag によって変換します。

“姿勢フラグ 選択”	腕 姿勢	肘 姿勢	手首 姿勢	J1Flag	J4Flag	J6Flag	移動量が 最短となる軸の 優先順
1	-	-	○	○	○	○	J4
2	-	-	-	○	○	○	-
3	-	-	○	○	○	○	J5
4	-	-	○	○	○	○	J6

Note: “-” は、“ポイント指定”で指定した姿勢と同じ姿勢です。

解説

垂直 6 軸型ロボットや N シリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは上記ポイント演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。

LJM 関数はこのような手首部の意図しない回転を防止するためにポイントデータの姿勢フラグを適切に変換します。

また、N シリーズでは、腕姿勢フラグや肘姿勢フラグを変更することで、タクトタイムの短縮や垂直 6 軸ロボットに必要な回避ポイントのティーチングを省くことも可能です。

RS シリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、第 1 アームが意図しない方向へ回転することがあります。LJM 関数はこのような第 1 アームの意図しない回転を防止するために、ポイントデータの姿勢フラグを適切に変換します。

また、RS シリーズでは、姿勢フラグが変換されたときに U 軸が動作範囲外に動作しようとしてエラーが発生することがあります。LJM 関数はこのような U 軸の動作範囲外エラーを防止するために、U 軸の目標角度を動作範囲内の目標角度に補正します。これは、姿勢フラグ選択で 2 を指定することによって利用可能となります。

垂直 6 軸型、N シリーズ、RS シリーズ以外のロボットでは、指定ポイントをそのまま返します。

注意

参照ポイントの省略と並列処理

参照ポイントの省略と並列処理を 1 つの動作命令内で混在させることはできません。

```
Go LJM(P10) ! D10; MemOn 1 !
```

上記のような使い方はできません。

```
P999 = Here
```

```
Go LJM(P10,P999) ! D10; MemOn 1 !
```

上記のようなプログラムに変更してください。

N2 シリーズの姿勢フラグ選択について

- 姿勢フラグ選択 1, 2:

ロボットのタクトタイムを短縮させたい場合、姿勢フラグ 1、または 2 を選択してください。

第 1 関節の移動量が最少となる姿勢をとるため、ほとんどの動作でタクトタイムが最短となる動作を行うことができます。第 5 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 1 を、第 4 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 2 を選択してください。

- 姿勢フラグ選択 3, 4:

垂直 6 軸型と同じように使用したい場合に選択してください。

姿勢フラグ 3 は垂直 6 軸型の姿勢フラグ 1 と同じです。

姿勢フラグ 4 は垂直 6 軸型の姿勢フラグ 2 と同じです。

- 姿勢フラグ選択 5, 6:

ロボット動作中にハンドがロボット周辺の壁などに接触する場合、姿勢フラグ 5、または 6 を選択してください。

ハンドがロボットの原点近傍を通過するため、ロボットが周囲の障害物に当たりにくい動作を行うことができます。第 5 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 5 を、第 4 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 6 を選択してください。

- 姿勢フラグ選択 7, 8:

下肘姿勢を取りたい場合、姿勢フラグ 7 または 8 を選択してください。

動作によっては、ロボットは姿勢選択フラグ 5, 6 のように原点近傍を通過する動作を行うため、ロボットの周囲に障害物がある場合に、障害物に当たりにくい動作を行うことができます。第 5 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 7 を、第 4 関節の移動量を少なくする動作を行う場合は姿勢選択フラグ 8 を選択してください。

ローカル番号

LJM 関数で返されるポイントのローカル番号は、“ポイント指定”と同じローカル番号になります。

参照

Pallet

LJM 関数使用例

```

Function main
  Integer i, j

  P0 = XY(300, 300, 300, 90, 0, 180)
  P1 = XY(200, 280, 150, 90, 0, 180)
  P2 = XY(200, 330, 150, 90, 0, 180)
  P3 = XY(-200, 280, 150, 90, 0, 180)

  Pallet 1, P1, P2, P3, 10, 10

  Motor On
  Power High
  Speed 50; Accel 50, 50
  SpeedS 1000; AccelS 5000

  Go P0
  P11 = P0 -TLZ(50)

  For i = 1 To 10
    For j = 1 To 10
      'ポイントの指定
      P10 = P11                                '退避ポイント
      P12 = Pallet(1, i, j)                    '目標ポイント
      P11 = P12 -TLZ(50)                       '接近開始ポイント
      '各ポイントの LJM 変換
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      '動作実行
      Jump3 P10, P11, P12 C0
    Next
  Next
Fend

Function main2
  P0 = XY(300, 300, 300, 90, 0, 180)
  P1 = XY(400, 0, 150, 90, 0, 180)
  P2 = XY(400, 500, 150, 90, 0, 180)
  P3 = XY(-400, 0, 150, 90, 0, 180)
  Pallet 1, P1, P2, P3, 10, 10

  Motor On
  Power High
  Speed 50; Accel 50, 50
  SpeedS 1000; AccelS 5000

  Go P0

  Do
    'ポイントの指定
    P10 = Here -TLZ(50)                                '退避ポイント
    P12 = Pallet(1, Int(Rnd(9)) + 1, Int(Rnd(9)) + 1) '目標ポイント
    P11 = P12 -TLZ(50)                                '接近開始ポイント

    If TargetOK(P11) And TargetOK(P12) Then            'ポイントのチェック
      '各ポイントの LJM 変換
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      '動作実行
      Jump3 P10, P11, P12 C0
    EndIf
  Loop
Fend

```

LoadPoints

ポイントファイルを、ロボットのポイントメモリーエリアにロードします。

書式

LoadPoints ファイル名 [, Merge]

パラメーター

ファイル名	ロボットのポイントメモリーエリアにロードするファイルを指定する文字列 拡張子は“.pts”固定です。拡張子を省略した場合、“.pts”が追加されます。指定するファイルはプロジェクト内のファイルに限ります。 パスを指定することはできません。また、ChDisk などの影響を受けません。 詳細は、ChDisk を参照してください。
Merge	新しいポイントをロードする前に現在のポイントをクリアしたくないときに設定しておきます。これを設定すると、設定されているポイントに新しいポイントが追加されます。追加するポイントがすでにファイルの中にある場合は、上書きします。省略可能です。

解説

LoadPoints は、コントローラーのメインメモリーにポイントファイルをロードします。

ポイントファイルを統合するときは、Merge を設定します。例えば、共通ポイントを 0~100 の範囲で含む 1 つのメインポイントファイルがあるとしします。それらのポイントをクリアせずに現在動作中の各パーツに対する新たなポイントファイルをロードするときは、Merge を設定します。その場合、範囲は 101~999 になります。

起こりやすいエラー

パスは指定できません

ファイル名が軌道を含む場合、エラーが発生します。

指定ファイルが見つからない場合 (ファイルが存在しない)

ファイルが見つからない場合はエラーとなります。

別のロボットのポイントファイル

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。この場合、プロジェクトエディターでポイントファイルを追加するか、SavePoints または ImportPoints を実行します。

参照

ImportPoints, Robot, SavePoints

LoadPoints 使用例

```
Function main

' 現在のロボットに共通のポイントをロード
LoadPoints "R1Common.pts"

' パーツモデル 1 のポイントを統合
LoadPoints "R1Model1.pts", Merge

ロボット 2
' ロボット 2 のポイントファイルをロード
LoadPoints "R2Model1.pts"

Fend
```

Local

ローカル座標系の定義と表示をします。

書式

- (1) Local ローカル座標系番号, (ポイント番号 1 : ポイント番号 2), (ポイント番号 3 : ポイント番号 4)
[, { L | R }], [BaseU]
- (2) Local ローカル座標系番号, 座標系データ
- (3) Local ローカル座標系番号, 原点, [X 軸指定], [Y 軸指定], [{ X | Y }]
- (4) Local ローカル座標系番号

パラメーター

ローカル座標系番号	ローカル座標系の番号を指定します。ローカル座標系は 1~15 の整数で、合計 15 まで定義することができます。
ポイント番号 1, ポイント番号 3 ポイント番号 2, ポイント番号 4 L R	ローカル座標系のポイントデータをポイント変数で指定します。 ベース座標系のポイントデータをポイント変数で指定します。 ローカル原点を、左 (1 番)または右 (2 番)にあわせませす。省略可能です。
BaseU	指定すると U 軸座標はベース座標系を使います。省略可能で、省略すると、U 軸座標はローカル座標系を使います。
座標系データ	ローカル座標系の原点と向きをポイントデータで直接指定します。スカルロボット (RS シリーズ含む)の場合、V 座標と W 座標を“0”を指定してください。
原点	ローカル座標系原点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。
X 軸指定	ローカル座標系の X 軸上の点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。省略可能です。
Y 軸指定	ローカル座標系の Y 軸上の点を定義するロボット座標系上の位置を、P#(整数)または P(式)で指定します。省略可能です。
X	原点と X 軸指定を結んだ直線をローカル座標系の X 軸と定義します。X 軸と 3 つのポイントで作られた平面からローカル座標系が算出されるため、Y 軸指定は、必ずしも Y 軸上のポイントになりません。省略可能です。(デフォルト)
Y	原点と Y 軸指定を結んだ直線をローカル座標系の Y 軸と定義します。Y 軸と 3 つのポイントで作られた平面からローカル座標系が算出されるため、X 軸指定は、必ずしも X 軸上のポイントになりません。省略可能です。

解説

- (1) Local は、ベース座標系の 2 点の位置データ、ポイント番号 2 とポイント番号 4 に合致するローカル座標系の 2 点、ポイント番号 1 およびポイント番号 3 を指定して、ローカル座標系を定義します。

例:

```
Local 1, (P1:P11), (P2:P12)
```

P1 と P2 はローカル座標系の点です。P11 と P12 はベース座標系の点です。

もし、ローカル座標系で指定した 2 点間の距離が、ベース座標系の 2 点間の距離と異なる場合は、ローカル座標系の 2 点の midpoint とベース座標系の 2 点の midpoint が一致する位置にローカル座標系を定義します。

同様に、2 つの座標系の midpoint を使って、ローカル座標系の Z 軸を定義します。

- (2) 原点とベース座標系に対する角度でローカル座標系を定義します。

例:

```
Local 1, XY(x, y, z, u)
Local 1, XY(x, y, z, u, v, w)
Local 1, P1
```

- (3) 原点, X 軸上ポイント, Y 軸上ポイントを指定して、3 次元ローカル座標系を定義します。各ポイントの X, Y, Z 座標のみを使用し、U, V, W 座標は無視します。パラメーターの X を指定すると、X 軸指定はローカル座標系の X 軸上になり、Y 軸指定の Z 座標のみ使用します。パラメーターの Y を指定すると、Y 軸指定はローカル座標系の Y 軸上になり、X 軸指定の Z 座標のみ使用します。

例:

```
Local 1, P1, P2, P3
Local 1, P1, P2, P3, X
Local 1, P1, P2, P3, Y
```

- (4) 指定されたローカルの設定を表示します。

L/R パラメーターの使用

ローカル座標系は、前述したように基本的に midpoint によって定義しますが、オプションの L または R を使って座標系の左右を指定することができます。

左ローカル

左ローカルは、ローカル座標系の点、ポイント番号 1 が、ベース座標系の点、ポイント番号 2 に一致する位置にローカル座標を定義します。(Z 軸方向も含まれます。)

右ローカル

右ローカルは、ローカル座標系の点ポイント番号 3 が、ベース座標系の点ポイント番号 4 に一致する位置に、ローカル座標を定義します。(Z 軸方向も含まれます。)

BaseU パラメーターの使用

BaseU パラメーターを省略すると、設定された 4 点の XY 座標値に適合するようローカル座標系の U 軸が自動的に補正されます。したがって、ベース座標系の 2 点は、最初から U 軸座標値を含んでいることもあります。

ローカル座標系の U 軸は、自動的に補正するよりも、ティーチングによる回転軸の補正など、ベース座標系の 2 点の U 軸値に基づいて補正を行うことをお勧めします。そのときには、BaseU パラメーターの設定が必要です。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

水平多関節ロボットの場合、V,W に対しての設定はしないでください。

水平多関節ロボットを使用する場合、ベース座標系にて V,W 座標に数値を設定しないか、“0”を入力してください。設定した場合は、J4 での範囲外にエラーなどが発生する場合があります。

参照

ArmSet, Base, ECPSet, LocalClr, TLSet, Where

Local 使用例

コマンドウィンドウからの操作例です。

左ローカルでローカル座標系の原点を定義する例:

```
> p1 = 0, 0, 0, 0/1  
> p2 = 100, 0, 0, 0/1  
> p11 = 150, 150, 0, 0  
> p12 = 300, 150, 0, 0  
> local 1, (P1:P11), (P2:P12), L
```

```
> p21 = 50, 0, 0, 0/1  
> go p21
```

原点をローカル座標系の原点と定義する例:

```
> local 1, 100, 200, -20, 0
```

X 軸を 45 度回転させた原点をローカル座標系の原点と定義する例:

```
> local 2, 50, 200, 0, 0, 45, 0
```

P2 をローカル座標系の X 軸に合わせた位置を 3 次元ローカル座標系の原点と定義する例:

```
> local 3, p1, p2, p3, x
```

P3 をローカル座標系の Y 軸に合わせた位置を 3 次元ローカル座標系の原点と定義する例:

```
> local 4, p1, p2, p3, y
```

Local 関数

設定されたローカル座標系データを返す関数です。

書式

Local(ローカル座標系番号)

パラメーター

ローカル座標系番号 ローカル座標系番号 (1~15 の整数)を、式または数値で指定します。

戻り値

設定されたローカル座標系データをポイントデータとして返します。

参照

Local

Local 関数使用例

```
P1 = Local (1)
```

LocalClr

ローカル座標系をクリア (未定義)にします。

書式

LocalClr ローカル座標系番号

パラメーター

ローカル座標系番号 設定をクリア (未定義)にするローカル座標系番号 (1~15 の整数)を、式または数値で指定します。

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

Arm, ArmSet, ECPSet, Local, Tool, TLClr, TLSet

LocalClr 使用例

```
LocalClr 1
```


LocalDef 関数

ローカルの設定状態を返します。

書式

LocalDef(ローカル座標系番号)

パラメーター

ローカル座標系番号 状態を返すローカルの座標系番号 (1~15 の整数)を指定します。

戻り値

指定したローカルが設定されている場合“True”、設定されていない場合“False”を返します。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLCI, TLSet

LocalDef 関数使用例

```
Function DisplayLocalDef(localNum As Integer)

    If LocalDef(localNum) = False Then
        Print "Local ", localNum, "is not defined"
    Else
        Print "Local 1: ",
        Print Local(localNum)
    EndIf
End
```

Lof 関数

指定した RS-232C ポート、または TCP/IP ポートのバッファの受信データの行数を返します。

書式

Lof(通信ポート番号)

パラメーター

通信ポート番号	OpenCom (RS-232C)、または OpenNet (TCP/IP)ステートメントで指定した番号を指定します。

戻り値

受信バッファのデータ行数を返します。受信データが存在しない場合は“0”を返します。

解説

Lof 関数は、指定した通信ポートの受信データの有無を確認します。受信データは Input# 命令により引き出されます。

Wait 命令により Lof 関数の戻り値を待機することができます。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

PC COM ポート (1001~1008)を使用している場合、Wait 命令で Lof 関数を使用することはできません。

参照

ChkCom, ChkNet, Input#, Wait

Lof 関数使用例

次の例はコマンドウィンドウで実行します。通信ポート 1 の受信データの行数を表示します。

```
>print lof(1)
5
>
```

LogIn

他のユーザーとして、EPSON RC+にログインします。

書式

LogIn ログ ID, パスワード

パラメーター

ログ ID	ユーザーログイン ID の文字列式
パスワード	ユーザーパスワードの文字列式

解説

アプリケーションで EPSON RC+のセキュリティーを操作することができます。例えば、他のユーザーがシステムにログインするためのメニューを表示することができます。ユーザーは独自のセキュリティーの権限を所有しています。セキュリティーについての詳細は、EPSON RC+ ユーザーズガイドを参照してください。

開発環境でプログラムを実行した場合、プログラムが停止したあと、プログラムがスタートする前のユーザーに戻ります。

Auto モードでオペレーターウィンドウを実行しているときは、Auto LogIn が有効になっている限り、アプリケーションがゲストユーザーとしてログインします。この場合、EPSON RC+ システムで設定されていれば、アプリケーションがカレントウィンドウズユーザーとしてログインします。

注意

このコマンドは、セキュリティーオプションが有効になっているときのみ使用できます。

参照

GetCurrentUser\$ 関数

LogIn 使用例

```
Integer errCode
errCode = LogIn("operator", "oprpass")
```

Long

Long 型の変数を宣言します。(4 バイト整数型変数)

書式

Long 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 Long 型に宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Long は整数型変数の宣言をします。Long 型変数の範囲は -2147483648~2147483647 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Real, Short, String, UByte, UInt32, UInt64, UShort

Long 使用例

次の例は Long を使って Long 型の変数を宣言するプログラムです。

```
Function longestest

    Long A(10)           'Long 型の 1 次元配列
    Long B(10, 10)       'Long 型の 2 次元配列
    Long C(5, 5, 5)      'Long 型の 3 次元配列
    Long var1, arrayVar(10)
    Long i
    Print "Please enter a Long Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter a Long Number"
        Input arrayVar(i)
        Print "Value Entered was ", arrayVar(i)
    Next i
End
```

LSet\$ 関数

指定された文字列の最後にスペースを加えて、指定の長さにした文字列を返します。

書式

LSet\$ (文字列,文字列の長さ)

パラメーター

文字列	文字列式を指定します。
文字列の長さ	返される文字列の長さを示す、整数または式を指定します。

戻り値

指定された文字列の最後に、スペースを加えた文字列を返します。

参照

RSet\$, Space\$

LSet\$ 関数 使用例

```
temp$ = "123"  
temp$ = LSet$(temp$, 10) ' temp$ = "123      "
```

LShift 関数

数値データを指定したビット数だけ左シフトします。

書式

LShift(数値, シフトビット数)

パラメーター

数値	シフトする整数値を指定します。
シフトビット数	左シフトするビット数 (0~31 の整数値)を指定します。

戻り値

指定した数値を、指定したビット数だけ左シフトした結果を返します。

解説

指定数値を左 (桁の大きい方)に、指定ビット数シフトします。シフトした分の下位のビットは、常に 0 がセットされます。

Lshift は数値 * 2 シフトビット数 (数値を 2 シフトビット数の回数だけ掛け算する)のと同じです。

注意

数値データ型

数値型は数種類あります。Lshift は Byte 型, Double 型, Int32 型, Integer 型, Long 型, Real 型, Short 型, UByte 型, UInt32 型, UShort 型の数値に使うことができます。

参照

And, LShift64, Not, Or, RShift, RShift64, Xor

LShift 関数使用例

```
Function lshiftst
    Integer i
    Integer num, snum
    num = 1
    For i = 1 to 10
        Print "i =", i
        snum = LShift(num, i)
        Print "The shifted num is ", snum
    Next i
Fend
```

コマンドウィンドウから Lshift 関数の結果を返す、その他の例です。

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

LShift64 関数

数値データを指定したビット数だけ左シフトします。

書式

LShift64(数値, シフトビット数)

パラメーター

数値	シフトする整数値を指定します。
シフトビット数	左シフトするビット数 (0~63 の整数値)を指定します。

戻り値

指定した数値を、指定したビット数だけ左シフトした結果を返します。

解説

指定数値を左 (桁の大きい方)に、指定ビット数シフトします。シフトした分の下位のビットは、常に 0 がセットされます。

Lshift64 は数値 * 2 シフトビット数 (数値を 2 シフトビット数の回数だけ掛け算する)のと同じです。

注意

数値データ型

数値型は数種類あります。Lshift64 は Int64 型, UInt64 型の数値に使うことができます。

参照

And, LShift, Not, Or, RShift, RShift64, Xor

LShift64 関数使用例

```
Function lshiftst
    Int64 i
    Int64 num, snum
    num = 1
    For i = 1 to 10
        Print "i =", i
        snum = LShift64(num, i)
        Print "The shifted num is ", snum
    Next i
End
```

コマンドウィンドウから Lshift64 関数の結果を返す、その他の例です。

```
> Print LShift64(2,2)
8
> Print LShift64(5,1)
10
> Print LShift64(3,2)
12
>
```

LTrim\$ 関数

文字列の左側スペースを削除して返します。

書式

LTrim\$ (文字列)

パラメーター

文字列 文字列式を指定します。

戻り値

左側のスペースを削除した文字列を返します。

参照

RTrim\$, Trim\$

LTrim\$ 関数使用例

```
str$ = "  data  "  
str$ = LTrim$(str$)    ' str$ = "data  "
```


Mask 演算子

Wait 命令の条件を示す値をビット単位でマスクします。

書式

Wait 値 1 Mask 値 2

パラメーター

値 1 Wait の入力条件を示す値を指定します。

値 2 result で返す数値を指定します。

解説

Mask 演算子は、Wait 入力条件を示す値をビット And 演算します。

参照

Wait

Mask 演算子使用例

・入力ポート 0 の下位 3 ビットが 1 になるまで待機
Wait In(0) **Mask** 7 = 1

MCal

インクリメンタルエンコーダーのロボットの原点復帰(メカ原点の検出)を行います。

書式

MCal

解説

インクリメンタル型エンコーダーのロボットでは原点復帰 (メカ原点の検出)をしなければなりません。この原点復帰は、電源投入後に実行してください。原点復帰を行う前に、動作コマンドや、その他の現在位置データを必要とするコマンドを実行するとエラーになります。

原点復帰は、MCordr コマンドで指定された関節順序で実行されます。工場出荷時の MCordr 初期値は、マニピュレーターの機種によって異なりますので、詳細については対象となるマニピュレーターのマニュアルを参照してください。

起こりやすいエラー

MCal 実行前に動作命令を実行しようとしたとき

原点復帰を行う前に、動作コマンドや、その他の現在位置データを必要とするコマンド (例えば Plist* 命令など)を実行するとエラーになります。

アブソリュートエンコーダー搭載のロボット

アブソリュートエンコーダー搭載のロボットでは、MCal は必要ありません。

ロボット設置時の注意

第 3 関節の原点復帰のためのスペース

第 3 関節が原点復帰するときには、まず上昇移動してから下降して原点に停止します。そのため、第 3 関節が原点復帰できるスペースを確保しておくことが非常に重要です。推奨値としては、Z 上限値の上部に 6mm 以上のスペースを確保しておくことをお勧めします。(ロボット上部 6mm 間のスペースにツールや治具をつけないでください。)

参照

Hofs, Home, Hordr, Mcorg, MCordr

MCal 使用例

モニターウィンドウからの操作例です。

```
> Motor On
> Mcal
>
```

MCalComplete 関数

MCal の状態を返します。

書式

MCalComplete

戻り値

MCal が正常に動作した場合は True、それ以外は False を返します。

参照

MCal

MCalComplete 関数使用例

```
If Not MCalComplete Then  
    MCal  
EndIf
```

MCordr

MCal による原点復帰時の、関節動作順序の指定と表示を行います。
インクリメンタルエンコーダー搭載のロボットのみ、必要です。

書式

- (1) MCordr 設定値 1, 設定値 2, 設定値 3, 設定値 4[, 設定値 5][, 設定値 6][, 設定値 7][, 設定値 8][, 設定値 9]
(2) MCordr

パラメーター

- 設定値 1 MCal プロセスの第 1 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 2 MCal プロセスの第 2 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 3 MCal プロセスの第 3 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 4 MCal プロセスの第 4 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 5 MCal プロセスの第 5 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 6 MCal プロセスの第 6 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 7 MCal プロセスの第 7 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 8 MCal プロセスの第 8 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)
- 設定値 9 MCal プロセスの第 9 ステップで原点復帰する関節 (0~9)を、ビットパターン (2 進数の値)で指定します。(下記ビットパターン表を参照してください。)

戻り値

パラメーターを省略すると現在のメカ原点復帰順序を表示します。

解説

電源オン時、アームを動作させる前には、必ず MCal 命令を実行してください。MCal 命令が実行されると各関節は、それぞれの原点復帰位置に移動します。

MCal コマンド実行時の関節の動作順序を指定します。設定値 1 で指定された関節が動作し、原点復帰を終了したら、次に設定値 2 で指定された関節が動作、というように順次、設定値 3 の関節、設定値 4 の関節と原点復帰します。

MCordr 命令の意義は、原点復帰時の各関節の復帰順序を、ユーザーが変更できるようにすることです。復帰順序は、9 つのステップに分かれて設定されています。ユーザーは MCordr により各ステップで復帰する関節を指定することができます。各ステップで復帰する関節は、複数を指定することも可能です。しかし、一般的には、第 3 関節を最初にステップ 1 で移動させ、他の関節をそれ以降のステップで復帰させることをお勧めします。(注意を参照してください。)

MCordr 命令では、9 つのステップに対しビットパターンが指定されなければなりません。各関節に対し、ビットパターンが定められています。あるステップでビットが“1”であれば、対応する関節が原点復帰します。ビットが“0”であれば、対応する関節は、そのステップにおいては原点復帰しません。各関節に対するビットパターンは次のように割り当てられています。

ビットパターン表

関節名:	第 1 関節	第 2 関節	第 3 関節	第 4 関節	
ビット番号:	bit 0	bit 1	bit 2	bit 3	
2 進数表記:	&B000001	&B000010	&B000100	&B001000	
関節名:	第 5 関節	第 6 関節	第 7 関節	第 8 関節	第 9 関節
ビット番号:	bit 4	bit 5	bit 6	bit 7	bit 8
2 進数表記:	&B010000	&B100000	&B100000	&B10000000	&B100000000
			0		

注意

MCordr と Hordr の違い

Hordr と MCordr コマンドは重要な点が異なります。MCordr は、MCal とともに用いられ、ロボットの原点復帰時の関節復帰順序を指定しますが、Hordr は、Home とともに用いられ、ホーム位置への関節復帰順序を指定します。

原点復帰位置への復帰順序のデフォルト設定

工場出荷時の設定は、次のようになっています。

第 1 ステップで第 3 関節 (Z) が復帰します。

第 2 ステップで、第 1 関節 (X)、第 2 関節 (Y)、第 4 関節 (U) が同時に原点復帰位置に復帰します。

第 3 および第 4 ステップは使われていません。デフォルト値は次のとおりです。

```
MCordr &B0100, &B1011, 0, 0
```

通常第 3 関節 (Z) を先に原点復帰させます

最初に単独で第 3 関節 (Z) を復帰させる理由は、水平移動が行われる前にツールをワーク表面から移動させることにあります。これは、原点復帰の際に、ツールが動作エリア内のものに干渉することを防ぐためです。

MCordr 値は保持されます

MCordr 値は、ユーザーが変更を行うか、ロボットが再設定されない限り、保持されます。

参照

MCal

MCordr 使用例

下記はモニターウィンドウからの 4 軸ロボットの操作例です。

この例では、原点復帰順序を 2 進数で次のように設定します

第 1 ステップで第 3 関節、第 2 ステップで第 1 関節、第 3 ステップで第 2 関節、第 4 ステップで第 4 関節が原点復帰します。

```
> MCordr &B0100, &B0001, &B0010, &B1000
```

この例では、原点復帰順序を 10 進数で次のように設定します
第 1 ステップで第 3 関節、第 2 ステップで第 1 関節, 第 2 関節, 第 4 関節が同時に原点復帰します。

```
> MCordr 4, 11, 0, 0
```

次の例では、現在の原点復帰順序を 10 進数で表示しています。

```
>mcordr  
4, 11, 0, 0  
>
```

MCordr 関数

MCordr パラメーター設定を返します。

書式

MCordr(設定値番号)

パラメーター

設定値番号 参照する設定値番号 (1~9 の整数)を、式または数値で指定します。

戻り値

原点復帰する指定した関節の設定値を、2 進数の値 (整数)で返します。

解説

MCal により原点復帰するときの関節の動作順序を返します。

参照

MCal

MCordr 関数使用例

MCordr 関数を使ったプログラム例です。

```
Integer a  
a = MCordr (1)
```

MemIn 関数

指定したメモリーI/O ポートの状態を返します。各ポートには 8 個のメモリービットがあります。

書式

MemIn(ポート番号)

パラメーター

ポート番号 メモリーI/O のバイトを指定します。

戻り値

0~255 までの整数を返します。戻り値は、8ビットで、各ビットがメモリーI/O ビット 1 個に対応します。

解説

MemIn は、1 度にメモリーI/O ビット 8 個の値を見ることができます。MemIn 命令は、メモリーI/O ビット 8 個の状態を 1 つの変数に保存したり、Wait とともに用いることで、2 個以上の I/O ビットに関係する特定の条件に一致するまで、プログラムを待機したりできます。

1 度に 8 ビットの値を得ることができるので、戻り値の範囲は、0~255 になります。下記の表で、それぞれの戻り値と個々のメモリーI/O ビットの状態の対応を、参照してください。

メモリーI/O ビット表 (ポート 0 使用時)

戻り値	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

メモリーI/O ビット表 (ポート 31 使用時)

戻り値	255	254	253	252	251	250	249	248
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

注意

MemIn と MemSw の違い

MemSw 命令により、メモリーI/O ビット 1 の値が読み取れます。MemSw の戻り値は 0 または 1 で、メモリーI/O ビットが On であるか Off であるかを表わします。MemSw はメモリーI/O ビットを個別にチェックします。MemIn 命令は、メモリーI/O ビットの状態をチェックするという点で、MemSw 命令に似ています。ですが、MemSw 命令はメモリーI/O を、1 ビットずつ状態をチェックするのに対し、MemIn 命令は 8 ビット同時にチェックします。MemIn の戻り値は、0~255 までの値を返します。その値により、8 ビットのうちどれが On で、どれが Off であるかがわかります。

参照

In, InBCD, Off, MemOff, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemIn 関数使用例

下記は、最初の 8 個のメモリーI/O の現在の値を求めたあと、8 個すべての値が“0”であることを確認して次に進むプログラムの例です。戻り値が“0”でない場合は、エラーメッセージが表示され、タスクは停止します。

```
Function main
  Integer var1

  var1 = MemIn(0)  '最初の 8 個のメモリーI/O ビットの値を取得
  If var1 = 0 Then
    Go P1
    Go P2
  Else
    Print "Error in initialization!"
    Print "First 8 memory I/O bits were not all set to 0"
  EndIf
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memout 0, 1
> print MemIn(0)
1
> memon 1
> print MemIn(0)
3
> memout 31, 3
> print MemIn(31)
3
> memoff 249
> print MemIn(31)
1
>
```

MemInW 関数

メモリーI/O ポートの状態をワード単位で返します。
ワードポートは 16 個のメモリーI/O ビットで構成されます。

書式

MemInW(ワードポート番号)

パラメーター

ワードポート番号 I/O のワードポートを指定します。

戻り値

メモリーI/O ポートの状態 (0~65535 の Long 型整数)を返します。

参照

MemIn, MemOut, MemOutW

MemInW 関数使用例

```
Long word0  
  
word0 = MemInW(0)
```

MemOff

メモリーI/O の指定ビットをオフします。

書式

MemOff { ビット番号 | メモリーI/O ラベル }

パラメーター

ビット番号 メモリーI/O のビットを整数で指定します。

メモリーI/O ラベル メモリーI/O ラベルを指定します。

解説

MemOff は、指定したメモリーI/O のビットをオフ (0 に) します。MemSw 命令で指定したメモリービットの状態チェックを行います。Wait 命令もメモリービットに使われ、指定したメモリーI/O 状態になるまでシステムを待たせます。

注意

メモリーI/O のオフ

メモリーI/O は、コントローラー再起動時にオフされます。非常停止, 安全扉開, プログラム終了, Reset, EPSON RC+ 再起動では、オフされません。

参照

In, MemIn, InBCD, Off, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemOff 使用例

下記は、それぞれが動作命令を起動する 2 つのタスクを使った例です。これら 2 つのタスクには、お互いに相手がロボット動作命令を終了したところで順番に制御をゲインするよう、インターロック機能が働いています。そのため、2 つのタスクが順番にそれぞれ指示どおりの動作ステートメントを実行することができます。MemSw は Wait 命令とともに用いて、安全のために、再度動作を始める前に、メモリーI/O ビット 1 が適切な値になるのを待ちます。MemOn と MemOff は、正確に同期するように、メモリーI/O をオン、またはオフします。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For i = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend
```

```
Function task2
  Integer I
  For i = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> MemOn 1          'メモリーI/O ビット 1 をオン
> Print MemSw(1)
1
> MemOff 1         'メモリーI/O ビット 1 をオフ
> Print MemSw(1)
0
```

MemOn

メモリーI/O の指定ビットをオンします。

書式

MemOn {ビット番号 | メモリーIO ラベル}

パラメーター

ビット番号 メモリーI/O のビットを整数で指定します。

メモリーIO ラベル メモリーI/O のラベルを指定します。

解説

MemOn は、指定したビットをオン (1) にします。MemSw 命令で、指定したメモリービットの状態チェックを行います。Wait 命令もメモリービットに使われ、指定した S/W 状態になるまでシステムを待たせます。

注意

メモリーI/O のオフ

メモリーI/O は、コントローラー再起動時にオフされます。非常停止, 安全扉開, プログラム終了, Reset, EPSON RC+ 再起動では、オフされません。

参照

In, MemIn, InBCD, Off, MemOff, On, OpBCD, Oport, Out, MemOut, Sw, MemSw, Wait

MemOn 使用例

下記は、それぞれが動作命令を起動する 2 つのタスクを使った例です。これら 2 つのタスクには、お互いに相手がロボット動作命令を使い終わったところで順番に制御をゲインするよう、インターロック機能が働いています。そのため、2 つのタスクが順番にそれぞれ指示どおりの動作ステートメントを実行することができます。MemSw は Wait 命令とともに用いて、安全のため、再度動作を始める前にメモリーI/O ビット 1 が適切な値になるのを待ちます。MemOn と MemOff は、正確に同期するように、メモリーI/O をオン、またはオフします。

タスク同期には、Signal と Waitsig 命令を使うこともできます。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For i = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
  Integer I
  For i = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memon 1  
> print memsw(1)  
1  
> memoff 1  
> print memsw(1)  
0
```

MemOut

8 個のメモリーI/O ビットを同時に設定します。

書式

MemOut ポート番号, 出力データ

パラメーター

ポート番号 メモリーI/O のバイトポート番号を指定します。ポート番号は、下記のようにビットに対応します。

ポート番号	出力データ
0	0-7
1	8-15
.	.

出力データ ポート番号で指定した出力グループの出力パターンを、0~255 の整数値で指定します。16 進数で表示された場合は、&H0~&HFF が範囲になります。下位桁は、下位ビット (または最初の 4 出力ビット) を、上位桁は、上位ビット (または次の 4 出力ビット) を表わします。

解説

MemOut は、設定する出力ビットを指示するポート番号と出力データの組み合わせで、8 個のメモリーI/O ビットを同時に設定します。ポート番号パラメーターで、どのグループ (どの 8 個の出力ビット) を使うか指定します。例えば、ポート番号 = 0 のとき、出力ビット 0~7 を指定します。ポート番号 = 1 のとき、出力ビット 8~15 を指定します。

まずポート番号で 8 個の出力ビットを指定したのち、出力データパラメーターで特定の出力パターンを定義します。出力データパラメーターがとり得る値は 0~255 で、16 進数あるいは 10 進数の整数で指定します。(&H0~&HFF または 0~255)

下記表は、I/O の組み合わせ例の一部を、ポート番号が“0”および“1”のときに分けて、それに対応する出力データ値とともに示したものです。

ポート番号=0 の時の出力設定 (出力ビット番号)

出力データ	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

ポート番号=0 の時の出力設定 (出力ビット番号)

出力データ	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

参照

In, MemIn, InBCD, Off, MemOff, On, MemOn, OpBCD, Oport, Out, Sw, MemSw, Wait

MemOut 使用例

下記は、メインタスクが“iotask”という名前のタスクを起動するプログラムです。“iotask”は、S/W メモリーI/O ビットの 0~3 をオンオフする簡単なタスクです。MemOut 命令を使うと、S/W メモリーI/O ビットを個々にオンオフせず、1 つのコマンドでそれを行うことができます。

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
Fend

Function iotask

  Do
    MemOut 0, &H

    Wait 1
    MemOut 0, &H0
    Wait 1
  Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> MemOut 1,6      'メモリーI/O ビット 9 と 10 をオン
> MemOut 2,1      'メモリーI/O ビット 8 をオン
> MemOut 3,91     'メモリーI/O ビット 24, 25, 27, 28, 30 をオン
```


MemOutW

メモリーI/O ポートの状態を 16 ビット同時にワード単位で設定します。

書式

MemOutW ワードポート番号, 出力データ

パラメーター

ワードポート番号 メモリーI/O ワードを指定します。

出力データ メモリーI/O データ (0~65535 の整数)を、式または数値で指定します。

解説

パラメーターのワードポート番号で指定したメモリーI/O ポートグループの状態を、指定した出力データに変更します。

参照

MemIn, MemInW, MemOut

MemOutW 使用例

MemOutW 0, 25

MemSw 関数

指定されたメモリーI/Oビットの状態を返します。

書式

MemSw(ビット番号)

パラメーター

ビット番号 メモリーI/Oビット番号を表わす数値で指定します。

戻り値

指定されたビットがオンのとき“1”を、オフのとき“0”を返します。

解説

MemSw は、1 メモリーI/Oビットの状態を返します。MemSw には、通常 MemOn と MemOff 命令がセットで使われます。MemOn は指定されたビットをオンにし、MemOff は指定されたビットをオフします。

参照

In, MemIn, InBCD, Off, MemOff, On, MemOn, OpBCD, Oport, Out, MemOut, Sw, Wait

MemSw 関数使用例

下記例では、2つのタスクが、それぞれが動作命令を起動する機能を持ちながら、一方がロボット制御をしていないときに、一方がロボット動作を制御するようインターロックしています。それにより、各々のタスクが与えられた動作ステートメントを、きちんと想定通りの順序にしたがって実行することができます。MemSw は、Wait 命令とともに使われることで、メモリーI/Oビット1が、次の動作を安全に行える適正な値になるまで待つて、動作を再開します。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For i = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
  Integer I
  For i = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

MHour 関数

ロボットのモーターの積算励磁時間を返します。

書式

MHour ([ロボット番号])

パラメーター

ロボット番号 励磁時間を確認するロボットの番号を整数値で指定します。
省略した場合、現在選択されているロボットが対象となります。

戻り値

モーターの積算励磁時間を実数値で返します。

参照

Time, Hour

MHour 関数使用例

```
Robot 2  
Print MHour  
Print MHour (1)
```

Mid\$ 関数

文字列より指定した開始位置から、指定した文字数の文字を取り出します。

書式

Mid\$(文字列, 開始位置 [, 文字数])

パラメーター

文字列	元になる文字列を指定します。
開始位置	文字列を取り出す開始位置を指定します。
文字数	文字列から取り出す文字数を指定します。省略可能です。省略すると開始位置から文字列の最後まで文字を返します。

戻り値

元となる文字列から取り出した文字列を返します。

解説

Mid\$ は元になる文字列より指定した開始位置から、文字数分の文字列を取り出します。

参照

Asc, Chr\$, InStr, Left\$, Len, Right\$, Space\$, Str\$, Val

Mid\$ 関数使用例

下記の例は、文字列“ABCDEFGHIJ”の真中から 2 文字と、開始位置 5 の残りの文字列を取り出します。

```
Function midtest
    String basestr$, m1$, m2$
    basestr$ = "ABCDEFGHIJ"
    m1$ = Mid$(basestr$, (Len(basestr$) / 2), 2)
    Print "The middle 2 characters are: ", m1$
    m2$ = Mid$(basestr$, 5)
    Print "The string starting at 5 is: ", m2$
End
```

MkDir

サブディレクトリーを作成します。

書式

MkDir ディレクトリー名

パラメーター

ディレクトリー名 作成するサブディレクトリーのパスと名称の文字列を指定します。
パスの詳細は、ChDisk を参照してください。

解説

指定したパス上に、新しくディレクトリーを作成します。パスを省略した場合は、カレントディレクトリー内にサブディレクトリーを作成します。

注意

- ディスクが PC の場合に実行可能です。

参照

ChDir, ChDrive, RenDir, Rmdir

Mkdir 使用例

コマンドウィンドウでの操作例

```
> Mkdir \Data  
  
> Mkdir \Data\PTS  
  
> Mkdir \TEST1 \TEST2
```

Mod 演算子

数値式を他の数値式で割った残り値を返します。

書式

被除数 Mod 除数

パラメーター

被除数 割られる数を指定します。

除数 割る数を指定します。

結果

被除数を除数で割った残り値を返します。

解説

Mod は、2 つの数値を割った残り値 (整数)を得るときに使います。Mod 命令の便利な使い方としては、偶数か奇数かを調べるときなどです。

Mod 命令の手順は、次のとおりです

:被除数を除数で割り算します。この割り算の残りが、Mod 命令の戻り値として返されます。

参照

Abs, Atan, Atan2, Cos, Int, Not, Sgn, Sin, Sqr, Str\$, Tan, Val

Mod 演算子使用例

下記の例では、ある数値 (var1)が偶数か奇数か調べます。その数値が偶数ならば、Mod 命令は、結果“0”を返します。奇数ならば、結果“1”を返します。

```
Function modtest
  Integer var1, result

  Print "Enter an integer number:"
  Input var1
  result = var1 Mod 2
  Print "Result = ", result
  If result = 0 Then
    Print "The number is EVEN"
  Else
    Print "The number is ODD"
  EndIf
Fend
```

その他の使用例として、コマンドウィンドウからの操作例をあげます。

```
> Print 36 Mod 6
> 0

> Print 25 Mod 10
> 5
>
```

Motor

ロボットの全軸のモーターをオン、またはオフします。

書式

Motor On | Off

パラメーター

On | Off モーターパワーをオン状態に設定する場合には On を、オフ状態にする場合には Off を指定します。

解説

Motor On 命令はモーターパワーをオンに設定し、全軸のブレーキを解除します。Motor Off 命令はモーターパワーをオフに設定し、全軸のブレーキを設定します。

ロボットを動かすためには、モーターパワーがオンでなければなりません。

非常停止後、または Reset コマンドを必要とするエラーが発生した後は、Reset コマンドを実行してから Motor On します。

Motor On 命令を実行すると、ロボット制御パラメーターを下記の設定にします。

ロボット制御パラメーター

Speed と SpeedR, SpeedS の設定値	(初期値に初期化されます。)
Accel と AccelR, AccelS の設定値	(初期値に初期化されます。)
QPDcelR, QPDcelS の設定値	(初期値に初期化されます。)
LimZ パラメーターの設定値	(0 に初期化されます。)
CP パラメーターの設定値	(Off に初期化されます。)
SoftCP パラメーターの設定値	(Off に初期化されます。)
Fine の設定	(初期値に初期化されます。)
Power Low 設定	(ローパワーモードになります。)
PTPBoost の設定値	(初期値に初期化されます。)
TCLim, TCSpeed の設定値	(初期値に初期化されます。)
PgLSpeed の設定値	(初期値に初期化されます。)
PerformMode の設定値	(標準モードに初期化されます。)

参照

Brake, Power, Reset, SFree, SLock

Motor 使用例

次の例はコマンドウィンドウから実行します。

```
> Motor On
```

```
> Motor Off
```

Motor 関数

指定したロボットのモーターパワーの状態を返します。

書式

Motor [(ロボット番号)]

パラメーター

ロボット番号 状態を確認するロボット番号を整数値で指定します。
省略した場合、現在選択されているロボットが対象となります。

戻り値

0 = モーターオフ
1 = モーターオン

参照

Motor

Motor 関数使用例

```
If Motor = Off Then  
    Motor On  
EndIf
```


Move

アームを現在位置から指定位置まで直線補間動作で動かします。

書式

Move 目標座標 [ROT] [ECP] [CP] [Till | Find] [!並列処理!] [SYNC]

パラメーター

目標座標	ポイントデータで目標位置を指定します。
ROT	ツール姿勢変化を優先して、動作の速度、加減速度を決めます。省略可能です。
ECP	外部制御点動作を指定します。省略可能です。(ECP オプション使用時のみ有効)
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。

Till | Find

Till Sw(式) = {On | Off}

Find Sw(式) = {On | Off}

! 並列処理! 動作中に I/O などのコマンドを実行できるように、並列処理ステートメントを追加できます。省略可能です。

SYNC 動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

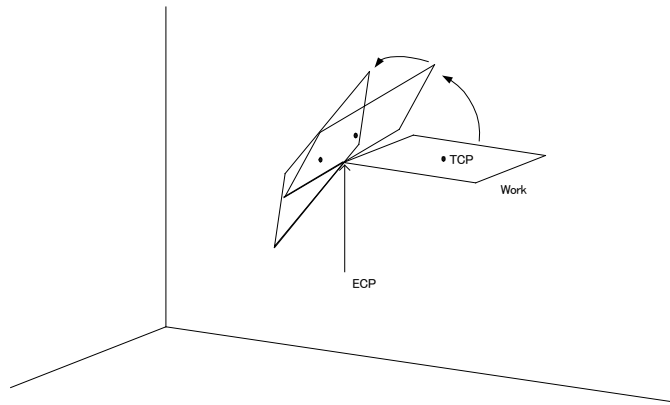
Move はアームを現在位置から目標座標まで直線移動させます。Move では全関節が同時に動き始め、同時に停止します。目標座標は、Move 命令の実行前にあらかじめティーチングしておきます。Move の加減速は、AccelS 命令で制御します。Move の速度は、SpeedS 命令で制御します。SpeedS で設定した値が、それぞれの関節の許容速度を 1 つでも超えた場合、モーターの励磁を切ってロボットは停止します。

Move の速度と加減速度はそれぞれ SpeedS と AccelS の設定値を使用します。速度と加減速度の関係については、“注意”の“Move を CP とともに用いる”を参照してください。ただし、ROT 修飾パラメーターを指定しているときの速度と加減速度は、それぞれ SpeedR と AccelR の設定値を使用します。その場合、SpeedS と AccelS の設定値は無効となります。

通常、移動距離が“0”で、姿勢関節のみが動作するような場合にはエラーとなります。ROT 修飾パラメーターを付けてツール姿勢変化の加減速を優先することで、エラーなく動作が可能になります。ROT 修飾パラメーターを付けたとき、姿勢変化がなく、移動距離が“0”でない場合には、エラーとなります。

また、移動距離に対してツール姿勢変化速度が大きすぎる場合や、指定された回転速度がマニピュレーターの限度を越えている場合もエラーとなります。その場合は指定速度を落とすか、または ROT 修飾パラメーターをつけて姿勢変化の加減速度を優先するようにしてください。

ECP を使っている場合、指定した ECP 番号に対応する外部制御点で、ワークが直線に動きます。このとき先端関節の中心は、直線をたどりません。



Move 動作完了以前にロボットを減速停止させたい場合は、ユーザーはその条件を、Till 修飾子を使って指定できます。ここで指定した条件は、入力ビットのうち 1 つをチェックすることになり、Sw 命令を使って行います。ユーザーは、入力がオンかオフかをチェックし、指定した条件に基づいてアームを停止させます。この機能は、入力条件が成立すると Move が中止される割込みに似ています。Move 動作中一度も入力条件が成立しなければ、目標座標で指定された位置までアームは到達します。Till 修飾子は省略可能です。Till 修飾子の詳細については、Till コマンドを参照してください。

注意

Move でできないこと

動作を行う前に動作範囲確認をすることはできません。したがって目標座標位置が許容動作範囲内にあっても、そこに至る軌道が許容動作範囲外を通過すると、アームが突然停止して、サーボに衝撃を与え、障害を生じる危険があります。これを防ぐために、Move を高速度で実行する前に、低速度で動作範囲確認を行ってください。つまり、目標座標がアームの動作範囲内にあっても、Move 動作でそこに至る軌道が、物理的にアームの許容動作範囲外にあると、アームは動かなくなります。

Move を CP とともに用いる

CP パラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。CP 指定なしの Move 命令では、アームは必ず減速して、指定された目標座標に停止します。

Move に適切な速度、加減速度指示を与える

SpeedS と AccelS 命令は、Move 動作中のマニピュレーターの速度と加減速度を指定しますが、SpeedS と AccelS は、直線および円弧補間動作に対する命令であることに注意してください。PTP 動作に対しては、Speed と Accel 命令が適用になります。

起こりやすいエラー

直線移動距離が 0 となる動作を実行しようとした場合

Move は、4 自由度ロボット(水平多関節型 (RS シリーズを含む)など)の U 座標値や 6 自由度ロボット (垂直 6 軸型 (N シリーズを含む))の U, V, W 座標値のみを変化させるような動作を行おうとするとエラーが発生します。この場合は ROT パラメーターを使用してください。

関節の制限スピードを超えるエラー

指示された動作中、1 関節でも許容速度を超えると、オーバースピードエラーが発生します。モーターオーバースピードエラーの場合、アームは停止し、モーターの励磁は切れます。

RS シリーズで原点近傍を通過する動作を実行した場合

RS シリーズでは、Move で原点近傍を通過するような動作を行うと、オーバースピードエラーが発生することがあります。原点近傍を通過するような動作については、以下のような対策を行ってください。

- SpeedS の設定速度を下げてください。
- 原点近傍を通過しない経路に変更してください。
- Move の代わりに Go などの PTP 動作を使用してください。

参照

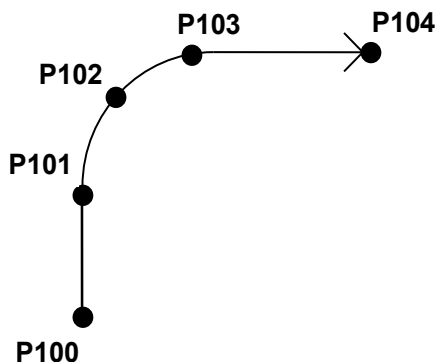
AccelS, Arc, CP, Go, Jump, Jump3, Jump3CP, SpeedS, Sw, Till

Move 使用例

下記の例は、P0~P1 までアームを PTP 動作し、その後、直線で P0 まで戻す簡単な例です。プログラム後半で、アームは、入力ビット 2 がオンになるまで P2 に向かって直線移動します。動作中に入力ビット 2 がオンになると、アームは P2 に到達前でも減速停止し、次のプログラム命令を実行します。

```
Function movetest
  Home
  Go P0
  Go P1
  Move P0
  Move P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  EndIf
Fend
```

次に Move を CP とともに用いる例です。下記図は、P100 から始まる円弧軌道を示します。P100 から P101 まで直線移動し、P101 から円弧が始まり P102 を通って P103 に円弧を描き続けます。P103 から P104 までは直線動作になり、この間に減速停止します。P104 で停止するまで、途中のポイントでは減速しないことに注目してください。下記のファンクションでは、このような動作をプログラムしています。



```
Function CornerArc
  Go P100
  Move P101 CP      'P101 で停止しない
  Arc P102, P103 CP 'P103 で停止しない
  Move P104        'P104 に減速停止
Fend
```

MsgBox

ダイアログボックスのメッセージを表示し、ユーザーがボタンを選択するのを待ちます。

書式

MsgBox メッセージ\$ [, ボタンの種類] [, タイトル\$] [, 結果を受け取る文字列変数]

パラメーター

メッセージ\$ 表示されるメッセージ

ボタンの種類 表示するボタンの数と型, アイコンのスタイル, ボタンの見出しなどを指定する値の合計を表わす数値、または式を指定します。EPSON RC+には、このパラメーター用にあらかじめ決められた定数があり、下記表の値を用います。省略可能です。

定数	値	意味
MB_OK	0	<OK>ボタンだけを表示
MB_OKCANCEL	1	<OK>と<キャンセルボタン>を表示
MB_ABORTRETRYIGNORE	2	<中止><再試行><無視>ボタンを表示
MB_YESNOCANCEL	3	<はい><いいえ><キャンセルボタン>を表示
MB_YESNO	4	<はい><いいえ>ボタンを表示
MB_RETRYCANCEL	5	<再試行>と<キャンセルボタン>を表示
MB_ICONSTOP	16	Stop サイン
MB_ICONQUESTION	32	“?”マーク
MB_ICONEXCLAMATION	64	“!”マーク
MB_DEFBUTTON1	0	1 番目のボタンはデフォルト
MB_DEFBUTTON2	256	2 番目のボタンはデフォルト

タイトル\$ ダイアログボックスのタイトルバーに表示される文を指定します。省略可能です。

結果を受け取る文字列変数

ユーザーの選択を表わす値 (整数)を受け取る変数を指定します。EPSON RC+には、このパラメーター用にあらかじめ決められた定数があります。このパラメーターで返される値は、下記表のとおりです。省略可能です。

定数	値	意味
IDOK	1	<OK>ボタンが選択された。
IDCANCEL	2	<キャンセル>ボタンが選択された。
IDABORT	3	<中止>ボタンが選択された。
IDRETRY	4	<再試行>ボタンが選択された。
IDYES	6	<はい>ボタンが選択された。
IDNO	7	<いいえ>ボタンが選択された。

解説

MsgBox は、自動的にメッセージをフォーマットします。白紙状態にしたい場合は、メッセージに CRLF を用います。下記例を参照してください。

参照

InputBox

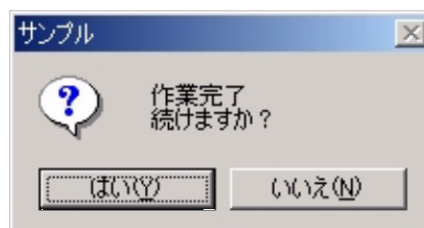
MsgBox 使用例

下記の例では、ユーザーに、作業を継続するか、しないかを確認するメッセージボックスを表示します。メッセージボックスは、<はい>と<いいえ>の 2 つのボタンを表示します。“?”マークアイコンも表示します。ユーザーがボタンを選択して、MsgBox が返ってくると、その答えを検討します。<いいえ>が答えだと、Quit コマンドですべてのタスクを終了します。

```
Function msgtest
    String msg$, title$
    Integer mFlags, answer

    msg$ = Chr$(34) + "作業完了" + Chr$(34) + CRLF
    msg$ = msg$ + "続けますか?"
    title$ = "サンプル"
    mFlags = MB_YESNO + MB_ICONQUESTION
    MsgBox msg$, mFlags, title$, answer
    If answer = IDNO then
        Quit All
    EndIf
End
```

上記のコードで作られるメッセージボックスは、下記画面のようになります。



制限事項

パラメーターのメッセージ\$、およびタイトル\$ に半角カンマ“,”が含まれる場合、文字列が正しく表示できません。半角カンマを含まない文字列をご使用ください。

MyTask 関数

現在のプログラムのタスク番号を返します。

書式

MyTask

戻り値

現在のタスクのタスク番号を返します。タスク番号の範囲は以下の整数値です。

通常タスク	: 1 から 32
バックグラウンドタスク	: 65 から 80
トラップタスク	: 257 から 267

解説

MyTask は現在のプログラムのタスク番号を数値で返します。MyTask 関数をプログラムに記述し実行した場合、プログラムを実行しているタスク番号を返します。

参照

Xqt

MyTask 関数使用例

次の例は 1 から 8 の I/O ポートをオン、またはオフします。

```
Function main
  Xqt 2, task      'タスク 2 を実行
  Xqt 3, task      'タスク 3 を実行
  Xqt 4, task      'タスク 4 を実行
  Xqt 5, task      'タスク 5 を実行
  Xqt 6, task      'タスク 6 を実行
  Xqt 7, task      'タスク 7 を実行
  Xqt 8, task      'タスク 8 を実行
  Call task
Fend

Function task
  Do
    On MyTask      '現在のタスク番号と同じ番号の I/O ポートをオン
    Off MyTask     '現在のタスク番号と同じ番号の I/O ポートをオフ
  Loop
Fend
```

Next

For と Next 命令は、組み合わせて用いてループを作成します。For と Next の間にある一連の命令は、ユーザーが指定する回数だけ繰り返します。

書式

```
For 変数名 1 = 初期値 To 終了値 [Step 増分値 ]
    ステートメント
Next 変数名 1
```

パラメーター

変数名 1	For/Next ループで、繰り返しデータを代入するカウンター変数名を指定します。通常は整数変数として定義されますが、実数を指定することもできます。
初期値	指定した変数に、ループの最初に代入する数値を、式または直接数値で指定します。
終了値	ループの終了を表わす数値を、式または直接数値で指定します。この値が成立すると For/Next ループは終了し、Next 命令の次のステートメントの実行に移ります。
増分値	For/Next ループ 1 回ごとに変数を増分する値を、式または直接数値で指定します。この値は、正負いずれの値でも指定することができます。しかし負の値を指定するときは、初期値 > 終了値 となるように設定してください。このパラメーターが省略されていると、自動的に“1”を増分値として増減します。省略可能です。
ステートメント	For/Next ループに挿入する SPEL+ のステートメントを記述します。

解説

For/Next は、ループ内にある一連のステートメントを、指定された回数だけ繰り返します。ループは、For ステートメントで始まり、Next ステートメントで終わります。変数でループ内のステートメントが繰り返された回数を数えます。

初期値は、カウンターの最初の数値です。終了値と増分値の設定が正しければ、正負いずれの値でもかまいません。

終了値は、カウンターの最後の数値です。この値に達すると、For/Next ループは終了し、プログラムの制御は、Next 命令の次の命令に移ります。

Next 命令に達するまで、プログラムは、For ステートメントの次のステートメントを実行します。カウンター変数は、増分値で設定した値だけ、増減していきます。増分値の設定がないと、“1”で増減します。

次にカウンター変数の値を終了値と比較します。もしカウンター変数の値が終了値より少ないか、あるいは同じ場合、For ステートメント後の命令の実行に戻ります。カウンター変数の値が終了値より大きければ、For/Next ループの次の実行に移り、Next 命令直後の命令に続きます。

注意

負の値の増分値

増分値が負の値の場合、カウンター変数の値はループが行われるごとに減っていきます。この場合、初期値は終了値よりも必ず大きくしてください。

参照

For

For/Next 使用例

```
Function fornext
  Integer ctr
  For ctr = 1 to 10
    Go Pctr
  Next ctr
  ,
  For ctr = 10 to 1 Step -1
    Go Pctr
  Next ctr
Fend
```


Not 演算子

数値をビット単位で補数を取ります。

書式

Not 演算数

パラメーター

演算数 整数値を指定します。

結果

演算数の 1 の補数を返します。

解説

Not 関数は演算数をビット単位で補数を取ります。結果のビットは、対応する演算数のビットを反転します。0 のビットは 1 に、1 ビットは 0 に変換します。

参照

Abs, And, Atan, Atan2, Cos, Int, LShift, Mod, Or, RShift, Sgn, Sin, Sqr, Str\$, Tan, Val, Xor

Not 使用例

コマンドウィンドウで Not を使った例です。

```
>print not(1)
-2
>
```

Off

ビット番号で指定された出力ビットをオフ、または指定時間オフした後オンします。

書式

Off { 出力ビット番号 |出力ラベル } [,時間] [,非同期指定] [,Forced]

パラメーター

出力ビット番号	オフする I/O 出力ビットを整数値で指定します。
出力ラベル	出力ラベルを指定します。
時間	出力ビットがオフしている時間を、秒で指定します。指定された時間経過後、出力ビットは再びオンします。オフ時間は、0.01 秒以上 10 秒以下で指定してください。省略可能です。
非同期指定	タイマーが設定されると、非同期指定を用いて次のコマンドを実行するタイミングを指定します。省略可能です。 0 - 出力ビットがオフすると同時に、次の命令を実行します。 1 - デフォルト設定。指定時間オフした後オンして、次の命令を実行します。
Forced	省略可能です。通常は省略します。

解説

Off は、指定した出力ビットをオフ (または 0 に) 戻します。

時間パラメーターが指定されると、出力ビット番号で指定された出力ビットは、指定時間オフした後、再びオンします。もしオフ実行前に、すでに出力ビットがオフになっていれば、指定時間経過後にオンします。

非同期指定設定は、指定時間の設定が次のようになっているときに有効です。

- 1: 出力ビットをオフし、指定時間後再度オンして次のコマンドを実行します。(非同期指定の設定は、これがデフォルト設定となっています。このパラメーターを省略すると“1”を設定したことになります。)
- 0: 出力ビットをオフし、同時に次のコマンドを実行します。

注意

リモート制御用に割り当てられた出力ビット

システム出力用に設定された出力ビットを指定するとエラーになります。リモート出力ビットは、システムの状態に応じて自動的にオンオフします。

非常停止が発生したとき

EPSON RC+ では、非常停止が発生すると、すべての出力ビットがオフします。非常停止の場合でも設定を保持したい場合には、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートを OFF]チェックボックスで設定し直すことができます。

Forced フラグ

NoPause タスク、NoEmgAbort タスク(Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)から、非常停止中や安全扉開時に I/O 出力のオフを行う場合にこのフラグを指定します。非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InBCD, MemOn, MemOff, MemOut, MemSw, OpBCD, Oport, Out, Wait

Off 使用例

下記例では、メインタスクが、“iotask”という名前のタスクを起動させます。“iotask”は、出力ビット 1 と 2 を別々にオン、またはオフし、10 秒後に再度実行する簡単なタスクです。

```
Function main
  Xgt 2, iotask
  Go P1
  .
  .
  .
Fend
```

```
Function iotask
  Do
    On 1
    On 2
    Off 1
    Off 2
    Wait 10
  Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> on 1
> off 1, 10      '出力 1 をオフし、10 秒後再度オン
> on 2
> off 2
```

OLAccel

過負荷率に応じた加減速度の自動調整を設定します。

書式

OLAccel {On | Off}

パラメーター

On | Off On: 過負荷率に応じた加減速度の自動調整を有効にします。
 Off: 過負荷率に応じた加減速度の自動調整を解除します。

解説

OLAccel は、ロボットの過負荷率 (OLRate) に応じた加減速度の自動調整機能を有効にするかどうかの設定をします。OLAccel を On すると、PTP 動作命令の実行時に、その時のロボット過負荷率に応じて加減速度が自動調整されます。つまり、PTP 動作実行時にロボットの過負荷率が一定値を越えているとき、動作時の加減速度を自動的に落としていき、過負荷エラーの発生を未然に防ぐ働きをします。従来、過負荷エラーが発生するような高デューティーな動作に対しては、お客さまがプログラミングによってロボットを停止させたり、速度や加速度を調節したりする必要がありましたが、OLAccel を使用することによって、その必要性が少なくなります。ただし、あらゆるサイクルで過負荷エラーが発生しなくなるわけではなく、非常に高デューティーで負荷の大きいサイクルでは、過負荷エラーの発生を必ずしもなくすることはできません。その場合には、お客さまによるロボットの停止や、速度、加減速度の調節が必要となります。また、使用環境によっては、過負荷エラーが発生せずにロボットが動作しつづけることによって、モーターが温度上昇しオーバーヒートエラーが発生することがあります。

適正な負荷の状態では、このコマンドを使用することはありません。

テストサイクル時に、OLRate を使って、ロボットが過負荷エラーの起きやすい状態にあるかどうか、チェックできます。

下記のいずれかの場合、OLAccel 値は、初期化されます。

コントローラー起動時 Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

注意

加減速度の自動調整機能をサポートしないロボットに対して OLAccel On を実行すると、エラーが発生します。

参照

OLAccel 関数, OLRate

OLAccel 使用例

```
>olaccel on
>olaccel
OLACCEL is ON

Function main
  Motor On
  Power High
  Speed 100
  Accel 100, 100
  OLAccel On
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Do
    'OLRateを表示
    OLRate
    Wait 1
  Loop
Fend
```

OLAccel 関数

過負荷率に応じた加減速度の自動調整の状態を返します。

書式

OLAccel

戻り値

Off = 過負荷率に応じた加減速度の自動調整が解除

On = 過負荷率に応じた加減速度の自動調整が有効

参照

OLAccel, OLRate

OLAccel 関数使用例

```
If OLAccel = Off Then  
    Print "OLAccel is off"  
EndIf
```

OLRate

指定関節の過負荷率を表示します。

書式

OLRate [関節番号]

パラメーター

関節番号 1~9 の整数で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

解説

OLRate は、指定された関節の過負荷率を表示する関数です。OLRate は、サイクルが関節に過剰な負担をかけていないかチェックします。負荷の多いサイクルで使用すると温度や電流がサーボエラーの原因となることがあります。OLRate で、ロボットがサーボエラーの起きやすい状態にあるかどうか、チェックします。

サイクル運転中、OLRate をモニターする別のタスクを実行します。もし、OLRate が 1.0 を超える関節があれば、サーボエラーが起きます。

サーボエラーは、負荷が過剰になったときに最も発生しやすくなります。テストサイクル時に OLRate を使って、速度や加減速度設定を確認し、実使用時にサーボエラーが発生しないよう予防的措置をとることができます。

有効値を得るために、OLRate はロボットの動作中に実行してください。

適正な負荷の状態では、このコマンドを使用することはありません。

参照

OLRate 関数

OLRate 使用例

```
>olrate
0.10000  0.20000
0.30000  0.40000
0.50000  0.60000

Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
Do
  Jump P0
  Jump P1
Loop
Fend

Function MonitorOLRate
Do
  OLRate      'OLRate を表示
  Wait 1
Loop
Fend
```

OLRate 関数

指定関節の過負荷率を返します。

書式

OLRate(関節番号)

パラメーター

関節番号 1~9 の整数で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

指定された関節の過負荷率を返します。戻り値は、0.0~2.0 です。

解説

OLRate は、サイクルが関節に過剰な負担をかけていないかチェックします。負荷の多いサイクルで使用すると温度や電流がサーボエラーの原因となることがあります。OLRate で、ロボットがサーボエラーの起きやすい状態にあるかどうか、チェックします。

サイクル運転中、OLRate をモニターする別のタスクを実行します。もし、OLRate が 1.0 を超える関節があれば、サーボエラーが起きます。

サーボエラーは、負荷が過剰になったときに最も発生しやすくなります。テストサイクル時に OLRate を使って、速度や加減速度設定を確認し、実使用時にサーボエラーが発生しないよう予防的措置をとることができます。

有効値を得るために、OLRate はロボットの動作中に実行してください。

適正な負荷の状態では、このコマンドを使用することはありません。

参照

OLRate

OLRate 関数使用例

```
Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Integer i
  Real olRates(4)
  Do
    For i = 1 to 4
      olRates(i) = OLRate(i)
      If olRate(i) > .5 Then
        Print "Warning: OLRate(", i, ") is over .5"
      EndIf
    Next i
  Loop
Fend
```

On

指定された出力ビットをオンし、指定時間後、オフします。

書式

On { 出力ビット番号 | 出力ラベル } [, 時間] [, 非同期指定] [, Forced]

パラメーター

出力ビット番号	オンする I/O 出力ビットを整数で指定します。
出力ラベル	出力ラベルを指定します。
時間	指定した出力ビットがオンしている時間を秒で指定します。この時間が経過すると、指定出力ビットはオフします。(経過時間は、0.01 秒以上を指定してください。)省略可能です。
非同期指定	時間設定を行ったとき、非同期指定で次のコマンドの実行タイミングを指定することができます。省略可能です。 0 - 出力ビットがオンすると同時に、次の命令を実行します。この場合時間設定の最大は 10 秒です。 1 - デフォルト設定。指定時間オンした後オフして、次の命令を実行します。
Forced	省略可能です。通常は省略します。

解説

On は、指定した出力ビットをオン (1 に設定) します。

時間パラメーターが設定されていると指定された出力ビットがオンし、指定時間経過後オフします。

非同期指定パラメーターの設定は、時間を設定したときに次のように用います。

- 1: 出力ビットをオンし、指定時間経過後オフして次のコマンドを実行します。(非同期指定の設定は、これがデフォルト設定となっています。このパラメーターを省略すると“1”を設定したことになります。)
- 0: 出力ビットをオンし、同時に次のコマンドを実行します。

注意

リモートに設定された出力ビット

リモートに設定された出力ビットを指定するとエラーになります。リモート出力ビットはシステムの状態に応じて自動的にオンしたりオフしたりするようになっています。リモートに関する詳細は、EPSON RC+ ユーザーズガイドを参照してください。リモートコネクターの個々のビットを、出力や I/O に設定するのは、EPSON RC+ の[セットアップ]メニューから[システム設定]-[コントローラー]-[リモート]パネルから行います。

非常停止が発生したとき

ロボットコントローラーでは、非常停止が発生すると、すべての出力ビットがオフします。非常停止の場合でも設定を保持したい場合には、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートを OFF]チェックボックスで設定し直すことができます。

Forced フラグ

NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時に I/O 出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InBCD, MemOff, MemOn, Off, OpBCD, Oport, Out, Wait

On 使用例

下記例では、メインタスクが、“iotask”という名前のタスクを起動させます。“iotask”は、出力ビット 1 と 2 を別々にオン、またはオフし、10 秒後に再度実行する簡単なタスクです。

```
Function main
  Xqt iotask
  Go P1
  .
  .
  .
Fend

Function iotask
  Do
    On 1
    On 2
    Off 1
    Off 2
    Wait 10
  Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> on 1
> off 1, 10      '出力 1 をオフし, 10 秒後再度オン
> on 2
> off 2
```

OnErr

エラー発生時に、エラー処理サブルーチンに制御を枝分かれさせる割込みを設定します。ユーザーがエラー処理できるようにします。

書式

OnErr GoTo {ラベル | 0}

パラメーター

ラベル	エラー発生時に移行する先のステートメントラベルを指定します。
0	OnErr 設定をクリアするパラメーターを指定します。

解説

OnErr によりユーザーはエラー処理を行うことができます。OnErr が使われていないと、エラー発生時にタスクは中止され、エラーが表示されます。しかし、OnErr を使うと、自動的にエラーから回復できるように、制御をエラー処理サブルーチンに移すことができます。エラーが回復すると、制御は EResume 命令で指定する戻り先に移ります。これにより、エラーが発生してもタスクの実行を中断することなく、自動的にエラー処理を行うことができます。また、複雑になりやすい問題も、常に同じやり方で自動的に処理されるので、ワークセルを格段にスムーズに働かせることができます。

OnErr コマンドがパラメーター“0”を指定して設定された場合は、現在の OnErr 設定はクリアされます。(OnErr 0 実行後、もしエラーが発生すると、プログラムの実行は停止されます。)

参照

Err, EResume

OnErr 使用例

下記は、簡単なユーティリティープログラムの例で、ポイントデータ P0-P399 が存在するか、存在しないかをチェックします。ポイントデータが存在しないと、画面に“このポイントは存在しません”というメッセージが表示されます。このプログラムでは、CX 命令を使って、ポイントが定義されているか、定義されていないかをテストします。もし定義されていなければ、制御はエラー処理サブルーチンに移され、画面に定義されていないポイントが表示されます。

```
Function errDemo
    Integer i, errNum

    OnErr GoTo errHandler

    For i = 0 To 399
        temp = CX(P(i))
    Next i
    Exit Function
'
'
'*****
'* Error Handler                                     *
'*****
errHandler:
    errNum = Err
    '未定義ポイントを使用していないか確認
    If errNum = 7007 Then
        Print "Point number P", i, " is undefined!"
    Else
        Print "ERROR: Error number ", errNum, " occurred while"
        Print "          trying to process point P", i, " !"
    EndIf
    EResume Next
Fend
```

OpBCD

2 進化 10 進数 (BCD) で 8 ビットの出力を同時に設定します。

書式

OpBCD ポート番号, 出力データ [, Forced]

パラメーター

ポート番号 I/O の出力バイトを指定します。指定数値は、それぞれ下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
2	16-23
3	24-31
...	...

出力データ ポート番号で指定した出力グループの出力パターンを、0 から 99 の整数値で指定します。2 番目の桁 (1 の位) は、選択されたグループのうち下位 4 出力ビットを、1 番目の桁 (10 の位) は、上位 4 出力ビットを表わします。

Forced 省略可能です。通常は省略します。

解説

OpBCD は、8 個の出力ビットを BCD で同時に設定します。標準および拡張出力ビットは、8 個ずつ 1 組にグループ化されています。OpBCD 命令のポート番号パラメーターで、どのグループ (どの 8 個の出力ビット) を使うか指定します。例えば、ポート番号 = 0 のとき出力ビット 0 から 7、ポート番号 = 1 のとき出力ビット 8 から 15 を指定します。

ポート番号 (8 個の出力ビット) を選択したのち、出力データパラメーターで特定の出力パターンを定義します。出力データパラメーターは、1 または 2 桁の値で、有効範囲は 0 から 99 です。1 番目の桁 (10 の位) は、ポート番号で選択した 8 出力ビットのうち出力の上位 4 ビットを、2 番目の桁 (1 の位) は、ポート番号で選択した 8 出力ビットのうち出力の下位 4 ビットを表わします。

BCD では各桁の有効数値が 0 から 9 までのので、すべての I/O の組み合わせを満たすことができません。下記表は、ポート番号=0 のときの I/O の組み合わせ例と、それに対応する出力番号値を示します。

出力設定 (出力番号)

出力番号	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

BCD では、10 進数値でしか指定することができません。そのため、BCD を用いる OpBCD 命令ではすべての出力ビットをオンすることはできないことになります。出力番号値の最大値は、各桁ともに“9”なので、OpBCD で使える最大の値は、“99”になります。上記の表を見ると、“99”ではすべての出力をオンすることができないことがわかります。出力番号値が“99”のときの出力状態は、0, 3, 4, 7 がオンで、他はすべてオフになっています。

注意

OpBCD と Out の違い

SPEL+ の Out と OpBCD は、下記の点で大きく異なります。

- OpBCD 命令は、オン、またはオフする 8 個の出力ビットを指定するのに 2 進法 10 進 (BCD) フォーマットを使います。2 進法 10 進フォーマットでは、&HA, &HB, &HC, &HD, &HE, &HF などの値は使えないので、8 個の出力ビットの組み合わせすべてを満たすことができません。
- Out 命令は、出力をオン、またはオフする 8 ビット値に 0 から 255 の値を使います。(OpBCD では 0 から 99 です。)これにより、8 ビット出力グループの可能な組み合わせすべてを満たせることになり、ユーザーの仕様に合わせて指定することができます。

リモートに設定された出力ビット

リモートに設定されている出力ビットに対し、OpBCD がオンするよう指定するとエラーになります。リモートの出力ビットは、システムの状態に応じて自動的にオン、またはオフしています。リモートについての詳細は、EPSON RC+ ユーザーズガイドを参照してください。[セットアップ]メニューから[システム設定]-[コントローラー]-[リモート]パネルからリモートコネクタの個々のビットの設定(リモートまたは I/O に設定するなど)を行うことができます。

非常停止時の出力

ロボットコントローラーには、非常停止時にはすべての出力がオフする機能があります。この機能を有効、あるいはオフするには、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートを OFF]チェックボックスの設定を行います。

Forced フラグ

NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時に I/O 出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InBCD, MemOff, MemOn, MemSw, Off, On, Oport, Out, Sw, Wait

OpBCD 使用例

下記は、メインタスクが“iotask”という名前のタスクを起動するプログラムです。“iotask”は、出力ビット 1 と 2 をオンしてから出力ビット 0 と 3 をオンする簡単なタスクです。出力ビット 1 と 2 がオンすると、出力ビット 0 と 3 がオフし、また前者がオフすると後者はオンします。

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
  .
Fend

Function iotask
  Do
    OpBCD 0, 6
    OpBCD 0, 9
    Wait 10
  Loop
Fend
```

下記はコマンドウィンドウからの簡単な操作例です。

```
> OpBCD 1, 6      '出力 1 と 2 をオン
> OpBCD 2, 1      '出力 8 をオン
> OpBCD 3, 91     '出力 24, 28, 31 をオン
```

OpenDB

データベース、Excel ブックをオープンします。

書式

OpenDB #データベース番号, データベース種別 [, SQL サーバー名], データベース名

パラメーター

データベース番号	501 から 508 の整数を指定します。
データベース種別	オープンするデータベースの種別を[SQL], [Access], [Excel]の中から選択します。
SQL サーバー名	データベース種別に[SQL]を指定した場合、SQL サーバー名を指定します。 省略した場合 (LOCAL)サーバーを指定したことになります。ネットワーク上の SQL サーバーは指定できません。 データベース種別に[Access]または[Excel]を指定した場合、SQL サーバー名は指定しません。
データベース名	データベース種別に[SQL]を指定した場合、SQL サーバー上のデータベース名を指定します。 [Access]を指定した場合、Access ファイル名を指定します。 Access ファイル名のパスを省略した場合、カレントフォルダーを検索します。 詳細は、ChDisk を参照してください。 [Excel]を指定した場合、Excel ファイル名を指定します。 Excel ファイルに指定できる形式は、Excel 2007 ブック、Excel 97-2003 ブックファイルです。Excel ファイル名のパスを省略した場合、カレントフォルダーを検索します。 詳細は、ChDisk を参照してください。

解説

指定したデータベースを指定したファイル番号でオープンします。
指定したデータベースは、RC+がインストールされている PC のディスク上に存在しなければなりません。
存在しない場合、エラーとなります。指定したファイル番号はデータベースをオープンしている間、そのデータベースを識別するために使われ、CloseDB コマンドでデータベースをクローズするまで別のデータベースを参照するために使うことはできません。ファイル番号は、データベース操作命令 (SelectDB, Print#, Input#, CloseDB)で使われます。
Microsoft office 2010 64 bit 版の Access と Excel ファイルは使用できません。

注意

RC+ がインストールされている PC の接続が必要です

参照

SelectDB, CloseDB, UpdateDB, DeleteDB, Input #, Print #

OpenDB 使用例

SQL データベースを使用した例

下記に、SQL サーバー2000 のサンプルデータベース Northwind を使用した、テーブルからデータを読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees")
For i = 0 To count - 1
    Input #501, eid, Lastname$, Firstname$, Title$
    Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
Next
CloseDB #501
```

Access データベースを使用した例

下記に、Microsoft Access 2007 のサンプルデータベース生徒名簿を使用した、テーブルからデータを読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, dummy$

OpenDB #502, Access, "c:\MyDataBase\生徒名簿.accdb"
count = SelectDB(#502, "生徒")
For i = 0 To count - 1
    Input #502, eid, dummy$, dummy$, Lastname$, dummy$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #502
```

Excel ブックを使用した例

下記に、Microsoft Excel ブック生徒名簿を使用した、シートからデータを読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$

OpenDB #503, Excel, "c:\MyDataBase\生徒名簿.xls"
count = SelectDB(#503, "[生徒$]")
For i = 0 To count - 1
    Input #503, eid, Lastname$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #503
```

OpenCom

RS-232C ポートをオープンします。

書式

OpenCom #ポート番号

パラメーター

ポート番号	オープンする RS-232C のポート番号を整数値で指定します。
SPEL+制御部	1 から 8
PC 部	1001 から 1008

解説

RS-232C ポートをタスクから使用可能にします。

SPEL+制御部のポートを使用するには、コントローラーにオプションの I/O ボードを装着する必要があります。

PC 部のポートを使用するには RC+の設定が必要です。EPSON RC+ ユーザーズガイド「5.13 [セットアップ]メニュー」の RC-232C に関する記述を参照してください。

参照

ChkCom, CloseCom, SetCom

OpenCom 使用例

```
Integer PortNo

PortNo = 1001
OpenCom #PortNo
Print #PortNo, "Data from COM1"
CloseCom #PortNo
```

OpenCom 関数

OpenCom を実施しているタスク番号を取得します。

書式

OpenCom(ポート番号)

パラメーター

ポート番号	RS-232C のポート番号を整数値で指定します。
SPEL ⁺ 制御部	1 から 8
PC 部	1001 から 1008

解説

OpenCom を実施しているタスク番号を取得します。

参照

ChkCom, CloseCom, OpenCom, SetCom

OpenCom 関数 使用例

```
Print OpenCom(PortNo)
```

OpenNet

TCP/IP ネットワークポートをオープンします。

書式

OpenNet #ポート番号 As { Client | Server }

パラメーター

ポート番号 オープンする TCP/IP のポート番号の整数値を指定します。
ポート番号の範囲は 201~216 です。

解説

OpenNet はネットワーク上の他のコンピューターと通信するために、TCP/IP ポートをオープンします。

1 つのシステムはサーバーとして、他のシステムはクライアントとしてオープンします。どちらが先に起動してもかまいません。

参照

ChkNet, CloseNet, SetNet

OpenNet 使用例

2 つのコントローラーの TCP/IP 設定が、下記のようにになっている例です。

Controller #1:

Port: #201
Host Name: 192.168.0.2
TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
End
```

Controller #2:

Port: #201
Host Name: 192.168.0.1
TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
End
```

OpenNet 関数

OpenNet を実施しているタスク番号を取得します。

書式

OpenNet(ポート番号)

パラメーター

ポート番号	TCP/IP のポート番号の整数値を指定します。 ポート番号の範囲は 201~216 です。

解説

OpenNet を実施しているタスク番号を取得します。

参照

ChkNet, CloseNet, OpenNet, SetNet

OpenNet 関数 使用例

```
Print OpenNet (PortNo)
```

Oport 関数

指定された出力ビットの状態を返す関数です。

書式

Oport(出力ビット番号)

パラメーター

出力ビット番号 I/O の出力ビットを指定します。

戻り値

I/O の指定出力ビットの状態を 0 または 1 の整数で返します。

0: オフ

1: オン

解説

Oport は、出力ビットの状態チェックを行います。入力ビットに対する Sw 命令の機能とよく似た働きをします。Oport が最もよく使われるのは、フィーダーやコンベヤーやグリッパーなどです。また I/O を通じて機能するその他のデバイスのホストなどに接続されている出力ビットの状態チェックにも使われます。Oport 関数でチェックされる状態は、1 または 0 の値で返されます。これらの値は、指定された出力ビットがオンかオフかを示します。

注意

Oport と Sw の違い

Oport と Sw 命令には、重要な違いがあります。両者とも I/O 状態のチェックに使われますが、チェックする I/O のタイプが異なります。Sw 命令では入力ビットのチェックを行います。Oport 命令は、標準および拡張ハードウェアの出力ビットのチェックを行います。これらのハードウェアポートは、コントローラー外部のデバイスと交信する独立した出力ビットです。

参照

In, InBCD, MemIn, MemOff, MemOn, MemOut, MemSw, Off, On, OpBCD, Out, Sw, Wait

Oport 関数使用例

下記は、出力ビット 5 をオンし、それがオンしているかチェックしてからプログラムを継続する例です。

```
Function main
  TMOut 10
  OnErr errchk
  Integer errnum
  On 5      '出力 5 をオン
  Wait Oport(5)
  Call mkpart1
  Exit Function

errchk:
  errnum = Err(0)
  If errnum = 94 Then
    Print "TIME Out Error Occurred during period"
    Print "waiting for Oport to come on. Check"
    Print "Output #5 for proper operation. Then"
    Print "restart this program."
  Else
    Print "ERROR number ", errnum, "Occurred"
    Print "Program stopped due to errors!"
  EndIf
  Exit Function
Fend
```

コマンドウィンドウからの簡単な操作例です。

```
> On 1
> Print Oport(1)
1
> Off 1
> Print Oport(1)
0
>
```


Or 演算子

ビット単位または論理で、2つの値を Or 演算します。

書式

値 1 Or 値 2

パラメーター

値 1 整数値または Boolean 値を指定します。

値 2 整数値または Boolean 値を指定します。

結果

整数値で指定した場合は、値をビット単位で Or 演算した結果を返します。Boolean 値で指定した場合は、論理 Or 演算の結果を返します。

解説

整数値の場合、Or 演算子は、演算数をビット単位で Or 演算します。演算結果は、2つの演算数の各ビットを Or した値です。

Boolean 値の場合、値のどちらか一方が真 (True)であれば、演算結果は真 (True)になります。

参照

And, LShift, Mod, Not, RShift, Xor

Or 演算子使用例

次の例はビット単位 Or 演算子の場合です。

```
>print 1 Or 2
3
>
```

次の例は論理 Or 演算子の場合です。

```
If a = 1 Or b = 2 Then
  c = 3
EndIf
```

Out

8 個の出力ビットを同時に設定 (出力) します。

書式

Out ポート番号, 出力データ [, Forced]

パラメーター

ポート番号 I/O 出力バイトを指定します。指定数値は、下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
...	...

出力データ ポート番号で指定した出力グループの出力パターンを、0~255 の整数値で指定します。16 進数で表示された場合は、&H0 から &HFF が範囲になります。下位桁は、下位ビット (または最初の 4 出力ビット) を、上位桁は、上位ビット (または次の 4 出力ビット) を表わします。

Forced 省略可能です。通常は省略します。

解説

Out は、ポート番号と出力データの組み合わせで 8 個の出力ビットを同時に設定します。ポート番号パラメーターで、どのグループ (どの 8 個の出力ビット) を使うか指定します。例えば、ポート番号=0 のとき出力ビット 0~7、ポート番号=1 のとき出力ビット 8~15 を指定します。

まずポート番号で 8 個の出力ビットを指定したのち、出力データパラメーターで特定の出力パターンを指定します。出力データパラメーターがとる値は 0~255 で、16 進数あるいは 10 進数の整数で指定します。(&H0~&HFF、または 0~255)

下記表は、I/O の組み合わせ例の一部を、それに対応する出力データ値とともに、ポート番号が“0”および“1”のときに分けて示したものです。

ポート番号=0 の時の出力設定 (出力ビット番号)

出力データ値	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

ポート番号=1 の時の出力設定 (出力ビット番号)

出力データ値	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

注意

OpBCD と Out の違い

SPEL+の Out と OpBCD は、下記の 2 つの点で大きく異なります。

- OpBCD 命令は、オン、またはオフする 8 個の出力ビットを指定するのに 2 進 10 進 (BCD) フォーマットを使います。2 進 10 進フォーマットでは、&HA, &HB, &HC, &HD, &HE そして &HF などの値は使えないので、8 個の出力ビットの組み合わせすべてを満たすことができません
- Out 命令は、出力をオン、またはオフする 8 ビットの値に 0~255 の値を使います (OpBCD では 0~99 です)。これにより、8 ビット出力グループの可能な組み合わせすべてを満たせることになり、ユーザーの仕様に合わせて指定することができます。

Forced フラグ

NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時に I/O 出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InBCD, MemOff, MemOn, MemOut, MemSw, Off, On, Oport, Sw, Wait

Out 使用例

下記は、メインタスクが“iotask”という名前のタスクを起動するプログラムです。“iotask”は、出力ビット 0~3 をオンオフする簡単なタスクです。Out 命令を使うと、出力ビットを個々にオンオフせず、1 つのコマンドでそれを行うことができます。

```
Function main
    Xqt iotask
Do
    Go P1
    Go P2
Loop
Fend

Function iotask
Do
    Out 0, &H0F
    Out 0, &H00
    Wait 10
Loop
Fend
```

コマンドウィンドウからの簡単な操作例です。

```
> Out 1, 6      '出力 9 と 10 をオン
> Out 2, 1      '出力 8 をオン
> Out 3, 91     '出力 24, 25, 27, 28, 30 をオン
```

Out 関数

出力ポートの状態をバイト単位で返します。

書式

Out(ポート番号)

パラメーター

ポート番号 I/O 出力バイトを指定します。指定数値は、下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
...	...

戻り値

指定した出力ポートの状態をバイト単位で返します。

参照

Out

Out 関数使用例

```
Print Out(0)
```

OutReal

実数値の出力データを 32 ビット浮動小数点データ (IEEE754 準拠) として出力ポート 2 ワード (32 ビット) の状態を設定します。

書式

OutReal ワードポート番号, 出力データ [,Forced]

パラメーター

ワードポート番号	I/O 出力ワードを指定します。
出力データ	出力データ (Real 型実数) を、式または数値で指定します。
Forced	省略可能です。通常は省略します。

解説

ワードポート番号で指定した出力ワードポートと次の出力ワードポートへ指定 IEEE754 の Real 値を出力します。ワードポート番号パラメーターには、出力ワードラベルを使用できます。

注意

Forced フラグ

NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク) から、非常停止中や安全扉開時に I/O 出力のオンを行う場合にこのフラグを指定します。
非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InW, InBCD, InReal, Out, OutW, OpBCD, OutReal 関数

OutReal 使用例

```
OutReal 32, 2.543
```

OutReal 関数

出力ポートの状態を 32 ビット浮動小数点データ (IEEE754 準拠)として取得します。

書式

OutReal(ワードポート番号)

パラメーター

ワードポート番号 I/O 出力ワードを指定します。

戻り値

指定した出力ポートの状態を 32 ビット浮動小数点データ (IEEE754 準拠)で返します。

参照

In, InW, InBCD, InReal, Out, OutW, OpBCD, OutReal

OutReal 関数使用例

```
Real rdata01  
  
rdata01 = OutReal(0)
```

OutW

出力ポートの状態を 16 ビット同時にワード単位で設定します。

書式

OutW ワードポート番号, 出力データ [, Forced]

パラメーター

ワードポート番号	I/O 出力ワードを指定します。
出力データ	出力データ (0 から 65535 の整数)を、式または数値で指定します。
Forced	省略可能です。通常は省略します。

解説

パラメーターのワードポート番号で指定したユーザー I/O の出力ポートグループの状態を、指定した出力データに変更します。

注意

Forced フラグ

NoPause タスク, NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時に I/O 出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

参照

In, InW, Out

OutW 使用例

OutW 0, 25

OutW 関数

出力ポートの状態をワード単位 (2 バイト) で返します。

書式

OutW(ワードポート番号)

パラメーター

ワードポート番号 I/O 出力ワードを指定します。

戻り値

指定した出力ポートの状態を 16 ビットで返します。

参照

OutW

OutW 関数使用例

```
OutW 0, &H1010
```


P# (1. ポイント定義)

ポイントデータを定義します。

書式

P ポイント番号 = ポイント指定
 ポイントラベル = ポイント指定

パラメーター

ポイント番号	数値または括弧付きの式で指定します。 P 番号 P(式)
ポイントラベル	ポイントラベルを指定します。
ポイント指定	下記のポイントデータで指定します。 P ポイント番号, ポイントラベル, Here, Pallet, ポイントデータ関数 (Here 関数, XY 関数, JA 関数, Pulse 関数など) 詳細は、「P# (2. ポイント指定)」を参照してください。

解説

他のポイントまたはポイント式を代入することによってポイントデータを定義します。

参照

Local, Pallet, PDef, PDel, Plist

ポイントの代入例

下記はコマンドウィンドウからの操作例です。

P1 に座標データを設定します。
 > P1 = XY(300,200,-50,100)

左アームの姿勢を指定します:
 > P2 = XY(-400,200,-80,100) /L

P2 の X 座標データに 20 を加算し、その値を P3 として定義します。

```
> P3 = P2 +X(20)
> plist 3
P3 = XY(-380,200,-80,100) /L
```

P2 の Y 座標データから 50 を減算し、Z 座標データを-30 とし、右アームの姿勢として、その値を P4 として定義します。

```
>P4=P2 -Y(50) :Z(-30) /R
> plist 4
P4 = XY(-450,200,-30,100) /R
```

Pallet(3, 5)の U coord (U 座標値)に 90 を加算し、P6 として定義します。

```
> P5 = Here
> P6 = pallet(3,5) +U(90)
```

P# (2. ポイント指定)

ポイント指定と動作コマンドのためのポイントデータを指定します。

書式

point [{ + | - } ポイント指定] [ローカル番号] [ハンド(腕系)姿勢] [肘姿勢] [手首姿勢] [j4flag 値] [j6flag 値] [j1flag 値] [j2flag 値] [相対オフセット値] [絶対座標]

パラメーター

ポイント指定 ポイントデータを下記のいずれかで指定します。

P 番号

P(式)

ポイントラベル

Pallet (パレット番号, 分割番号)

現在位置指定

Here

ポイントデータ関数

XY 関数, JA 関数, Pulse 関数など

ローカル番号 ローカル番号 (1~15)を指定します。番号の前に必ずスラッシュ“/”か、“@”マークをつけてください。“/”は座標がローカルになるのを意味しています。“@”マークは座標がローカル座標に変換されることを意味します。省略可能です。

ハンド(腕系)姿勢 水平多関節型 (RS シリーズを含む)と垂直 6 軸型 (N シリーズを含む)のロボットのハンド(腕系)姿勢です。/L (左腕姿勢)、または/R (右腕姿勢)を指定します。省略可能です。

肘姿勢 垂直 6 軸型ロボット (N シリーズを含む)の場合に必要なパラメーターです。
/A (ABOVE)または/B (BELOW)を指定します。

手首姿勢 垂直 6 軸型ロボット (N シリーズを含む)の場合に必要なパラメーターです。
/F (FLIP)または/NF (NOFLIP)を指定します。

j4flag 値 垂直 6 軸型ロボット (N シリーズを含む)の場合に必要なパラメーターです。
/J4F0 または/J4F1 を指定します。

j6flag 値 垂直 6 軸型ロボット (N シリーズを含む)の場合に必要なパラメーターです。
/J6F0 から/J6F127 の値を指定します。

j1flag 値 RS シリーズと垂直 6 軸型ロボット (N シリーズを含まない)の場合に必要なパラメーターです。/J1F0 または/J1F1 を指定します。

j2flag 値 RS シリーズの場合に必要なパラメーターです。/J2F0 または/J2F1 を指定します。

j1angle 値 RS シリーズと N シリーズの場合に必要なパラメーターです。/J1A (実数値)を指定します。

j4angle 値 N シリーズの場合に必要なパラメーターです。/J4A (実数値)を指定します。

相対オフセット値 1 つまたはそれ以上の相対座標調整を行います。省略可能です。

{+ | -} {X | Y | Z | U | V | W | RZ | RY | RX | R | S | T | ST } (式)

TL オフセットは、現在のツール座標系の相対オフセットです。

{+ | -} {TLX | TLY | TLZ | TLU | TLV | TLW } (式)

絶対座標 1 つまたはそれ以上の絶対座標を指定します。座標の前に必ずコロン“:”をつけてください。この絶対座標は省略可能です。

: {X | Y | Z | U | V | W | R | S | T | ST } } (式)

解説

ポイント代入や動作コマンドでは、ポイント式を使用します。(下記例参照)

```
Go P1 + P2
P1 = P2 + XY(100, 100, 0, 0)
```

相対オフセット値の使用

ベースポイントに対して1つまたはそれ以上の座標をオフセットすることができます。次の例は、現在位置からロボットをX軸の正方向に20 mm動かしています。

```
Go Here +X(20)
```

これを再度実行すると、相対移動なので、ロボットは更にX軸方向に20 mm移動します。

垂直6軸型ロボット(Nシリーズを含む)では、座標軸周りの相対回転をするときは次のように指定します。次の例は、ロボットのツール姿勢を現在の姿勢を基準にX軸の正方向に20°回転します。

```
Go Here +RX(20)
```

相対ツールオフセットを行うことができます。

```
Go Here +TLX(20) -TLY(5.5)
```

垂直6軸型ロボット(Nシリーズを含む)では、相対オフセットで得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは相対オフセット演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。

LJM関数はこのような手首部の意図しない回転を防止できます。

```
Go LJM(Here +X(20))
```

絶対座標の使用

絶対座標を用いて、ベースポイントの1つまたはそれ以上の座標を変更することができます。次の例は、ロボットをX軸20 mmの位置に移動します

```
Go Here :X(20)
```

再度実行しても、ロボットのアームはすでに前の移動で指定された絶対座標位置に来ているので、動きません。

相対オフセット値と絶対座標は、ポイントデータを一時的に修正するときに便利です。次の例では、Zの相対オフセット10 mmを指定することで、ピック位置の上空10 mmのところを動き、ゆっくりとピックアップ位置に移動します。

```
Speed fast
Jump pick +Z(10)
Speed slow
Go pick
```

次の例は、第3関節の絶対値“0”を指定することで、現在位置から垂直上に移動させます。

```
LimZ 0
Jump Here :Z(0)
```

ローカルの使用

“/”、または“@”マークを使うことでローカル番号を指定できます。“/”と“@”マークには、それぞれ異なる機能があります。

座標をローカル座標系で指定するときは、“/”を使います。下記ステートメント例では、“/1”を付加することで「P1 の位置は、ローカル 1 の 0,0,0,0」と指定しています。

```
P1 = XY(0, 0, 0, 0) /1
```

座標をローカル座標に変換するときは、“@”マークを使います。下記ステートメント例では、現在位置を示すローカル座標値を P1 に登録しています。

```
P1 = Here @1
```

参照

Go, LJM, Local, Pallet, Pdel, Plist, Hand, Elbow, Wrist, J4Flag, J6Flag, J1Flag, J2Flag

ポイント指定使用例

代入ステートメントや動作コマンドでの、ポイント指定の例です。

```
P1 = XY(300,200,-50,100)
P2 = P1 /R
P3 = pick /1
P4 = P5 + P6
P(i) = XY(100, 200, CZ(P100), 0)
Go P1 -X(20) :Z(-20) /R
Go Pallet(1, 1) -Y(25.5)
Move pick /R
Jump Here :Z(0)
Go Here :Z(-25.5)
Go JA(25, 0, -20, 180)
pick = XY(100, 100, -50, 0)
```

```
P1 = XY(300,200,-50,100, -90, 0)
P2 = P1 /F /B
P2 = P1 +TLV(25)
```

PAgl 関数

指定した座標値から関節位置を計算して返します。

書式

PAgl (座標値, 関節番号)

パラメーター

座標値	関節位置を計算する座標値を指定します。
関節番号	関節番号 (1~9 の整数値) を、式または数値で指定します。 付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

計算した関節位置を、実数値で返します。回転関節 (単位: deg), 直動関節 (単位: mm)

参照

Agl, CX, CY, CZ, CU, CV, CW, CR, CS, CT, PPIs

PAgl 関数使用例

```
Real joint1  
  
joint1 = PAgl(P10, 1)
```

Pallet

パレットの定義と定義パレットの表示を行います。

書式

- (1) Pallet [Outside,] [/パレット番号, ポイント番号 1, ポイント番号 2, ポイント番号 3 [, ポイント番号 4], 分割数 1, 分割数 2]
- (2) Pallet [Outside,] パレット番号, 座標系データ 1, 座標系データ 2, 座標系データ 3
[, 座標系データ 4], 分割数 1, 分割数 2
- (3) Pallet

パラメーター

Outside	指定した行および列の範囲外にアクセス可能なパレットを作成します。 省略可能です。
パレット番号	パレット番号 (0~15 の整数)を式または数値で指定します。
ポイント番号 1~3	パレット定義 (標準の 3 点定義)に使用するポイント変数を指定します。
ポイント番号 4	4 点定義する場合に、ポイント番号 1~3 とともに用います。省略可能です。
分割数 1	パレットのポイント番号 1 (座標系データ 1)とポイント番号 2 (座標系データ 2)の 分割数を整数で指定します。 範囲は 1~32767 です。(分割数 1×分割数 2 <32767)
分割数 2	パレットのポイント番号 1 (座標系データ 1)とポイント番号 3 (座標系データ 3)の 分割数を整数で指定します。範囲は 1~32767 です。 (分割数 1×分割数 2 <32767)
座標系データ 1~3	パレット定義 (標準の 3 点定義)に使用する座標系をポイントデータで直接指定 します。
座標系データ 4	4 点定義する場合に、座標系データ 1~3 とともに用います。省略可能です。

結果

- (3)パラメーターを省略すると、定義されているパレットをすべて表示します。

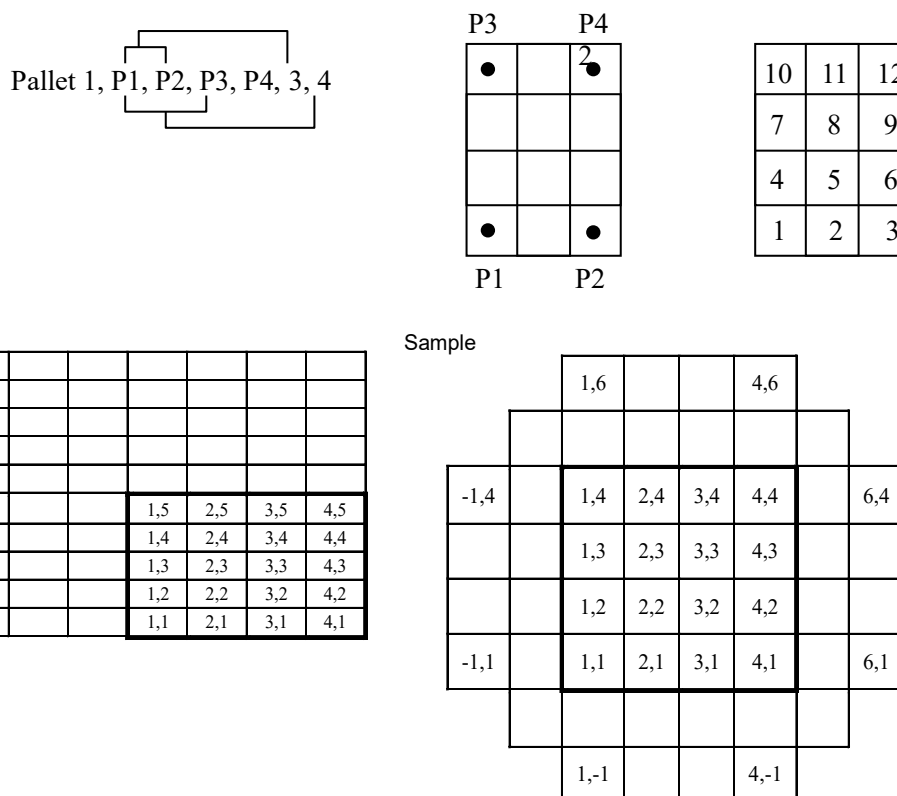
解説

ロボットに、ポイント番号 1 (座標系データ 1), ポイント番号 2 (座標系データ 2), ポイント番号 3 (座標系データ 3)の 3 点を最小限ティーチングし、なおかつポイント番号 1 (座標系データ 1)とポイント番号 2 (座標系データ 2)の分割数、およびポイント番号 1 (座標系データ 1)とポイント番号 3 (座標系データ 3)の分割数を指定し、パレットを定義します。

パレットが精度のよい方形であればコーナー 4 点のうち 3 点の位置指定だけで十分ですが、ほとんどの場合、全コーナー 4 点の位置を指定してパレット定義を行うことをお勧めします。

パレットを定義するには、まずコーナーの 3 または 4 点をティーチングし、以下のように進めます。

- 4 点定義の場合:下に P1, P2, P3 および P4 を示します。P1-P2 間には 3 点、
P1-P3 間には 4 点あり、合計 12 点を使って次の書式で定義します。



パレットの分割を示す各ポイントは、自動的に分割番号を割り当てられます。上記図の場合、P1 から始まっています。この分割番号は、Pallet 関数でも必要になります。

Outside で、行および列の範囲外にアクセス可能なパレットを作成するよう指定できます。

例:

```
Pallet Outside 1, P1, P2, P3, 4, 5
Jump Pallet(1, -2, 10)
```

注意

パレットサイズの上限

パレット定義で用いられるポイント数は、32767 以下でなければなりません。(分割数 1×分割数 2 < 32767)

不正確なパレット形状の定義

ポイントの順番やポイント間の分割数をまちがえると、パレット形状が不正確に定義されてしまいます。

パレット面の定義

パレット平面の高さは、コーナー3 点の Z 座標値で定義されます。したがって、縦置きパレットも定義することができます。

1 列パレットのパレット定義

3 点指定の Pallet 命令により、1 列のパレットも定義できます。1 列の場合は、両端の 2 点をティーチングし、次のように入力、実行します。同一番号方向の分割数は 1 とします。

```
> Pallet 2, P20, P21, P20, 5, 1      '5x1 パレットを定義
```

UVW 座標値

Pallet 命令に指定した 3 点 (4 点) の UVW 座標値が異なる場合、ポイント番号 1、および座標系データ 1 の UVW 座標値を使用します。

ポイント番号 2~4、および座標系データ 2~4 の UVW 座標値は無視されます。

付加軸座標値

Pallet 命令に指定した 3 点 (4 点) の座標値が付加軸座標値 (ST 軸値) を保持している場合、付加軸座標値においても均等に分割されます。つまり、付加軸を走行軸として使用している場合、パレット定義は走行軸の動作も考慮して計算するため、走行軸の位置を考慮したロボット動作範囲を超える大きなパレットを定義することが可能です。逆にパレット定義に影響されない付加軸が定義されている場合でも、パレット定義時には付加軸の位置にも注意を払う必要があります。

参照

Pallet 関数

Pallet 使用例

下記はコマンドウィンドウから P1, P2, P3 で定義されるパレットを設定する例です。パレット面は、15 点を均等に配置、P1-P2 間にパレットポイント番号 1~3 が並びます。

```
> pallet 1, P1, P2, P3, 3, 5  
> jump pallet(1, 2)          'パレットの指定位置へ Jump
```

この設定の結果作られるパレットは下記ようになります。

P3		
13	14	15
10	11	12
7	8	9
4	5	6
1	2	3
P1		P2

Pallet 関数

パレット上の位置を参照するポイントデータ関数です。

書式

- (1) Pallet (パレット番号, パレット位置番号)
- (2) Pallet (パレット番号, 分割横座標, 分割縦座標)

パラメーター

パレット番号	パレット番号 (0~15 の整数)を数値で指定します。
パレット位置番号	分割ポイントを指定する番号 (整数)を、式または数値 (1~32767)で指定します。
分割横座標	パレットの定義で指定した横座標を、数値 (-32768~32767)で指定します。
分割縦座標	パレットの定義で指定した縦座標を、数値 (-32768~32767)で指定します。

解説

Pallet は、あらかじめ Pallet ステートメントで定義されたパレット上の 1 点の位置を返します。この関数を動作命令 (Go または Jump 命令など)とともに用いることにより、アームをパレット上の指定位置に動かすことができます。

パレット位置番号は、式または整数値で定義することができます。

注意

垂直 6 軸型ロボット (N シリーズを含む)でのパレット動作

垂直 6 軸型ロボット (N シリーズを含む)では、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは上記ポイント演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。

LJM 関数はこのような手首部の意図しない回転を防止できます。

RS シリーズでのパレット動作

同様に RS シリーズでも、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、第 1 アームが意図しない方向へ回転することがあります。LJM 関数はこのような第 1 アームの意図しない回転を防止するために、ポイントデータの姿勢フラグを適切に変換します。

また、RS シリーズでは、姿勢フラグが変換された際に U 軸が動作範囲外に動作しようとしてエラーが発生することがあります。LJM 関数はこのような U 軸の動作範囲外エラーを防止するために、U 軸の目標角度を動作範囲内の目標角度に補正します。これは、姿勢フラグ選択で 2 を指定することによって利用可能となります。

UVW 座標値

Pallet 命令に指定した 3 点 (4 点)の U VW 座標値が異なる場合、ポイント番号 1、および座標系データ 1 の U VW 座標値を使用します。

ポイント番号 2~4、および座標系データ 2~4 の U VW 座標値は無視されます。

付加軸座標値

Pallet 命令に指定した 3 点 (4 点)の座標値が付加軸座標値 (ST 軸値)を保持している場合、付加軸座標値においても均等に分割されます。つまり、付加軸を走行軸として使用している場合、パレット定義は走行軸の動作も考慮して計算するため、走行軸の位置を考慮したロボット動作範囲を超える大きなパレットを定義することが可能です。逆にパレット定義に影響されない付加軸が定義されている場合でも、パレット定義時には付加軸の位置にも注意を払う必要があります。

参照

LJM, Pallet

Pallet 関数使用例

下記は、ワークをパレット1からパレット2に移すプログラム例です。

```
Function main
  Integer index
  Pallet 1, P1, P2, P3, 3, 5      'パレット1を定義
  Pallet 2, P12, P13, P11, 5, 3  'パレット2を定義
  For index = 1 To 15
    Jump Pallet(1, index)        'パレット1のポイントインデックスへ移動
    On 1                          'ワークをピックアップ
    Wait 0.5
    Jump Pallet(2, index)        'パレット2のポイントインデックスへ移動
    Off 1                         'ワークをリリース
    Wait 0.5
  Next I
Fend

Function main
  Integer i, j

  P0 = XY(300, 300, 300, 90, 0, 180)
  P1 = XY(200, 280, 150, 90, 0, 180)
  P2 = XY(200, 330, 150, 90, 0, 180)
  P3 = XY(-200, 280, 150, 90, 0, 180)

  Pallet 1, P1, P2, P3, 10, 10

  Motor On
  Power High
  Speed 50; Accel 50, 50
  SpeedS 1000; AccelS 5000

  Go P0
  P11 = P0 -TLZ(50)

  For i = 1 To 10
    For j = 1 To 10
      'ポイントの指定
      P10 = P11                      '退避ポイント
      P12 = Pallet(1, i, j)        '目標ポイント
      P11 = P12 -TLZ(50)            '接近開始ポイント
      '各ポイントの LJM 変換
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      '動作実行
      Jump3 P10, P11, P12 C0
    Next
  Next
Next
Fend
```

```
Function main2
  P0 = XY(300, 300, 300, 90, 0, 180)
  P1 = XY(400, 0, 150, 90, 0, 180)
  P2 = XY(400, 500, 150, 90, 0, 180)
  P3 = XY(-400, 0, 150, 90, 0, 180)
  Pallet 1, P1, P2, P3, 10, 10

  Motor On
  Power High
  Speed 50; Accel 50, 50
  Speeds 1000; Accels 5000

  Go P0

  Do
    'ポイントの指定
    P10 = Here -TLZ(50)                                '退避ポイント
    P12 = Pallet(1, Int(Rnd(9)) + 1, Int(Rnd(9)) + 1) '目標ポイント
    P11 = P12 -TLZ(50)                                  '接近開始ポイント

    If TargetOK(P11) And TargetOK(P12) Then              'ポイントのチェック
      '各ポイントの LJM 変換
      P10 = LJM(P10)
      P11 = LJM(P11, P10)
      P12 = LJM(P12, P11)
      '動作実行
      Jump3 P10, P11, P12 C0
    EndIf
  Loop
Fend
```

PalletClr

設定された Pallet をクリアします。

書式

PalletClr パレット番号

パラメーター

パレット番号 設定をクリアするパレット番号 (0~15 の整数)を数値で指定します。

参照

Pallet

PalletClr 使用例

```
PalletClr 1
```

ParseStr / ParseStr 関数

文字列を分析し、トークンの配列を返します。

書式

ParseStr 文字列\$, トークン\$(), 区切り文字\$
 トークン数 = ParseStr(入力文字列\$, トークン\$(), 区切り文字\$)

パラメーター

文字列\$	分析する文字列を指定します。
トークン\$()	トークンを格納する配列を指定します。 ByRef 宣言された配列は指定することができません。
区切り文字\$	1 文字以上の区切り文字を指定します。

戻り値

関数として使用する場合、分析したトークンの数を返します。

参照

Redim, String

ParseStr / ParseStr 関数使用例

```
String toks$(0)
Integer i

ParseStr "1 2 3 4", toks$(), " "

For i = 0 To UBound(toks)
    Print "token ", i, " = ", toks$(i)
Next i
```

Pass

指定点の近傍を停止しないで通過する PTP 動作です。

書式

Pass ポイント指定 [, {On | Off | MemOn | MemOff} ビット番号 [, ポイント指定 ...]] [LJM [姿勢フラグ選択]]

パラメーター

ポイント指定	P 番号, P(式), ポイントラベルを指定します。 ポイントデータに抜けがなく、昇順、または降順に並んでいる場合は、2 つのポイント番号をコロンで結合して P(1:5)のように指定できます。
ビット番号	オンオフする I/O 出力ビットまたは、メモリー I/O ビットを指定します。整数か出力ラベルで指定します。
LJM	待避座標, 接近座標, 目標座標を LJM 関数で変換します。省略可能です。
姿勢フラグ選択	LJM 関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。

解説

Pass は、指定ポイントデータ付近を通過してロボットアームを動かします。指定ポイントデータ自体は通過しません。

ポイントデータを指定するには、(P0, P1, ...) を使います。各ポイントはカンマで区切ってください。

動作実行中に出力ビットをオン、またはオフするには、ポイント間をカンマで区切って On/Off コマンドを挿入してください。On/Off が挿入された直前のポイントにロボットのアームが達する前に、On/Off が実行されます。

Pass の直後にもう 1 つの Pass ステートメントが続くと、制御は、最初の Pass の最終指定点付近で停止せず、次の Pass に移ります。

Pass の直後に Pass 以外の動作コマンドが続くと、ロボットは Pass ステートメントの最終指定点で止まりますが、Fine による位置決めは行いません。

Pass の直後に動作コマンド以外のコマンド、ステートメント、ファンクションなどが続くと、Pass で指定した最終点にロボットのアームが到達する前に、それらのコマンド、ステートメント、ファンクションなどが実行されます。

目標位置への Fine による正確な位置決めが求められる場合は、下記例のように、目標位置を指定して Pass のあとに Go を入れてください。

```
Pass P5; Go P5; On 1; Move P10
```

加速値または減速値が大きいほど、アームは指定点の近くを通過します。Pass 命令を使うことで、障害物を避けることができます。

LJM パラメーターを使うと、LJM 関数を使用するプログラムを簡略化できます。

例えば、

```
P11 = LJM(P1, Here, 1)
```

```
P12 = LJM(P2, P11, 1)
```

```
P13 = LJM(P3, P12, 1)
```

```
Pass P11, P12, P13
```

という 4 行のプログラムを

```
Pass P1, P2, P3 LJM 1
```

という 1 行のプログラムに置き換えることができます。

LJM パラメーターは垂直 6 軸型ロボット (N シリーズを含む)、および RS シリーズロボットで有効です。

姿勢フラグ選択をデフォルト値で使用する場合は、省略が可能です。

```
Pass P1, P2, P3 LJM
```

参照

Accel, Go, Jump, Speed

Pass 使用例

下記は、Pass 命令を使ってロボットアームを操作する例です。

```
Function main
```

```
Jump P1
```

```
Pass P2      'アームを P2 に近づけ P2 に到達する前に次のコマンドを実行
```

```
On 2
```

```
Pass P3
```

```
Pass P4
```

```
Off 0
```

```
Pass P5
```

```
End
```

Pause

一時停止可能なすべてのタスクを一時停止します。

書式

Pause

解説

Pauseを実行すると、一時停止可能なすべてのタスク (Xqt 命令で、NoPause や NoEmgAbort を指定しないタスク) を一時停止します。動作コマンド実行中のタスクは、すべて一時停止します。
バックグラウンドタスクは、Pause では一時停止しません。

注意

QP の Pause への影響

QP 命令は Pause が実行されたとき、直ちにロボットの動作を停止するか、動作終了後にプログラムを一時停止するかを設定します。詳細はヘルプの QP 命令を参照してください。

Pause 使用例

次の例は Pause によりタスクを一時停止します。Pause の行でプログラムは一時停止します。オペレーターウィンドウの<継続実行>ボタンを押してタスクを再開します。

```
Function main

    Xqt monitor
    Go P1
    On 1
    Jump P2
    Off 1
    Pause           'プログラムを一時停止
    Go P40
    Jump P50
Fend
```


PauseOn 関数

一時停止状態 (Pause 状態)を返します。

書式

PauseOn

戻り値

一時停止状態のときは“True”、それ以外のときは“False”を返します。

解説

本関数は、NoPause タスク, NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)と、バックグラウンドタスクでのみ使用します。

参照

ErrorOn, EstopOn, SafetyOn, Wait, Xqt

PauseOn 関数使用例

下記例は、コントローラーが一時停止になるのを監視し、一時停止が発生した場合 I/O をオンオフするプログラムです。ただし、安全扉が開いて一時停止状態になった場合は I/O をオンオフしません。

```
Function main

    Xqt PauseMonitor, NoPause
    :
    :
Fend

Function PauseMonitor
    Boolean IsPause
    IsPause = False
    Do
        Wait 0.1
        If SafetyOn = On Then
            If IsPause = False Then
                Print "Saftey On"
                IsPause = True
            EndIf
        ElseIf PauseOn = On Then
            If IsPause = False Then
                Print "InPause"
                If SafetyOn = Off Then
                    Off 10
                    On 12
                EndIf
                IsPause = True
            EndIf
        Else
            If IsPause = True Then
                Print "OutPause"
                On 10
                Off 12
                IsPause = False
            EndIf
        EndIf
    Loop
Fend
```

PDef 関数

指定したポイントデータが定義されているかを返します。

書式

PDef (ポイントデータ)

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルを指定します。

互換性に関する注意点

ポイントデータ引数に変数を指定できなくなりました。

変数を使用する場合には、PDef (P (varName)) と記述してください。

戻り値

ポイントデータが定義されている場合は“True”、それ以外は“False”を返します。

参照

Here, Pdel

PDef 関数使用例

```
If Not PDef (1) Then
    Here P1
EndIf
Integer i
For i = 0 to 10
    If PDef (P(i)) Then
        Print "P(";i;) is defined"
    EndIf
Next
```

PDel

指定したポイントデータを削除します。

書式

PDel 開始ポイント番号 [, 終了ポイント番号]

パラメーター

開始ポイント番号 削除するポイントデータ範囲の開始番号を、整数値または式で指定します。

終了ポイント番号 削除するポイントデータ範囲の終了番号を、整数値または式で指定します。

解説

ロボットのメモリー上から指定したポイントデータを削除します。パラメーターの開始ポイント番号から終了ポイント番号までのポイントデータを削除します。開始ポイント番号は、終了ポイント番号より小さい値にしてください。

PDel 使用例

```
> p1=10,300,-10,0/L
> p2=0,300,-40,0
> p10=-50,350,0,0
> pdel 1,2      'ポイント1と2を削除
> plist
P10 =   -50.000,   350.000,    0.000,    0.000 /R /0
> pdel 50      'ポイント50を削除
> pdel 100,200 'ポイント100から200を削除
>
```

PDescription

指定したポイントデータのコメントを設定します。

書式

PDescription ポイントデータ,新コメント

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルで指定します。

ポイントデータ引数に変数は指定できません。

変数を使用する場合には、PDescription (P(varName)), “新コメント”
と記述してください。

新コメント 指定したポイントデータのコメントを指定する文字列式です。

解説

PDescription は、コントローラーのメモリー上の指定したポイントデータにコメントを保存します。
コントローラーのメモリー上に保存したコメントは、プロジェクトを作成したりプログラムを実行したりするとメモリーから消去されます。保存が必要な場合は、“SavePoints”を実行してポイントファイルに保存してください。

参照

PDef 関数, PDescription\$ 関数, PLabel, PLabel\$ 関数

PDescription 使用例

```
PDescription 1, "Comment"
```

PDescription\$ 関数

指定したポイント番号に定義されたポイントのコメントを返します。

書式

PDescription\$(ポイントデータ)

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルで指定します。

ポイントデータ引数に変数は指定できません。

変数を使用する場合には、PDescription\$(P(varName))と記述してください。

戻り値

指定したポイント番号のコメントを文字列で返します。

参照

PDef 関数, PDescription, PLabel, PLabel\$関数

PDescription\$ 関数使用例

```
Print PDescription$(1)
Print PDescription$(P(i))
```

PeakSpeedClear

関節のピーク速度をクリアし、初期化します。

書式

PeakSpeedClear [関節指定 1 [, 関節指定 2 [, 関節指定 3 [, 関節指定 4 [, 関節指定 5 [, 関節指定 6 [, 関節指定 7 [, 関節指定 8 [, 関節指定 9]]]]]]]]]

パラメーター

関節指定 1 - 関節指定 9 関節番号を整数値、または式で指定します。パラメーターを指定しない場合は、全関節のピーク速度値がクリアされます。
付加軸の S 軸は 8、T 軸は 9 になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

PeakSpeedClear は指定した関節のピーク速度値をクリアします。
PeakSpeed 実行前に、必ず PeakSpeedClear を実行してください。

本コマンドは、PG 付加軸はサポートしません。

参照

AvgSpeed, PeakSpeed

PeakSpeedClear 使用例

<例 1>

全関節のピーク速度値をクリアしたのち、指定した関節番号の速度値を表示させるコマンド実行例です。

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
    -0.273
> PeakSpeed
    -0.273      -0.164
    -0.080      0.258
    -0.005      0.401
     0.000      0.000
     0.000
>
```

<例 2>

垂直多関節ロボットにおいて、第 1 関節, 第 4 関節, 第 5 関節のピーク速度値をクリアしたのち、指定した関節番号のピーク速度値を表示させるコマンド実行例です。

```
> PeakSpeedClear 4, 1, 5
> Go P1
> PeakSpeed 1
    -0.273
> PeakSpeed 4
    0.258
```

PeakSpeed

指定した関節のピーク速度値を表示します。

書式

PeakSpeed [関節番号]

パラメーター

関節番号 関節番号を整数値、または式で指定します。省略可能です。
付加軸の S 軸は 8、T 軸は 9 になります。

結果

全関節について現在のピーク速度値を表示します。

解説

PeakSpeed は、関節速度の絶対値が最大となる速度に符号をつけた値を表示します。最大速度は 1 になります。ピーク速度は、-1~1 までの実数値で表わされます。

PeakSpeedClear を実行してから、PeakSpeed を実行し、関節のピーク速度値を表示してください。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度からピーク速度を計算します。
本コマンドは、PG 付加軸はサポートしません。

参照

AvgSpeed, PeakSpeedClear, PeakSpeed 関数

PeakSpeed 使用例

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
    -0.273
> PeakSpeed
    -0.273      0.163
    -0.080      0.258
    -0.005     -0.401
     0.000      0.000
     0.000
```

PeakSpeed 関数

指定した関節のピーク速度を返します。

書式

PeakSpeed (関節番号)

パラメーター

関節番号 関節番号を整数値、または式で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

-1~1 までの実数値で返します。

解説

PeakSpeed 関数は、関節速度の絶対値が最大となる速度に符号をつけた値を返します。最大速度は 1 になります。ピーク速度は、-1~1 までの実数値で表わされます。

PeakSpeedClear を実行してから、PeakSpeed を実行し、関節のピーク速度値を表示してください。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度からピーク速度を計算します。
本関数は、PG 付加軸はサポートしません。

参照

AvgSpeed, PeakSpeedClear, PeakSpeed

PeakSpeed 関数使用例

プログラム中に PeakSpeed 関数を使用した例です。

```
Function DisplayPeakSpeed
  Integer i

  PeakSpeedClear
  Go P1
  Print "Peak Speeds:"
  For i = 1 To 6
    Print "Joint ", i, " = ", PeakSpeed (i)
  Next i
End
```


PerformMode

ロボットのモードを選択できます。

書式

- (1) PerformMode [モード番号] [, ロボット番号]
- (2) PerformMode

パラメーター

モード番号 設定するモード番号を整数値 (0~2)もしくは以下に示す定数で指定します。
コマンドウィンドウからの実行時のみ省略可能です。

定数	値	内容
MODE_STANDARD	0	標準モードを設定します。
MODE_BOOST	1	ブーストモードを設定します。
MODE_LOW_VIBRATRION	2	低振動モードを設定します。

ロボット番号 設定するロボットの番号を整数値で指定します。
省略した場合、現在選択されているロボットが対象となります。

結果

- (1)の書式で指定すると、指定したモード番号で動作モードを設定します。
- (2)の書式で指定すると、現在選択されているロボットに設定されているモード番号を表示します。

解説

PerformMode は用途に応じて優先するロボットの性能 (モード)を変更する機能です。選択可能なモードは、各マニピュレーターマニュアルを参照してください。

標準モード

タクトタイム, 動作デューティー, 動作停止時の振動のバランスを優先したモードです。
あらゆるアプリケーションに使用できます。

ブーストモード

一動作の動作時間を短縮させることを優先したモードです。
標準モードに比べ、動作デューティーと動作停止時の振動は悪化しますが、一動作時間が短くなります。
高デューティーの用途で使用する場合、標準モードの方が、タクトタイムが短くなることがあります。
推奨アプリケーション: 搬送など

低振動モード

動作停止時の振動を小さくすることを優先したモードです。
標準モードに比べ、動作時間が長くなりますが、停止時の振動が小さくなります。
推奨アプリケーション: 精密部品の搬送, 組み立てなど

各モードの性能比較表

モード	比較項目		
	一動作時間 (*1)	停止時振動	動作デューティ (*2)
標準	○	○	○
ブースト	◎	△	△
低振動	△	◎	○

表中の記号は性能の度合いを表します。

○: 基準 ◎: 向上 △: やや低下

(*1) 一動作時間とはロボットが現在位置から目標座標までにかかる移動時間です。

(*2) 動作デューティとは、最大加減速度で過負荷エラーなく、動作できる動作時間の割合です。

注意

- 未対応機種の場合、ブーストモード、および低振動モードに変更すると、エラーが発生します。
- 対応動作指令: PTP 動作 (Go, BGo, TGo, Jump, JTran, PTran, Pulse)

* CP 動作の以下の性能は変化しません。

軌跡精度

AccelS, AccelR, SpeedS, SpeedR の上限値

加速度指令エラー, 速度指令エラーの発生確率

モードが自動的に初期モード (標準モード)に変更される条件について

下表にモードが自動的に変更される条件を示します。

	モードの変化
コントローラー電源 ON	初期モードに変わる
コントローラーの再起動	初期モードに変わる
モーターON	初期モードに変わる
ビルド / リビルド	変わらない
Reset 実行時	変わらない
SFree 実行時	初期モードに変わる

参照

Bo, Go, Jump, JTran, PerformMode 関数, TGo

PerformMode 使用例

```
PerformMode MODE_STANDARD
Go P1
PerformMode 2
Go P2
```

PerformMode 関数

ロボットの動作モードの状態を返します。

書式

PerformMode ([ロボット番号])

パラメーター

ロボット番号 状態を確認するロボットの番号を整数値で指定します。
省略した場合、現在選択されているロボットが対象となります。

戻り値

現在設定されている動作モードの値を整数値で返します。

- 0 = 標準モード
- 1 = ブーストモード
- 2 = 低振動モード

参照

PerformMode

PerformMode 関数使用例

```
Print PerformMode (1)
```

PG_FastStop

連続回転中のパルス出力軸を急停止します。

書式

PG_FastStop

解説

PG_FastStopを実行すると、現在選択されているPGロボットの連続回転動作を急停止します。
通常の停止は減速停止 (PG_SlowStop)を使用してください。

参照

PG_Scan, PG_SlowStop

PG_FastStop 使用例

次の例はパルス出力軸を 10 秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_FastStop          '連続回転動作を急停止
End
```

PG_LSpeed

パルス出力軸の加速開始時のパルス速度および減速終了時のパルス速度を設定します。

書式

PG_LSpeed 速度設定値 [, 減速終了速度],

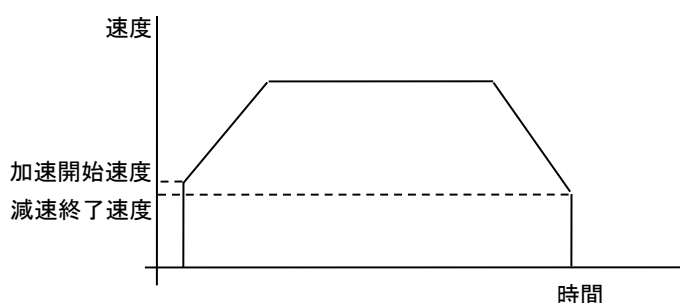
パラメーター

速度設定値 パルス速度 (1~32767 の整数、単位: pulse/sec)を式または数値で指定します。

減速終了速度 減速終了時のパルス速度 (1~32767 の整数、単位: pulse/sec)を、式または数値で指定します。

解説

PG_LSpeed は、パルス出力軸の加速開始時のパルス速度、および減速終了時のパルス速度を指定します。ステッピングモータの初速度、および終了速度を最大自起動周波数の範囲内で高く設定し、高性能なモータの性能を発揮したり、低く設定してステッピングモータの脱調を防止したりするのに使用します。デフォルト値は、300pulse/sec が設定されており、通常はそのまま使用してください。



減速終了速度を省略すると、速度設定値が、減速終了速度の設定値となります。

下記のいずれかの場合、PG_LSpeed 値は、初期化されます。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

参照

PG_LSpeed 関数

PG_LSpeed 使用例

PG_LSpeed は、コマンドウィンドウでも、プログラム中でも、用いることができます。下記に両方の場合の使用例を示します。

```
Function pglstest
  Motor On
  Power High
  Speed 30;Accel 30,30
  PG_LSpeed 1000
  Go P0
Fend
```

コマンドウィンドウから PG_LSpeed 値を設定する例です。

```
> PG_LSpeed 1000,1100
>
```

PG_LSpeed 関数

現在設定されているパルス出力軸の加速開始時のパルス速度、および減速終了時のパルス速度を返します。

書式

PG_LSpeed [(パラメーター番号)]

パラメーター

パラメーター番号 下記に示す番号から、設定値番号を 1 つ指定します。
省略した場合は 1 が指定されたとみなします。
1: 加速開始時パルス速度
2: 減速終了時パルス速度

戻り値

1~32767 の整数値、単位: pulse/sec を返します。

参照

PG_LSpeed

PG_LSpeed 関数使用例

```
Integer savPGLSpeed  
savPGLSpeed = PG_LSpeed(1)
```

PG_Scan

パルス出力軸の連続回転動作を開始します。

書式

PG_Scan 回転方向

パラメーター

回転方向	連続回転の回転方向を指定します。
0:	+ (CW)方向
1:	- (CCW)方向

解説

PG_Scan を実行すると、現在選択されている PG ロボットの連続回転動作を開始します。
連続回転動作を実行するためには、ロボット設定にて PG パラメーターの連続回転を有効にする必要があります。
プログラム実行タスクが終了すると、連続回転は停止します。

参照

PG_FastStop

PG_Scan 使用例

次の例はパルス出力軸を 10 秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  Power High
  Speed 10; Accel 10,10
  PG_Scan 0
  Wait 10
  PG_SlowStop
End
```

PG_SlowStop

連続回転中のパルス出力軸を減速停止します。

書式

PG_SlowStop

解説

PG_SlowStopを実行すると、現在選択されているPGロボットの連続回転動作を減速停止します。

参照

PG_Scan, PG_FastStop

PG_SlowStop 使用例

次の例はパルス出力軸を10秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_SlowStop           '連続回転動作を急停止
End
```


PLabel

指定したポイントデータのラベルを設定します。

書式

PLabel ポイント番号,新ラベル

パラメーター

ポイント番号 ポイント番号を式または数値で指定します。

新ラベル 指定したポイントデータに使うラベルを指定する文字列式です。

参照

PDef 関数, PDescription, PDescription\$ 関数, PLabel\$ 関数, PNumber 関数

PLabel 使用例

```
PLabel 1, "pick"
```

PLabel\$ 関数

指定したポイント番号に定義されたポイントラベルを返します。

書式

PLabel\$(ポイントデータ)

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルで指定します。

互換性に関する注意点

ポイントデータ引数に変数を指定できなくなりました。

変数を使用する場合には、PLabel\$(P(varName))と記述してください。

参照

PDef 関数, PDescription, PDescription\$ 関数, PLabel, PNumber 関数

PLabel\$ 関数使用例

```
Print PLabel$(1)
Print PLabel$(P(i))
```

Plane

進入検出平面の設定と表示をします。

書式

- (1) Plane 平面番号[, ロボット番号], 座標系データ
- (2) Plane 平面番号[, ロボット番号], 原点, X 軸指定, Y 軸指定
- (3) Plane 平面番号[, ロボット番号]
- (4) Plane

パラメーター

平面番号	進入検出平面の番号を指定します。進入検出平面は 1~15 の整数で、合計 15 個まで定義することができます。
ロボット番号	設定するロボットの番号を整数値で指定します。 ロボット番号を省略した場合、現在選択されているロボットが対象となります。
座標系データ	進入検出平面の座標系をポイントデータで直接指定します。
原点	進入検出平面の原点を定義するロボット座標系上の位置を、P#(整数)、または P(式)で指定します。
X 軸指定	進入検出平面の X 軸上の点を定義するロボット座標系上の位置を、P#(整数)、または P(式)で指定します。
Y 軸指定	進入検出平面の Y 軸上の点を定義するロボット座標系上の位置を、P#(整数)、または P(式)で指定します。

結果

- (3)の書式で指定すると、指定した平面番号の進入検出平面の設定を表示します。
- (4)の書式で指定すると、現在選択されているロボットの設定されているすべての進入検出平面の設定を表示します。

解説

Plane は、進入検出平面を設定します。進入検出平面は、現在選択しているツールに基づいて算出される手先位置が、設定した進入検出平面によって分けられる空間の一方に入っているかどうかを検出するために使用します。進入検出平面は、ロボットのベース座標系上に置かれる任意の座標系の XY 平面によって設定されます。そして、その平面によって分けられる空間のうち、その座標系の Z 軸+方向を含む空間内に手先位置が入ったとき進入検出平面に進入しているとみなします。

進入検出平面を設定すると、コントローラーの起動中はロボットのモーターパワーの状態に関わらず常に検出処理が実行されます。

以下、各書式について説明します。

- (1) ベース座標系を基準とした並進量と回転量を表すポイントデータによって進入検出平面の元になる座標系を指定し、進入検出平面を設定します。

例:

```
Plane 1, XY(x, y, z, u, v, w)
Plane 1, P1
```

- (2) 原点, X 軸上ポイント, Y 軸上ポイントの 3 点を指定することで、進入検出平面 (XY 平面) を定義します。各ポイントの X, Y, Z 座標のみを使用し、U, V, W 座標は無視します。同時に右手系をなすように Z 軸が計算され、進入検出方向が設定されます。

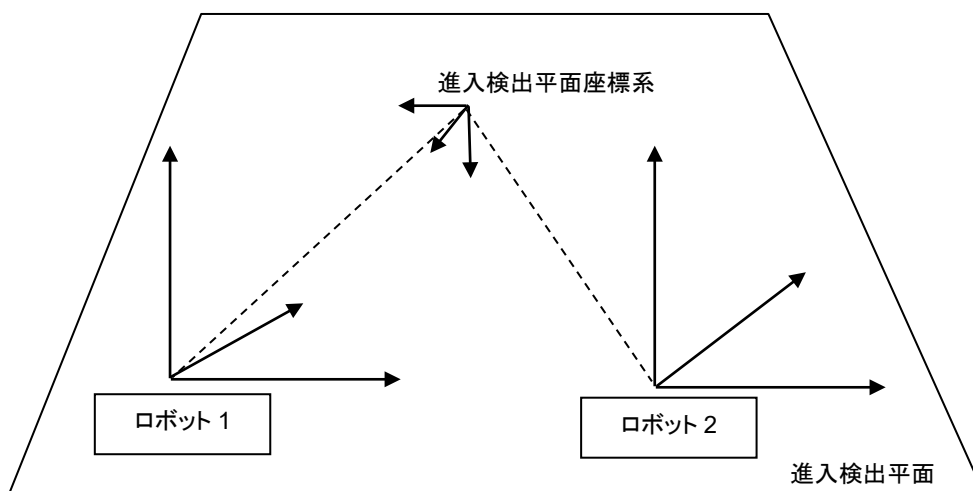
例:

Plane 1, P1, P2, P3

- (3) 指定された進入検出平面の設定を表示します。
- (4) 設定されているすべての進入検出平面の設定を表示します。

進入検出平面に対する進入検出の結果は、GetRobotInsidePlane 関数, InsidePlane 関数を使って随時取得できます。また Wait コマンドの待ち条件式として GetRobotInsidePlane 関数を利用することもできます。また、検出結果を I/O に出力するようにリモート出力設定することもできます。

複数のロボットで 1 つの平面を共有するには、それぞれのロボット座標から見た平面を定義する必要があります。



ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

ツール選択

進入検出は、現在選択しているツールで行います。したがって、ツールの選択を変更した場合、ロボットが動作していないのに平面内から平面外、またはその逆となるケースがあります。

付加軸

付加軸 (走行軸含む) S, T 軸を持つロボットの場合設定される進入検出平面は付加軸の位置には依存しません。ロボットのベース座標系を基準に設定されます。

参照

Box, GetRobotInsidePlane, InsidePlane, PlaneClr, PlaneDef

ヒント

Plane をロボットマネージャーで設定する

EPSON RC+ では、[プロジェクト]メニュー-[ロボットマネージャー]の[進入検出平面]パネルから Plane 値を設定できます。

Plane 使用例

コマンドウィンドウからの操作例です。

ロボット座標系を基準に Z 軸方向に-20mm の水平面で下向き方向を検出方向として定義する例:

```
> plane 1, xy(100, 200, -20, 90, 0, 180)
```

ロボット座標系を基準に X 方向に 50mm、Y 方向に 200mm 移動し、Y 軸周りに 45 度回転して得られる座標系が作る XY 平面を進入検出平面として定義する例:

```
> plane 2, xy(50, 200, 0, 0, 45, 0)
```

ロボットのツール座標系を使って進入検出平面を直接設定する。(垂直 6 軸型ロボット)

```
> plane 3, here
```

Plane 関数

設定された進入検出平面を返す関数です。

書式

Plane(平面番号[, ロボット番号])

パラメーター

- 平面番号 確認する進入検出平面番号 (1~15 の整数)を、式または数値で指定します。
- ロボット番号 設定するロボットの番号を整数値で指定します。
ロボット番号を省略した場合、現在選択されているロボットが対象となります。

戻り値

設定されている進入検出平面の元になる座標系データをポイントデータとして返します。

参照

GetRobotInsidePlane, InsidePlane, Plane, PlaneClr, PlaneDef

Plane 関数使用例

```
P1 = Plane(1)
```

PlaneClr

進入検出平面をクリア (未定義)にします。

書式

PlaneClr 平面番号[, ロボット番号]

パラメーター

平面番号	設定をクリア (未定義)にする進入検出平面番号 (1~15 の整数)を、式または数値で指定します。
ロボット番号	設定するロボットの番号を整数値で指定します。 ロボット番号を省略した場合、現在選択されているロボットが対象となります。

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

GetRobotInsidePlane, InsidePlane, Plane, PlaneDef

PlaneClr 使用例

```
PlaneClr 1
```

PlaneDef 関数

進入検出平面の設定状態を返します。

書式

PlaneDef(平面番号[, ロボット番号])

パラメーター

- 平面番号** 状態を返す進入検出平面の番号 (1~15 の整数)を指定します。
- ロボット番号** 設定するロボットの番号を整数値で指定します。
ロボット番号を省略した場合、現在選択されているロボットが対象となります。

戻り値

平面番号で指定された進入検出平面が設定されている場合は“True”、設定されていない場合は“False”を返します。

参照

GetRobotInsidePlane, Box, InsidePlane, Plane, PlaneClr

PlaneDef 関数使用例

```
Function DisplayPlaneDef(planeNum As Integer)

    If PlaneDef(planeNum) = False Then
        Print "Plane ", planeNum, "is not defined"
    Else
        Print "Plane 1: ",
        Print Plane(planeNum)
    EndIf
End
```


PList

現在のロボットのメモリー上のポイントデータを表示します。

書式

- (1) PList
- (2) PList ポイント番号
- (3) PList 開始ポイント番号,
- (4) PList 開始ポイント番号, 終了ポイント番号

パラメーター

ポイント番号	番号の範囲は 0~999 までです。
開始ポイント番号	表示する先頭のポイント番号を 0~999 までで指定します。
終了ポイント番号	表示する最後のポイント番号を 0~999 までで指定します。

結果

ポイントデータを返します。

解説

PList は現在のロボットのメモリー上のポイントデータを表示します。

指定した範囲のポイントデータが存在しない場合、データは表示されません。
開始ポイント番号を終了ポイント番号より大きい値に指定した場合は、エラーになります。

- (1) PList
すべての位置データを表示します。
- (2) PList ポイント番号
指定した 1 つのポイント番号の位置データを表示します。
- (3) PList 開始ポイント番号,
開始ポイント番号から位置データの終わりまで表示します。
- (4) PList 開始ポイント番号, 終了ポイント番号
指定した開始ポイント番号から終了ポイント番号までの位置データを表示します。

PList 使用例

表示される形式はロボットのタイプや付加軸の有無で変わります。
以下の例はスカラ型ロボットで、付加軸無しの場合です。

次の例は、指定したポイントデータを表示します。

```
> plist 1
P1   = XY( 200.000,    0.000,  -20.000,    0.000 )  /R /0
>
```

次の例は、10~20 までのポイントデータを表示します。この例では、指定した範囲内に 3 つのポイントデータが存在します。

```
> plist 10, 20
P10  =   XY( 290.000,    0.000,  -20.000,    0.000 ) /R /0
P12  =   XY( 300.000,    0.000,    0.000,    0.000 ) /R /0
P20  =   XY( 285.000,   10.000,  -30.000,   45.000 ) /R /0
>
```

次の例は、10 から終わりまでのポイントデータを表示します。

```
> plist 10,
P10  =   XY( 290.000,    0.000,  -20.000,    0.000 ) /R /0
P12  =   XY( 300.000,    0.000,    0.000,    0.000 ) /R /0
P20  =   XY( 285.000,   10.000,  -30.000,   45.000 ) /R /0
P30  =   XY( 310.000,   20.000,  -50.000,   90.000 ) /R /0
```

PLocal

指定したポイントデータのローカル属性を設定します。

書式

PLocal(ポイントデータ) = ローカル番号

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルを指定します。

互換性に関する注意点

ポイントデータ引数に変数を指定できなくなりました。

変数を使用する場合には、PLocal (P (varName)) と記述してください。

ローカル番号 設定するローカル属性の番号を、0~15 の整数値または式で指定します。

参照

PLocal 関数

PLocal 使用例

```
PLocal(pick) = 1
```

PLocal 関数

指定したポイント番号に設定されたローカル番号を返します。

書式

PLocal(ポイントデータ)

パラメーター

ポイントデータ 整数値, P 番号, P(式), ポイントラベルを指定します。

互換性に関する注意点

ポイントデータ引数に変数を指定できなくなりました。

変数を使用する場合には、PLocal (P (varName)) と記述してください。

戻り値

指定したポイント番号のローカル番号を返します。

参照

PLocal

PLocal 関数使用例

```
Integer localNum  
  
localNum = PLocal (pick)
```

Pls 関数

現在位置の各関節パルス値を返します。

書式

Pls(関節番号)

パラメーター

関節番号 現在のパルス値を求める関節を指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

関節番号で指定された関節の、現在位置のパルス値を数値で返します。

解説

Pls は、各関節の現在のパルス値を読むために使います。求められた値は保存し、後で **Pulse** コマンドとともに用いることができます。

参照

CX, CY, CZ, CU, CV, CW, Pulse

Pls 関数使用例

下記は、各関節のパルス値を求めて、その値を出力する簡単な例です。

```
Function plstest
  Real t1, t2, z, u
  t1 = pls(1)
  t2 = pls(2)
  z = pls(3)
  u = pls(4)
  Print "T1 joint current Pulse Value: ", t1
  Print "T2 joint current Pulse Value: ", t2
  Print "Z joint current Pulse Value: ", z
  Print "U joint current Pulse Value: ", u
Fend
```

PNumber 関数

ポイントラベルに対応するポイント番号を返します。

書式

PNumber(ポイントラベル)

パラメーター

ポイントラベル	現在のポイントファイルにあるポイントラベル、またはポイントラベルを表わす文字列を指定します。

参照

PDef 関数, PLabel\$関数

PNumber 関数使用例

```
Integer pNum
String pointName$

pNum = PNumber (pick)

pNum = PNumber ("pick")

pointName$ = "place"
pNum = PNumber (pointName$)
```

PosFound 関数

Find 命令時実行の状態を返します。

書式

PosFound

戻り値

アーム移動中に座標位置が見つかった場合は“True”、それ以外は“False”を返します。

参照

Find

PosFound 関数使用例

```
Find Sw(5) = ON
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Error: Cannot find the sensor signal."
EndIf
```

Power

パワーモードを High または Low にし、現在のモードを表示します。

書式

- (1) Power { High | Low } [, Forced]
- (2) Power

パラメーター

- High | Low High または Low を設定します。デフォルト設定は、Low です。
- Forced 省略可能です。通常は省略します。

結果

パラメーターを省略すると、現在のパワーモードを表示します。

解説

パワーモードを High または Low にします。また、現在のパワーモードを表示します。

Low : パワーモードが Low に設定されていると、ローパワーモードがオンになります。これは、ロボットがゆっくりと (250mm/sec 以下の速度で)動作することを意味します。また、モーターパワー出力を低く制限します。

High : パワーモードが High に設定されていると、ローパワーモードがオフになります。これは、ロボットが Speed, Accel, SpeedS, AccelS で指定した速度、加減速度で動作することを意味します。

下記に、ローパワーモードへの切り替え操作を示します。この場合、速度と加減速度は各ロボットのデフォルト値に制限されます。デフォルト値については各ロボットの仕様表に記載されています。ユーザーズガイドの「2. 安全について」も参照してください。

ローパワーモードになる条件

コントローラー電源 オン Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

デフォルト値に制限される設定

Speed Accel SpeedS AccelS

注意

ローパワーモード (Power Low)と最大速度の関係

ローパワーモードではモーター出力が制限され、実際の動作速度は初期値の範囲内になります。ローパワーモード設定時に、コマンドウィンドウまたはプログラム中で、高速にする指示が出されても、初期値速度で動作します。より高速での動作が必要ならば、**Power High** に設定しなくてはなりません。ロボットがハイパワーで動作中に、ローパワーモードに切り替えると、過速度エラーやローパワートルクエラーが発生する可能性があります。

ハイパワーモード (Power High)と最大速度の関係

ハイパワーモードでは初期値よりも高速の速度を用いることができます。

Forced フラグ

ロボット動作中 (一時停止中も含む) にパワーモードの変更が可能になります。ロボットがローパワーで動作中にハイパワーモードに切り替えると、次の動作から設定速度で高速に動作が可能になります。ロボットがハイパワーで動作中にローパワーモードに切り替えると、過速度エラーやローパワートルクエラーが発生する可能性があります。ロボットを一時停止してから、**Forced** フラグを指定してローパワーモードに切り替えてください。

参照

Accel, AccelS, Speed, SpeedS

Power 使用例

下記に **Power** を用いたコマンドウィンドウからの操作例を示します。

> Speed 50	'Low Power モードで高速度を設定
> Accel 100, 100	'高加速度を設定
> Jump P1	'低速度、低加速度で移動
> Speed	'現在の速度を表示
Low Power Mode	
50	
50 50	
> Accel	'現在の加速度を表示
Low Power Mode	
100 100	
100 100	
100 100	
> Power High	'High Power モードに設定
> Jump P2	'ロボットを高速度で動作

Power 関数

パワーモードを返します。

書式

Power [(ロボット番号)]

パラメーター

ロボット番号 状態を確認するロボット番号を整数値で指定します。
省略した場合、現在選択されているロボットが対象となります。

戻り値

0 = ローパワーモード
1 = ハイパワーモード

参照

Power

Power 関数使用例

```
If Power = 0 Then  
    Print "Low Power Mode"  
EndIf
```

PPIs 関数

指定した座標値から関節パルスを計算して返します。

書式

PPIs (座標値, 関節番号)

パラメーター

座標値	ポイントデータを指定します。
関節番号	関節番号 (1~9 の整数) を式または数値で指定します。 付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

計算した関節パルスを Long 型の値で返します。

参照

Agl, CX, CY, CZ, CU, CV, CW, PAgl

PPIs 関数使用例

```
Long pulses1  
  
pulses1 = PPIs(P10, 1)
```

Print

Run ウィンドウ, オペレーターウィンドウ, コマンドウィンドウ, マクロウィンドウなどのディスプレイへデータを表示します。

書式

```
Print 表示データ[,表示データ...][,]  
Print
```

パラメーター

表示データ 数値または文字列式で指定します。省略可能です。
, (カンマ) ステートメントの終わりにカンマがあると、改行されません。省略可能です。

結果

変数データまたは指定された文字列を返します。

解説

ディスプレイデバイスに、変数データまたは文字列を表示します。

行末にカンマがない限り、自動的に改行されます。

注意

本命令が一度にあつかえる最大データ長は 256 Byte です。

ループ文の中で Print 命令だけを使用しないでください

ループ文の中で Print 命令だけを使用すると、コントローラーがハングアップ状態になる場合があります。Wait 命令、または、動作命令と一緒に使用してください。

悪い例

```
Do  
    Print "1234"  
Loop
```

良い例

```
Do  
    Print "1234"  
    Wait 0.1  
Loop
```

参照

Print #

Print 使用例

下記例では、P100 の座標値から U 軸座標値を引いて、その値を変数 *uvar* に入れた値を、現在のディスプレイに表示しています。

```
Function test  
    Real uvar  
    uvar = CU(P100)  
    Print "The U Axis Coordinate of " + Chr$(34) + "P100" + Chr$(34) +  
    " is ", uvar  
End
```

Print

指定されたファイル, 通信ポート, データベースまたはデバイスにデータを出力します。

書式

Print #ポート番号, 出力データ [, 出力データ...][,]

パラメーター

ポート番号	ファイル, 通信ポート, データベースまたはデバイスを示す ID 番号です。 ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。 通信ポート番号は OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。 データベース番号は、OpenDB ステートメントで指定した番号です。 デバイス ID は以下の数値です。 21 RC+ 24 TP (TP1 のみ) 20 TP3
出力データ...	数値、あるいは文字列を指定します。
, (カンマ)	ステートメントの終わりにカンマがあると、改行されません。省略可能です。

解説

Print # は、ポート番号で指定された通信ポートまたはデバイスに、変数データ, 数値, あるいは文字列を出力します。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

最大データ長

本命令が一度にあつかえる最大データ長は 256 Byte です。
 ただし、データベースが対象の場合は 4096 Byte が最大データ長になります。
 通信ポート(TCP/IP)が対象の場合は、1024 Byte が最大データ長になります。

他のコントローラーと変数のやりとりをする場合

- 複数の文字列変数、および、数値変数と文字列変数の両方を指定するときには、文字列データはデリミター (“;”) を明示的に追加する必要があります。
 コントローラー間で、通信ポートを使って文字列変数, 数値変数をやりとりする例

送信側 (どちらのパターンでも OK です。)

```
Print #PortNum, "$Status,", InData, OutData
Print #PortNum, "$Status", ",", InData, OutData
```

受信側

```
Input #PortNum, Response$, InData, OutData
```

ファイルへの書き込みはバッファリングされます

バッファリングされたデータは Flush ステートメントで書き込むことができます。Close ステートメントでファイルを閉じた場合も書き込まれます。

Print #命令は、Wait 命令、または、動作命令と一緒に使用してください**ループ文の中で Print #命令だけを使用しないでください**

ループ文の中で Print #命令だけを使用すると、コントローラーがハングアップ状態になる場合があります。

コントローラーの負荷状態によっては Wait 命令、または、動作命令を使用していても、情報が正しく表示されない場合があります。出力先が TP1 の場合は、Wait 時間を 1 (秒)以上、その他の場合は、0.1 (秒)以上設けてください。

悪い例

```
Do
    Print #24, "1234"
Loop
```

良い例

```
Do
    Print #24, "1234"
    Wait 1
Loop
```

参照

Input#, Print, Write, WriteBin

Print # 使用例

Print # を使った簡単な例です。

```
Function printex
    String temp$
    Print #1, "5"           ' “5”をポート 1 に出力 temp$ = "hello"
    Print #1, temp$
    Print #2, temp$
    Print #1 " Next message for " + Chr$(34) + "port 1" + Chr$(34)
    Print #2 " Next message for " + Chr$(34) + "port 2" + Chr$(34)
End
```

PTCLR

関節のピークトルクをクリアし、初期化します。

書式

PTCLR [関節指定 1 [, 関節指定 2 [, 関節指定 3 [, 関節指定 4 [, 関節指定 5 [, 関節指定 6 [, 関節指定 7 [, 関節指定 8 [, 関節指定 9]]]]]]]]]

パラメーター

関節指定 1 - 関節指定 9 関節番号を整数値または式で指定します。パラメーターを指定しない場合は、全関節のピークトルク値がクリアされます。
付加軸の S 軸は 8、T 軸は 9 になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

PTCLR は指定した関節のピークトルク値をクリアします。
PTRQ 実行前に、必ず PTCLR を実行してください。

参照

ATRQ, PTRQ

PTCLR 使用例

<例 1>

全関節のピークトルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

```
> ptclr
> go pl
> ptrq 1
    0.227
> ptrq
    0.227      0.118
    0.249      0.083
    0.000      0.000
>
```

<例 2>

垂直多関節ロボットにおいて、J1, J4, J5 のピークトルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

```
> ptclr 4, 1, 5
> go pl
> ptrq 1
    0.227
> ptrq 4
    0.083
```

PTPBoost

PTP (point to point)動作微小移動時の加減速アルゴリズム調整パラメーターの設定と表示を行います。

書式

- (1) PTPBoost 調整設定値 [, Jump 退避調整] [, Jump 接近調整]
- (2) PTPBoost

パラメーター

調整設定値	調整設定値 (0~100 の整数)を式または数値で指定します。
Jump 退避調整	Jump 動作中の Z 座標の退避調整設定値 (0~100 の整数)を式または数値で指定します。省略可能です。
Jump 接近調整	Jump 動作中の Z 座標の接近調整設定値 (0~100 の整数)を式または数値で指定します。省略可能です。

結果

パラメーターを省略すると、現在の PTPBoost 設定値を表示します。

解説

PTP 動作の微小距離移動時の加減速アルゴリズムを設定します。本設定値は移動距離が微小なときのみ有効です。目標座標への移動距離が微小かどうかは PTPBoostOK 関数で確認します。

PTPBoost は通常の使用では設定値を変更する必要はありません。微小移動動作時に少しでもサイクルタイムを短くしたい場合や、逆にサイクルタイムを長くしても残留振動を軽減させたいような場合にのみ使用して下さい。

設定値を大きくするとサイクルタイムは短くなりますが、停止時の振動は発生しやすくなります。また、設定値を小さくするとサイクルタイムは長くなりますが、停止時の振動は発生しにくくなります。停止時の振動が大きい状態でロボットを使用すると、エラーや衝撃を生じ、ロボットの機能が十分に発揮できないばかりでなく、マニピュレーターの寿命を縮める原因となりますのでご注意ください。

下記のいずれかの場合、PTPBoost 値は、初期化されます。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

参照

PTPBoost 関数, PTPBoostOK

PTPBoost 使用例

PTPBoost 50, 30, 30

PTPBoost 関数

指定された PTPBoost 値を返します。

書式

PTPBoost(パラメーター番号)

パラメーター

設定番号 以下に示す設定値を整数で設定します。

- 1: 調整設定値
- 2: Jump 退避調整設定値
- 3: Jump 接近調整設定値

戻り値

調整設定値 (0~100 の整数値)を返します。

参照

PTPBoost, PTPBoostOK

PTPBoost 関数使用例

```
Print PTPBoost(1)
```

PTPBoostOK 関数

現在位置から目標座標への PTP (point to point)動作が微小移動であるかどうかを返します。

書式

PTPBoostOK(目標座標)

パラメーター

目標座標 目標座標をポイントデータで指定します。

戻り値

現在位置から目標座標までの PTP 動作微小移動であれば“True”を、そうでない場合は“False”を返します。

解説

現在位置から目標座標への PTP 動作が、PTPBoost 命令による加減速アルゴリズム調整が有効な微小移動であるかどうかを確認します。

参照

PTPBoost

PTPBoostOK 関数使用例

```
If PTPBoostOK(P1) Then
    PTPBoost 50
EndIf
Go P1
```

PTPTime 関数

PTP 動作命令の推定所要時間を返します。PTP 動作は行いません。

書式

- (1) PTPTime(目標座標, 目標アーム, 目標ツール)
- (2) PTPTime(開始座標, 開始アーム, 開始ツール, 目標座標, 目標アーム, 目標ツール)

パラメーター

- 開始座標** 開始位置をポイント式で指定します。
- 目標座標** 目標位置をポイント式で指定します。
- 目標アーム** 目標位置のアーム番号を整数値または式で指定します。
- 目標ツール** 目標位置のツール番号を整数値または式で指定します。
- 開始アーム** 開始位置のアーム番号を整数値または式で指定します。
- 開始ツール** 開始位置のツール番号を整数値または式で指定します。

戻り値

実数値を秒単位で返します。

解説

PTPTime 関数を使って、PTP 動作命令 (Go)の所要時間の見積りができます。現在位置から目標位置までの所要時間の見積りには書式 (1)を、開始位置から目標位置までの所要時間の見積りには書式 (2)を使用してください。

この関数を実行しても、実際の PTP 動作は行われません。現在の位置設定, アーム設定, ツール設定は変更されません。

到達できない位置を設定したり、アームやツールを誤って設定すると、戻り値は 0 になります。

付加軸を含むロボットの場合、付加軸がサーボ軸で構成される場合は、付加軸の動作時間も加味されます。付加軸がパルス出力軸の場合は、ロボット単独の動作時間を返します。

参照

ATRQ, Go, PTRQ

PTPTime 関数使用例

```
Real secs

secs = PTPTime(P1, 0, 0, P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs

Go P1
secs = PTPTime(P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs
```

PTran

現在位置から指定したパルス量を、1 関節のみ PTP 動作します。

書式

PTran 関節番号, パルス

パラメーター

関節番号	移動する関節番号の整数値を指定します。 付加軸の S 軸は 8、T 軸は 9 になります。
パルス	移動するパルス量を指定します。

解説

Ptran は現在位置から指定したパルス量を、1 関節のみ移動します。

参照

Go, JTran, Jump, Move

Ptran 使用例

```
PTran 1, 2000
```

PTRQ

指定した関節のピークトルク値を表示します。

書式

PTRQ [関節番号]

パラメーター

関節番号 関節番号を整数値または式で指定します。省略可能です。
付加軸の S 軸は 8、T 軸は 9 になります。

結果

全関節について現在のピークトルク値を表示します。

解説

PTCLR を実行してから、PTRQ を実行し、関節のピークトルク値を表示してください。

ピークトルクは、0~1 までの実数値で表わされます。

参照

ATRQ, PTCLR, PTRQ 関数

PTRQ 使用例

```
> ptclr
> go pl
> ptrq 1
    0.227
> ptrq
    0.227    0.118
    0.249    0.083
    0.000    0.000
>
```

PTRQ 関数

指定した関節のピークトルクを返します。

書式

PTRQ(関節番号)

パラメーター

関節番号 関節番号を整数値または式で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

0~1 までの実数値で返します。

参照

ATRQ, PTCLR, PTRQ

PTRQ 関数使用例

プログラム中に PTRQ 関数を使用した例です。

```
Function DisplayPeakTorque
  Integer i

  Print "Peak torques:"
  For i = 1 To 4
    Print "Joint ", i, " = ", PTRQ(i)
  Next i
Fend
```

Pulse

各関節のパルス値で指定されたポイントまで、ロボットアームを **PTP** 動作で動かします。

書式

- (1) Pulse 第 1 関節パルス値, 第 2 関節パルス値, 第 3 関節パルス値, 第 4 関節パルス値[, 第 5 関節パルス値, 第 6 関節パルス値] [, 第 7 関節パルス値] [, 第 8 関節パルス値, 第 9 関節パルス値]
- (2) Pulse

パラメーター

第 1 から第 4 関節パルス値	最初に 4 つの各関節のパルス値を指定します。Range 命令で指定された範囲内の値を、整数または Long 式で指定します。
第 5 関節パルス値、第 6 関節パルス値	垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボットで使用します。省略可能です。
第 7 関節パルス値	ジョイント型 7 軸ロボットで使用します。省略可能です。
第 8 関節パルス値、第 9 関節パルス値	付加軸で使用します。省略可能です。

結果

パラメーター省略時は、現在のロボットの位置を示すパルス値が表示されます。

解説

Pulse では、ロボットアームの位置を表わすのに、直交座標系の座標ではなく、0 パルス位置からの各関節のパルス値を使います。Pulse 命令は、ロボットアームを **PTP** 動作で動かします。

Pulse 命令で使える上限値と下限値は、Range 命令で設定されます。

注意

Pulse 使用前に、軌道上に障害物がないことを確認してください

Jump と異なり、Pulse は、第 3 関節の上昇と目標座標までの下降を含め、全関節を同時に動かします。したがって Pulse 使用時には、ハンドが障害物にあたらないよう、十分注意してください。

起こりやすいエラー

制限値を超えるパルス値

Pulse 命令で指示されたパルス値が、Range で設定された制限値範囲を超えると、エラーになります。

参照

Go, Accel, Range, Speed, Pls, Pulse 関数

Pulse 使用例

下記はコマンドウィンドウからの操作例です。
各関節のパルス値で指定された位置までロボットアームを動かします。

```
> pulse 16000, 10000, -100, 10
```

現在のロボットアーム位置の、第 1 から第 4 関節までのパルス値を表示します。

```
> pulse  
PULSE: 1: 27306 pls 2: 11378 pls 3: -3072 pls 4: 1297 pls  
>
```


Pulse 関数

各関節にパルスで指定されたロボット座標値を返します。

書式

Pulse (第 1 関節パルス値, 第 2 関節パルス値, 第 3 関節パルス値, 第 4 関節パルス値
[, 第 5 関節パルス値, 第 6 関節パルス値] [, 第 7 関節パルス値] [, 第 8 関節パルス値, 第 9 関節パルス値])

パラメーター

第 1 から第 4 関節パルス値	第 1 関節から第 4 関節の各関節のパルス値を指定します。 Range 命令で指定された範囲内の値を、整数または Long 式で指定します。
第 5 関節パルス値、第 6 関節パルス値	垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボットで使用します。省略可能です。
第 7 関節パルス値	ジョイント型 7 軸ロボットで使用します。省略可能です。
第 8 関節パルス値、第 9 関節パルス値	付加軸で使用します。省略可能です。

戻り値

指定したパルスで使用するロボット座標値を返します。

参照

Go, JA, Jump, Move, Pulse, XY

Pulse 関数使用例

```
Jump Pulse(1000, 2000, 0, 0)
```

QP

クイックポーズ機能の設定、解除と、現在の設定の表示をします。

書式

- (1) QP { On | Off }
- (2) QP

パラメーター

On | Off クイックポーズの On (設定)または Off (解除)を指定します。

結果

パラメーターを省略すると、現在の QP 設定を表示します。

解説

動作コマンド実行時に、Pause スイッチが押されるか、コントローラーにポーズ入力があった場合、ロボットを直ちに停止させるか、動作コマンドの実行終了を待って停止させるか、クイックポーズ機能で設定します。

直ちに減速、停止させることを「クイックポーズさせる」といいます。

パラメーター“On”が指定されると、QP はクイックポーズに設定します。

パラメーター“Off”が指定されると、QP はクイックポーズを解除します。

QP は、ロボットがポーズ入力に反応して直ちに停止するか、あるいは動作を完了させてから停止するか、現在の設定状態を表示します。QP は、クイックポーズ機能が設定されているか解除されているか、単に状態を表示する命令です。

注意

クイックポーズ機能は電源オン時にはデフォルトで設定されています

QP 命令で設定したクイックポーズ機能設定は、Reset 命令実行後も保存されます。しかし、コントローラーまたはドライブユニットを電源オフした後、再度電源オンすると、クイックポーズ機能の設定はデフォルトの“オン(設定)”になります。

QP と安全扉入力

クイックポーズ機能の設定が解除されていても、安全扉入力が“開”になると、ロボットは直ちに停止します。

参照

Pause

QP 使用例

下記のコマンドウィンドウからの操作例は、ポーズ入力でロボットが直ちに停止するか(クイックポーズ機能が設定されているか)表示させます。

```
> QP
QP ON

> QP on 'クイックポーズモードに設定
>
```

QPDcelR

CP 動作時のツール姿勢変化について、クイックポーズ減速度を設定します。

書式

- (1) QPDcelR QP 減速度設定値
- (2) QPDcelR

パラメーター

QP 減速度設定値 CP 動作におけるクイックポーズ時の減速度を実数値で指定します。
(単位: deg/sec²)

結果

パラメーターを省略すると、現在の QPDcelR 設定値を表示します。

解説

QPDcelR は、Move, Arc, Arc3, BMove, TMove, Jump3CP において、ROT 修飾パラメーターを使っている時のみ有効となります。

上記動作中にクイックポーズが実行されたとき、関節過加速度エラーが発生することがあります。これは、通常のクイックポーズ動作で自動的に設定されるクイックポーズ減速度が関節許容減速度を超えてしまうために発生します。特に、CP 動作の AccelR 設定値が大きい場合やロボットの特異姿勢近傍を通過するケースで発生しやすくなります。このようなエラーが発生する場合には、QPDcelR でクイックポーズ減速度を低く設定してください。QPDcelR は小さくしすぎるとクイックポーズに必要な移動量が増加するので極力大きな値を設定してください。通常は QPDcelR を設定する必要はありません。

QPDcelR は AccelR で設定された CP 動作の姿勢変化減速度より小さな減速度は設定できません。その場合は、パラメーター範囲外エラーが発生します。

また、QPDcelR 設定後に、設定した QP 減速度設定値より大きな減速度が AccelR で設定されたとき、QPDcelR は AccelR で設定された減速度と同じ QP 減速度を自動的に設定します。

下記のいずれかの場合、QPDcelR 値は、デフォルトの最大減速度に初期化されます。

コントローラー起動時
Motor On 実行
SFree, SLock, Brake 実行
Reset, Reset Error 実行
停止ボタン, Quit All 実行などによるタスクの終了

参照

QPDcelR 関数, QPDcelS, AccelR

QPDcelR 使用例

下記は、Move 命令の QPDcelR を設定する簡単な動作プログラム例です。

```
Function QPDcelTest
  AccelR 3000
  QPDcelR 4000
  SpeedR 100
  Move P1 ROT
  :
  :
Fend
```

QPDceIIR 関数

CP 動作のツール姿勢変化についてのクイックポーズ減速度の設定値を返す関数です。

書式

QPDceIIR

戻り値

CP 動作のツール変化についてのクイックポーズ減速度設定値 (実数値、単位: deg/s²)を返します。

参照

QPDceIIR, QPDceIS 関数

QPDceIIR 関数使用例

```
Real savQPDceIIR  
  
savQPDceIIR = QPDceIIR
```

QPDDecelS

CP 動作時のクイックポーズ減速度を設定します。

書式

- (1) QPDDecelS QP 減速度設定値 [, Jump3 退避 QP 減速度設定値, Jump3 接近 QP 減速度設定値]
- (2) QPDDecelS

パラメーター

QP 減速度設定値 CP 動作におけるクイックポーズ時の減速度を、実数値で指定します。
(単位: mm/sec²)

Jump3 退避 QP 減速度設定値

Jump3 退避動作におけるクイックポーズ時の減速度を、実数値で指定します。
(単位: mm/sec²)

Jump3 接近 QP 減速度設定値

Jump3 接近動作におけるクイックポーズ時の減速度を、実数値で指定します。
(単位: mm/sec²)

結果

パラメーターを省略すると、現在の QPDDecelS 設定値を表示します。

解説

CP 動作中にクイックポーズが実行されたとき、関節過加速度エラーが発生することがあります。これは、通常のクイックポーズ動作で自動的に設定されるクイックポーズ減速度が、関節許容減速度を超えてしまうために発生します。特に、CP 動作の AccelS 設定値が大きい場合やロボットの特異姿勢近傍を通過するケースで発生しやすくなります。このようなエラーが発生する場合には、QPDDecelS でクイックポーズ減速度を低く設定してください。QPDDecelS は小さくしすぎるとクイックポーズに必要な移動量が増加するので極力大きな値を設定してください。通常は QPDDecelS を設定する必要はありません。

QPDDecelS は AccelS で設定された CP 動作の減速度より小さな減速度は設定できません。

その場合は、パラメーター範囲外エラーが発生します。

また、QPDDecelS 設定後に、設定した QP 減速度設定値より大きな減速度が AccelS で設定されたとき、QPDDecelS は AccelS で設定された減速度と同じ QP 減速度を自動的に設定します。

下記のいずれかの場合、QPDDecelS 値は、デフォルトの最大減速度に初期化されます。

コントローラー起動時
Motor On 実行
SFree, SLock, Brake 実行
Reset, Reset Error 実行
停止ボタン, Quit All 実行などによるタスクの終了

参照

QPDDecelS 関数, QPDDecelR, AccelS

QPDcelS 使用例

下記は、Move 命令の QPDcelS を設定する簡単な動作プログラム例です。

```
Function QPDcelTest
  AccelS 3000
  QPDcelS 4000
  SpeedS 100
  Move P1
  .
  .
  .
End
```

QPDceIS 関数

CP 動作のクイックポーズ減速度の設定値を返す関数です。

書式

QPDceIS(設定値番号)

パラメーター

設定値番号

下記のいずれかの値を、整数値または式で指定します。

- 1: CP 動作時のクイックポーズ減速度設定値
- 2: Jump3 時と Jump3CP 時の退避動作クイックポーズ減速度設定値
- 3: Jump3 時と Jump3CP 時の接近動作クイックポーズ減速度設定値

戻り値

クイックポーズ減速度設定値 (実数値、単位: mm/s²)を返します。

参照

QPDceIS, QPDceIR 関数

QPDceIS 関数使用例

```
Real savQPDceIS  
  
savQPDceIS = QPDceIS(1)
```

Quit

指定されたタスク、またはすべてのタスクの実行を終了します。

書式

Quit { タスク識別子 | All }

パラメーター

タスク識別子	タスク名またはタスク番号を整数値、または式で指定します。 タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。 タスク番号の指定 (整数) 通常タスク : 1 から 32 バックグラウンドタスク : 65 から 80 トラップタスク : 257 から 267
All	バックグラウンドタスク以外のすべてのタスクを終了する場合、指定します。

解説

Quit は現在実行中のタスクや、Halt で一時停止されていたタスクを終了します。

Quit は、指定されたタスクが NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)、バックグラウンドタスクの場合も、タスクの実行を終了します。また、Quit All ではこれらのタスクを含めてバックグラウンドタスク以外のすべてのタスクの実行を終了します。

Quit All を実行すると、ロボット制御パラメーターを下記の設定にします。

ロボット制御パラメーター

Speed と SpeedR, SpeedS の設定値	(初期値に初期化されます。)
Accel と AccelR, AccelS の設定値	(初期値に初期化されます。)
QPDecelR, QPDecelS の設定値	(初期値に初期化されます。)
LimZ パラメーターの設定値	(0 に初期化されます。)
CP パラメーターの設定値	(Off に初期化されます。)
SoftCP パラメーターの設定値	(Off に初期化されます。)
Fine の設定	(初期値に初期化されます。)
Power Low 設定	(ローパワーモードになります。)
PTPBoost の設定値	(初期値に初期化されます。)
TCLim, TCSpeed の設定値	(初期値に初期化されます。)
PgLSpeed の設定値	(初期値に初期化されます。)

参照

Exit, Halt, Resume, Xqt

Quit 使用例

下記は、10 秒後に 2 つのタスクを終了する例です。

```
Function main
  Xqt winc1    'タスク winc1 を開始
  Xqt winc2    'タスク winc2 を開始
  Wait 10
  Quit winc1   'タスク winc1 を終了
  Quit winc2   'タスク winc2 を終了
Fend

Function winc1
  Do
    On 1; Wait 0.2
    Off 1; Wait 0.2
  Loop
Fend

Function winc2
  Do
    On 2; Wait 0.5
    Off 2; Wait 0.5
  Loop
Fend
```

RadToDeg 関数

ラジアンを角度に変換します。

書式

RadToDeg(ラジアン)

パラメーター

ラジアン 角度に変換するラジアンを実数値で指定します。

戻り値

角度を表わす Double 型の値を返します。

参照

ATan, ATan2, DegToRad 関数

RadToDeg 関数使用例

```
s = Cos (RadToDeg (x) )
```

Randomize

乱数系列の初期化を行います。

書式

- (1) Randomize シード値
- (2) Randomize

パラメーター

シード値 乱数を求めるための基となる値を 0 以上の実数値で指定します。

参照

Rnd 関数

Randomize 使用例

```
Function main
  Real r
  Randomize
  Integer randNum

  randNum = Int(Rnd(10)) + 1
  Print "Random number is:", randNum
End
```

Range

各サーボ関節の許容動作エリアを設定、表示します。

書式

- (1) Range 設定値 1, 設定値 2, 設定値 3, 設定値 4, 設定値 5, 設定値 6, 設定値 7, 設定値 8
[, 設定値 9, 設定値 10, 設定値 11, 設定値 12]
[, 設定値 13, 設定値 14]
[, 設定値 15, 設定値 16, 設定値 17, 設定値 18]
- (2) Range

パラメーター

設定値 1	第 1 関節の下限パルス値 (単位: パルス)
設定値 2	第 1 関節の上限パルス値 (単位: パルス)
設定値 3	第 2 関節の下限パルス値 (単位: パルス)
設定値 4	第 2 関節の上限パルス値 (単位: パルス)
設定値 5	第 3 関節の下限パルス値 (単位: パルス)
設定値 6	第 3 関節の上限パルス値 (単位: パルス)
設定値 7	第 4 関節の下限パルス値 (単位: パルス)
設定値 8	第 4 関節の上限パルス値 (単位: パルス)
設定値 9	第 5 関節の下限パルス値 (単位: パルス) 垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボット専用パラメーターです。
設定値 10	第 5 関節の上限パルス値 (単位: パルス) 垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボット専用パラメーターです。
設定値 11	第 6 関節の下限パルス値 (単位: パルス) 垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボット専用パラメーターです。
設定値 12	第 6 関節の上限パルス値 (単位: パルス) 垂直 6 軸型ロボット (N シリーズを含む)、およびジョイント型 6 軸ロボット専用パラメーターです。
設定値 13	第 7 関節の下限パルス値 (単位: パルス) ジョイント型 7 軸ロボット専用パラメーターです。
設定値 14	第 7 関節の上限パルス値 (単位: パルス) ジョイント型 7 軸ロボット専用パラメーターです。
設定値 15	第 8 関節の下限パルス値 (単位: パルス) 付加軸 S 関節専用パラメーターです。
設定値 16	第 8 関節の上限パルス値 (単位: パルス) 付加軸 S 関節専用パラメーターです。
設定値 17	第 9 関節の下限パルス値 (単位: パルス) 付加軸 T 関節専用パラメーターです。
設定値 18	第 9 関節の上限パルス値 (単位: パルス) 付加軸 T 関節専用パラメーターです。

結果

パラメーターを省略すると、現在の Range 値を表示します。

解説

Range は、各モーター関節の下限と上限パルス値を設定します。単位はパルスです。これによりユーザーは、各関節ごとに最大および最小許容動作範囲を定義することができます。同様に、XY 座標方向の制限値は、XYLim 命令を使って定義します。

Range の初期値はロボットの機種ごとに異なります。この命令で設定された値は、電源オフ後も保存されます。

パラメーター省略時は、現在の Range 値が表示されます。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

起こりやすいエラー

許容範囲外に動かそうとしたとき

1 関節でも、ロボットアームの許容動作範囲を超えて動かそうとすると、エラーになります。

関節が動かない

下限パルス値が、上限パルス値と同じか大きい場合、関節は動きません。

注意

第 6 関節の上限パルス値と下限パルス値の設定範囲は、機種によって異なります

C4 : -419430399 ~ 419430399
C8, C12, N2, N6 : -26847955 ~ 26847955

参照

JRange, SysConfig, XYLim

Range 使用例

コマンドウィンドウから現在の Range 設定値の表示と変更を行う簡単な例です。

```
> range
-18205, 182045, -82489, 82489, -36864, 0, -46695, 46695
>
> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

Read

ファイル、または通信ポートから指定した文字数を読み込みます。

書式

Read #ポート番号, 文字列変数\$, 文字数

パラメーター

ポート番号	ファイル、または通信ポートを示す ID 番号です。 ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。 通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。
文字列変数\$, 文字数	文字列を受信する文字列変数名を指定します。 読み込むバイト数を指定します。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

参照

ChkCom, ChkNet, OpenCom, OpenNet, Write, ReadBin

Read 使用例

```
Integer numOfChars
String data$

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    Read #1, data$, numOfChars
EndIf
```

ReadBin

ファイル、または通信ポートからバイナリーデータを読みます。

書式

ReadBin #ポート番号, 変数名

ReadBin #ポート番号, 配列変数名(), バイト数

パラメーター

ポート番号	ファイル、または通信ポートを示す ID 番号です。 ファイル番号は、BOpen などのステートメントで指定した番号です。 通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。
変数名	データバイトを受け取る、Byte 型変数、整数変数、または Long 型変数の名前を指定します。
配列変数名()	データバイトを受け取る、Byte 型変数、整数変数、または Long 型変数の名前を指定します。指定可能なのは 1 次元の配列変数です。
バイト数	読み込むバイト数を指定します。 配列の最大要素番号以下で、かつ 256Byte 以下である必要があります。 通信ポート (TCP/IP) が対象の場合は、配列の最大要素番号以下で、かつ 1024Byte 以下である必要があります。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

参照

Write, WriteBin, Read

ReadBin 使用例

```
Integer data
Integer dataArray(10)

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    ReadBin #1, data
EndIf

numOfChars = ChkCom(1)

If numOfChars > 10 Then
    ReadBin #1, dataArray(), 10
EndIf
```

Real

Real 型の変数を宣言します。(4 バイトの実数値)

書式

Real 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 Real 型に宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Real は Real 型変数の宣言をします。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

Real 型の有効桁数は 6 桁です。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Short, String, UByte, UInt32, UInt64, UShort

Real 使用例

次の例は、Real を使って Real 型の変数を宣言するプログラムです。

```
Function realtest
  Real var1
  Real A(10)           'Real 型の 1 次元配列
  Real B(10, 10)       'Real 型の 2 次元配列
  Real C(5, 5, 5)      'Real 型の 3 次元配列
  Real arrayVar(10)
  Integer i
  Print "Please enter a Real Number:"
  Input var1
  Print "The Real variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter a Real Number:"
    Input arrayVar(i)
    Print "Value Entered was ", arrayVar(i)
  Next i
End
```


RealAccel 関数

OLAccel 調整後の加減速度の設定値を返す関数です。

書式

RealAccel(設定値番号)

パラメーター

設定値番号 整数値で下記のそれぞれの値を指定します。

- 1: 加速設定値
- 2: 減速設定値
- 3: Jump 動作時の退避加速設定値
- 4: Jump 動作時の退避減速設定値
- 5: Jump 動作時の接近加速設定値
- 6: Jump 動作時の接近減速設定値

戻り値

1 以上の整数 (%)を返します。

用途

RealAccel を使用することにより、ロボットが連続動作できる最大の加減速度を知ることができます。
手順を下記に記載します。

- (1) OLAcel コマンドを On の状態でロボットを動作させます。
- (2) OLRate コマンドを実行し、過負荷率が上昇するか確認します。
- (3) 過負荷率が上昇する場合、過負荷率が 0.5 以上になった時点から加減速度の自動調整が開始されます。
- (4) 一定時間経過後、OLRate コマンドを実行し過負荷率が上昇しないことを確認してください。
- (5) 過負荷率が上昇しないことを確認した後、RealAccel 関数を実行します。
- (6) RealAccel 関数が返した値が、(1)の動作でロボットが連続動作できる最大の加減速度です。
 - 過負荷率が上昇している最中に RealAccel 関数を使用した場合、ロボットが連続動作できる最大の加減速度を知ることができません。
 - オーバーヒートエラーが発生する場合、上記の手順を行ってもロボットが連続動作できる最大の加減速度を知ることができません。

参照

Accel, OLAcel, OLRate

RealAccel 関数使用例

プログラム中に RealAccel 関数を使う例です。

```
Integer RealAccel1, RealDecel1
```

```
Accel 100, 100
```

```
OLAccel on
```

・現在の加減速度を取得

```
RealAccel1 = RealAccel (1)
```

```
RealDecel1 = RealAccel (2)
```

現在の加速度を表示

```
Print RealAccel1
```

現在の減速度を表示

```
Print RealDecel1
```

RealPls 関数

指定した関節のパルス値を返します。

書式

RealPls(関節番号)

パラメーター

関節番号 現在のパルス値を返す関節を指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

関節番号で指定した関節の、現在のエンコーダーパルス値を整数値で返します。

解説

各関節の現在のエンコーダー位置またはパルス値を **RealPls** 関数で返します。この値は保存し、**Pulse** コマンドで使用できます。

参照

CX, CY, CZ, CU, CV, CW, Pulse

RealPls 関数使用例

```
Function DisplayPulses
    Long joint1Pulses

    joint1Pulses = RealPls(1)
    Print "Joint 1 Current Pulse Value: ", joint1Pulses
End
```

RealPos 関数

指定したロボットの現在の位置を返します。

書式

RealPos

戻り値

指定したロボットの現在位置のポイントを返します。

解説

RealPos 関数はロボットの現在位置を返します。

参照

CurPos, CX, CY, CZ, CU, CV, CW, RealPls

RealPos 関数使用例

```
Function ShowRealPos  
  
    Print RealPos  
End  
  
P1 = RealPos
```

RealTorque 関数

指定した関節の現在のトルク指令値を返します。

書式

RealTorque(関節番号)

パラメーター

関節番号 トルク指令値を取得したい関節の関節番号を式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

現在のパワーモードにおける最大トルクに対する割合を-1~1 の実数値で返します。
正值は関節角の正方向を表し、負値は関節角の負方向を表します。

参照

TC, TCSpeed, TCLim

RealTorque 関数使用例

```
Print "現在の Z 軸トルク指令値 (スカラロボット) :", RealTorque (3)
```

Recover

安全扉が開いた時点の位置へ復帰動作を実行し、状態を返します。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

- (1) Recover ロボット番号 | All
- (2) Recover ロボット番号 | All , WithMove | WithoutMove

パラメーター

ロボット番号	復帰動作を行うロボット番号を指定します。
All	すべてのロボットで復帰動作を実行します。 省略時は All となります。
WithMove	値 0 の定数です。 励磁の復帰と安全扉が開いた時点の位置への移動を行います。 省略時は WithMove となります。
WithoutMove	値 1 の定数です。 励磁の復帰のみを行います。通常は使用しません。 AbortMotion と組み合わせて特殊な復帰を実現します。

解説

本命令をプログラムから実行するには、EPSON RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスをチェックする必要があります。

Recover は、安全扉が閉じられてから、モーターを再度オンし、ロボットを安全扉が開いた時点の位置までローパワーの PTP 動作で戻ります。復帰動作が完了すると、Cont を用いてサイクルを継続させることができます。

コントローラーに複数のロボットが設定されており、All が指定された場合は、ロボット番号が小さい順に復帰動作を実行します。

参照

AbortMotion, Cont, Recover 関数, RecoverPos

Recover 使用例



注意

- プログラムからRecoverコマンドを実行する場合、コマンドの仕様を理解して、システムとして復帰動作可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

```
Function main
  Xqt 2, monitor, NoPause
  Do
    Jump P1
    Jump P2
  Loop
Fend

Function monitor
  Do
    If Sw(SGOpenSwitch) = On then
      Wait Sw(SGOpenSwitch) = Off and Sw(RecoverSwitch) = On
      Recover All
    EndIf
  Loop
Fend
```

Recover 関数

安全扉が開いた時点の位置へ復帰動作を実行し、状態を返します。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

- (1) Recover
- (2) Recover(ロボット番号 | All)
- (3) Recover(ロボット番号 | All , WithMove | WithoutMove)

パラメーター

ロボット番号	復帰動作を行うロボット番号を指定します。 ロボット番号を省略した場合は、すべてのロボットで復帰動作を実行します。
All	すべてのロボットで復帰動作を実行します。 省略時は All となります。
WithMove	値 0 の定数です。 励磁の復帰と安全扉が開いた時点の位置への移動を行います。 省略時は WithMove となります。
WithoutMove	値 1 の定数です。 励磁の復帰のみを行います。通常は使用しません。 AbortMotion と組み合わせて特殊な復帰を実現します。

戻り値

Boolean 型の値を返します。復帰動作が完了すると、“True”を返します。完了していなければ、“False”を返します。

解説

本命令をプログラムから実行するには、EPSON RC+ の[セットアップ]–[システム設定]–[コントローラー]–[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。

Recover は、安全扉が閉じられてから、モーターを再度オンし、ロボットを安全扉が開いた時点の位置までローパワーの PTP 動作で戻ります。復帰動作が完了すると、Cont を用いてサイクルを継続させることができます。復帰動作が問題なく完了すると、“True”が返されますが、復帰動作中に、一時停止、中断、または安全扉が開かれると、Recover は、“False”を返します。

コントローラーに複数のロボットが設定されており、All が指定された場合は、ロボット番号が小さい順に復帰動作を実行します。



注意

- プログラムからRecoverコマンドを実行する場合、コマンドの仕様を理解して、システムとして復帰動作可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

AbortMotion, Cont, Recover, RecoverPos

Recover 関数 使用例

```
Boolean sts
Integer answer

sts = Recover
If sts = True Then
    MsgBox "Ready to continue", MB_ICONQUESTION + MB_YESNO,
    "MyProject", answer
    If answer = IDYES Then
        Cont
    EndIf
EndIf
```

RecoverPos 関数

安全扉が開いた時点の位置を返します。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

RecoverPos ([ロボット番号])

パラメーター

ロボット番号 ロボットの番号を整数値で指定します。
 ロボット番号を省略した場合、現在選択されているロボットが対象となります。

結果

安全扉が開いた時点の位置を返します。
安全扉が開いていない場合や、ロボットが復帰動作を終了している場合は、X から W の値に 0 が入ったポイントデータを返します。

解説

Cont または Recover で復帰動作を行う場合のロボットの復帰位置を返します。

参照

AbortMotion, Cont, Recover, Recover 関数, RealPos

RecoverPos 関数 使用例

復帰動作の直線距離が 10mm 未満の場合は復帰動作を行い、それ以上場合はプログラムを終了します。

```
If Dist(RecoverPos, RealPos) < 10 Then  
    Recover All  
Else  
    Quit All  
EndIf
```

Redim

ランタイム時に配列の最大要素番号を変更します。

書式

Redim [Preserve] 配列名(配列変数の最大要素番号)

パラメーター

Preserve 配列の前の値を保存します。省略可能で、省略すると配列はクリアされます。

配列名 配列変数の名前を指定します。通常の変数名の慣例にしたがって指定します。あらかじめ配列が宣言されていないとエラーになります。

配列変数の最大要素番号

配列変数の新しい最大要素番号を指定します。変数宣言時と同じ数の最大要素番号を与えてください。次の書式を用います。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。

各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

	String型以外	String型
ローカル変数	2,000	200
バックアップ変数 (Global Preserve)	4,000	400
グローバル変数とモジュール変数	100,000	10,000

解説

Redim は、ランタイム時に配列変数の最大要素番号を変更するときに使います。前の値を保存するときは、Preserve を指定します。

Byref 指定された配列変数は Redim することはできません。

頻繁な Redim の実行は、プログラムの実行速度を低下させます。特にバックアップ変数に対する Redim の実行は必要最小限にすることを推奨します。

参照

UBound

Redim 使用例

```
Integer i, numParts, a(0)

Print "Enter number of parts "
Input numParts

Redim a(numParts)

For i = 0 to UBound(a)
    a(i) = i
Next

'最大要素番号を20追加します
Redim Preserve a(numParts + 20)

'最初の要素番号を保持
For i = 0 to UBound(a)
    Print a(i)
Next
```

Rename

ファイル名を変更します。

書式

Rename 変更元ファイル名, 変更先ファイル名

パラメーター

変更元ファイル名	名前を変更するファイル名とパスの文字列式を指定します。 詳細は、ChDisk を参照してください。
変更先ファイル名	変更元ファイル名で指定したファイルに、新たにつける名前を指定します。 詳細は、ChDisk を参照してください。

解説

変更元ファイル名で指定したファイルの名前を、変更先ファイル名で指定する名前に変更します。
パスを省略すると、このステートメントは、カレントディレクトリーから変更元ファイル名を探します。
変更元ファイル名と変更先ファイル名が、同一ドライブ内で指定されている場合のみ実行可能です。
同じパスに存在する他のファイルと同じ名前に変更することはできません。
ワイルドカードとして使われる文字は、変更元ファイル名、変更先ファイル名のパラメーターには使えません。

参照

Copy

Rename 使用例

コマンドウィンドウでの実行例

```
> Rename A.PRG B.PRG
```

RenDir

ディレクトリー名を変更します。

書式

RenDir 変更元ディレクトリー名, 変更先ディレクトリー名

パラメーター

変更元ディレクトリー名 変更するディレクトリーのパスとディレクトリー名を文字列式で指定します。
変更先ディレクトリー名 変更後のディレクトリーのパスとディレクトリー名を文字列式で指定します。
パスの詳細は、ChDisk を参照してください。

解説

変更先ディレクトリーのパスは、変更元ディレクトリーのパスに存在しなければなりません。

変更元ディレクトリー名、変更先ディレクトリー名のパスを省略し、ディレクトリー名だけを指定した場合、カレントディレクトリーにあるディレクトリーを指します。

ワイルドカードとして使われる文字は、変更元ディレクトリー名、変更先ディレクトリー名のパラメーターには使えません。

注意

- ディスクが PC の場合に実行可能です。
-

参照

MkDir

RenDir 使用例

```
RenDir "c:\mydata", "c:\mydata1"
```

Reset

コントローラーを初期状態にリセットします。

書式

- (1) Reset
- (2) Reset Error

解説

Reset は、下記をリセットします。

Reset Error は、すべての通常タスクを終了して、エラー状態とロボット制御パラメーターのみリセットします。

Reset Error 命令をプログラムから実行するには、EPSON RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。

非常停止状態 (Reset 時のみ)

エラー状態

出力ビット (Reset 時のみ)

リモート出力に割り当てられている I/O と、ハンドに割り当てられている I/O を除く、すべての出力ビットはオフになります。

この機能は EPSON RC+ から解除することができます。

ロボット制御パラメーター

Speed と SpeedR, SpeedS の設定値	(初期値に初期化されます。)
Accel と AccelR, AccelS の設定値	(初期値に初期化されます。)
QPDcelR, QPDcelS の設定値	(初期値に初期化されます。)
LimZ パラメーターの設定値	(0 に初期化されます。)
CP パラメーターの設定値	(Off に初期化されます。)
SoftCP パラメーターの設定値	(Off に初期化されます。)
Fine の設定	(初期値に初期化されます。)
Power Low 設定	(ローパワーモードになります。)
PTPBoost の設定値	(初期値に初期化されます。)
TCLim, TCSpeed の設定値	(初期値に初期化されます。)
PgLSpeed の設定値	(初期値に初期化されます。)

サーボ関連のエラーや非常停止状態、その他のリセットが必要となる状態では、Reset 以外のコマンドは受けつけられません。この場合、まず Reset を実行してから、その他の必要な処理を行ってください。

例えば、非常停止の後では、まず周囲と操作の安全を確認し、Reset を実行します。その後、Motor On を実行してください。

重大エラー状態は、Reset では解除されません。

重大エラーが発生した場合、コントローラーの電源をオフし、エラーの原因を取りのぞいてください。

Reset 命令は、バックグラウンドタスクや Trap Emergency, Trap Error で起動されるタスクなどから実行することはできません。非常停止状態は、プログラムから解除することはできません。

注意

[Reset で出力ポートを Off]チェックボックス

EPSON RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[Reset で出力ポートを Off]チェックボックスがチェックされている場合は、Reset 命令が発行されると、ハンドに設定されたビットを除く、すべての出力はオフになります。この設定により、出力オフになったとき、ワークを落下させたり、それと似たような状況が生じないように、配線に配慮することが重要です。
ハンドについては、Hand 機能マニュアルを参照してください。

参照

Accel, AccelS, Fine, LimZ, Motor, Off, On, PTPBoost, SFree, SLock, Speed, SpeedS

Reset 使用例

Reset 命令をコマンドウィンドウから実行する例です。

```
>reset  
>
```


ResetElapsedTime

ElapsedTime 関数で使用するタクトタイム計測用タイマーをリセットします。

書式

ResetElapsedTime

解説

タクトタイム計測用タイマーをリセットし、スタートします。

参照

ElapsedTime 関数

ResetElapsedTime 使用例

```
ResetElapsedTime      'タクトタイム計測用タイマーをリセット
For i = 1 To 10        '10 回実行
    GoSub Cycle
Next
Print ElapsedTime / 10 'タクトタイムを計算して表示
```

Restart

現在のメインプログラムを再実行します。
本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

Restart

解説

Restart は実行中のタスクをすべて中断し、最後に実行していたメインプログラムを再実行します。
バックグラウンドタスクは中断せず、そのまま継続実行します。
すべての Trap 設定は解除されるため、本命令でタスクを中断しても Trap Abort は実行されません。
Pause 状態は本命令を実行することで解除されます。
エラー状態で本命令を実行するとエラーになります。Reset Error 命令などで、まずはエラーを解除してください。
非常停止状態で本命令を実行するとエラーになります。非常停止状態はプログラムから解除することはできません。



注意

■ プログラムからRestartコマンドを実行する場合、コマンドの仕様を理解して、システムとして再実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

注意

リモート I/O コントロールを使用している場合、SPEL+プログラムの Restart 命令とリモート入力 Start 信号を同じタイミングで実行しないでください。プログラムが 2 重で実行され、2503 エラーが発生する可能性があります。

参照

Quit, Reset, Trap, Xqt

Restart 使用例

```
Function main
  Trap Error Xqt eTrap
  Motor On
  Call PickPlac
Fend

Function eTrap
  Wait Sw(ERresetSwitch)
  Reset Error
  Wait Sw(RestartSwitch)
  Restart
Fend
```

Resume

Halt 命令により一時停止中のタスクを継続実行します。

書式

Resume { タスク識別子 | All }

パラメーター

タスク識別子	タスク名またはタスク番号を整数値、または式で指定します。 タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。 タスク番号の指定 (整数) 通常タスク : 1~32 バックグラウンドタスク : 65~80 トラップタスク : 257~267
All	すべてのタスクを継続実行するとき指定します。

解説

Resume は、Halt 命令により一時停止中のタスクを継続実行します。

参照

Halt, Quit, Xqt

Resume 使用例

Halt 命令の後に、Resume 命令を使用した例です。

```
Function main
  Xqt 2, flicker      'flicker をタスク 2 で実行

  Do
    Wait 3            'flicker を 3 秒間実行
    Halt flicker      'flicker タスクを停止
    Wait 3
    Resume flicker    'flicker タスクを一時停止
  Loop
Fend

Function flicker
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

Return

Return ステートメントは、GoSub ステートメントと組み合わせて使います。GoSub はプログラム制御をサブルーチンに移します。サブルーチンが完了すると、Return で、サブルーチンを始めた GoSub 命令の次の行にプログラム実行が継続するようにします。

書式

Return

解説

Return ステートメントは GoSub ステートメントと組み合わせて使います。Return ステートメントの主要な目的は、サブルーチンに制御を移した GoSub 命令の後に続く命令に、プログラム制御を戻すことです。

GoSub 命令は、プログラム制御をユーザー指定のステートメント行、またはラベルに枝分かれさせます。プログラムは、その移行先の行とそれに続く行を、Return 命令まで実行していきます。Return 命令は、サブルーチンへの移行を指示した GoSub の次の行に、プログラム制御を戻します。(つまり GoSub 命令はサブルーチンの実行を開始し、Return でその GoSub の次のステートメントに戻ります。)

起こりやすいエラー

GoSub がないのに Return を使用した場合

Return 命令は、サブルーチンから GoSub を発行した元のプログラムに戻るために使われます。GoSub がないのに Return 命令が使われるとエラー2383が発生します。システムはどこに戻ってよいのか分からないので、単独の Return 命令は意味がありません。

参照

OnErr, GoSub, GoTo

Return 使用例

下記は、GoSub 命令で、checkio というラベルに枝分かれして、最初の 16 個のユーザー入力をチェックする簡単な例です。その後、そのサブルーチンからメインプログラムに戻ります。

```
Function main
  Integer var1, var2
  GoSub checkio
  On 1
  On 2
  Exit Function

checkio:      'サブルーチンの開始位置
  var1 = In(0)
  var2 = In(1)
  If var1 <> 0 Or var2 <> 0 Then
    Print "Message to Operator here"
  EndIf
finished:
  Return      'サブルーチンの終了位置 40 行目に戻る
Fend
```

Right\$ 関数

文字列の最後から、指定数の文字列を取り出す関数です。

書式

Right\$(文字列, 文字数)

パラメーター

文字列 指定数の文字列を最後から取り出します。最長 255 文字の文字列式または直接文字列を指定します。

文字数 文字列の最後からコピーしてくる文字数 (正の整数) を、式または直接数値で指定します。

戻り値

指定した文字列の最後から、指定数の文字列を取り出して返します。

解説

Right\$ は指定した文字列の最後から指定数の文字列を返します。Right\$ は、直接文字列にあるだけの文字数を返すことができます。

参照

Asc, Chr\$, InStr, Left\$, Len, Mid\$, Space\$, Str\$, Val

Right\$ 使用例

下記は、ワークのデータ文字列を入力すると、ワークのパーツ番号, 名前, パーツカウントなどの情報を返すプログラムの例です。

```
Function SplitPartData(DataIn$ As String, ByRef PartNum$ As String,
ByRef PartName$ As String, ByRef PartCount As Integer)

    PartNum$ = Left$(DataIn$, 10)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Mid$(DataIn$, 11, 10)

    PartCount = Val(Right$(DataIn$, 5))

End
```

また、下記はコマンドウィンドウからの操作例です。

```
> Print Right$("ABCDEFGH", 2)
FG

> Print Right$("ABC", 3)
ABC
```

Rmdir

サブディレクトリーを削除します。

書式

Rmdir ディレクトリー名

パラメーター

ディレクトリー名 削除するディレクトリーのパスと名前を文字列式で指定します。
パスを省略しディレクトリー名だけを指定した場合、カレントディレクトリーにあるサブディレクトリーを指します。
パスの詳細は、**ChDisk** を参照してください。

解説

指定したサブディレクトリーを削除します。このステートメントを実行する前に、サブディレクトリー内のファイルを削除しておかなければなりません。

カレントディレクトリーや親ディレクトリーは削除することができません。

コマンドウィンドウから実行すると、引用符は省略することができます。

注意

- ディスクが PC の場合に実行可能です。
-

Rmdir 使用例

コマンドウィンドウでの実行例

```
> Rmdir \mydata
```

Rnd 関数

乱数を返します。

書式

Rnd(最大値)

パラメーター

最大値 最大の戻り値を実数値で設定します。

戻り値

0 から、パラメーターで指定した最大値までの、実数値の乱数を返します。

解説

乱数を生成するために使います。

参照

Int, Randomize

Rnd 関数使用例

1~10 までの乱数を生成する例です。

```
Function main
  Real r
  Integer randNum

  Randomize
  randNum = Int(Rnd(9)) + 1
  Print "Random number is:", randNum
Fend
```

Robot

ロボットを選択します。

書式

Robot ロボット番号

パラメーター

ロボット番号 ロボット番号を指定します。範囲は 1 から、設置されているロボット数です。

解説

Robot ステートメントは、次に動作命令を実行するロボットを選択します。

ロボットが 1 台の場合、Robot ステートメントを使用する必要はありません。

参照

Accel, AccelS, Arm, ArmSet, Go, Hofs, Home, HOrdr, Local, Move, Pulse, Robot 関数, Speed, SpeedS

Robot 使用例

```
Function main
  Integer I
  For I = 1 to 100
    Robot 1
    Go P(i)
    Robot 2
    Go P(i)
  Next I
Fend
```


Robot 関数

現在のロボット番号を返します。

書式

Robot

戻り値

現在のロボット番号の整数値を返します。

参照

Robot

Robot 関数使用例

```
Print "The current robot is: ", Robot
```

RobotInfo 関数

ロボットの状態情報を返します。

書式

RobotInfo(インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報の整数値を返します。

解説

戻り値のビット情報は次の表のとおりです。

インデックス	ビット	値	解説
0	0	&H1	未定義
	1	&H2	リセットできるエラー発生
	2	&H4	リセットできないエラー発生
	3	&H8	モーターON
	4	&H10	パワーHigh
	5	&H20	未定義
	6	&H40	未定義
	7	&H80	未定義
	8	&H100	ロボットは Halt 状態
	9	&H200	ロボットは Halt 状態ではない (動作中またはクイックポーズ中)
	10	&H400	一時停止または安全扉で、ロボット停止
	11		未定義
	12		未定義
	13		未定義
	14	&H4000	動作コマンドの後、TILL 条件を満たした
	15	&H8000	動作コマンドの後、SENSE 条件を満たした
	16-31		未定義
1	0	&H1	追従動作中 (コンベヤトラッキング中)
	1	&H2	復帰動作待ち (WaitRecover 状態)
	2	&H4	復帰動作実行中
	3-31		未定義
2	0	&H1	ロボットはホーム位置
	1-31		未定義
3	0	&H1	第 1 関節サーボ励磁中
	1	&H2	第 2 関節サーボ励磁中
	2	&H4	第 3 関節サーボ励磁中
	3	&H8	第 4 関節サーボ励磁中
	4	&H10	第 5 関節サーボ励磁中
	5	&H20	第 6 関節サーボ励磁中
	6	&H40	第 7 関節サーボ励磁中
	7	&H80	S 関節サーボ励磁中
	8	&H100	T 関節サーボ励磁中
	9-31		未定義

インデックス	ビット	値	解説
4	N/A	0 - 32 -1	ロボットコマンドを実行しているタスク番号 0 = コマンドウィンドウまたはマクロからコマンドを実行 -1 = タスクはマニピュレーターを未使用
5	0	&H1	第 1 関節ブレーキ ON
	1	&H2	第 2 関節ブレーキ ON
	2	&H4	第 3 関節ブレーキ ON
	3	&H8	第 4 関節ブレーキ ON
	4	&H10	第 5 関節ブレーキ ON
	5	&H20	第 6 関節ブレーキ ON
	6	&H40	第 7 関節ブレーキ ON
	7	&H80	S 関節ブレーキ ON
	8	&H100	T 関節ブレーキ ON
	9-31		未定義

参照

CtrlInfo, RobotInfo\$, TaskInfo

RobotInfo 関数使用例

```
If (RobotInfo(3) And &H1) = &H1 Then
  Print "Joint 1 is locked"
Else
  Print "Joint 1 is free"
EndIf
```

RobotInfo\$ 関数

ロボットのテキスト情報を返します。

書式

RobotInfo\$(インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報の文字列を返します。

解説

インデックス	解説
0	ロボット名
1	型名
2	デフォルトポイントファイル名
3	未定義
4	ロボットのシリアル番号

参照

CtrlInfo, RobotInfo, TaskInfo

RobotInfo\$ 関数使用例

```
Print "Robot Name: ", RobotInfo$(0)
```

RobotModel\$関数

ロボットの型名を返します。

書式

RobotModel\$

戻り値

型名の文字列を返します。これはロボットの後方パネルに記載されている、ロボットの名前です。

参照

RobotType

RobotModel\$ 関数使用例

```
Print "The robot model is ", RobotModel$
```

RobotName\$ 関数

ロボット名を返します。

書式

RobotName\$

戻り値

ロボット名を文字列で返します。

参照

RobotInfo, RobotModel\$

RobotName\$ 関数使用例

```
Print "The robot name is ", RobotName$
```

RobotSerial\$ 関数

ロボットのシリアル番号を返します。

書式

RobotSerial\$

戻り値

ロボットのシリアル番号を文字列で返します。

参照

RobotInfo, RobotName\$, RobotModel\$

RobotSerial\$ 関数使用例

```
Print "The robot serial number is ", RobotSerial$
```

RobotType 関数

ロボットタイプを返します。

書式

RobotType

戻り値

- 1: ジョイント型
- 2: 直角座標型
- 3: 水平多関節型
- 5: 垂直 6 軸型
- 6: RS シリーズ
- 7: N シリーズ

参照

RobotModel\$

RobotType 関数使用例

```
If RobotType = 3 Then
  Print "Robot type is SCARA"
EndIf
```


ROpen

ファイルを読み取り専用モードで開きます。

書式

ROpen ファイル名 As #ファイル番号

.

Close #ファイル番号

パラメーター

ファイル名	パスを含んだファイル名の文字列を指定します。 ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。 詳細は、ChDisk を参照してください。
ファイル番号	30~63 の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは指定したファイルからデータを読み出すときに使います。

注意

-
- PC ディスクのみ
 - ネットワークパスが使用できます。
-

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号を使うことはできません。ファイル番号は、ファイル操作命令 (Input#, Read, Seek, Eof, Close) で使用します。

Close ステートメントでファイルを閉じし、ファイル番号を解放します。

複数のタスクで同一の番号を使用しないように、FreeFile 関数でファイル番号を取得してください。

参照

Close, Input #, AOpen, BOpen, UOpen, WOpen, FreeFile

ROpen 使用例

```
Integer fileNum, i, j

fileNum = FreeFile
WOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

ROKOK 関数

目標座標への動作命令時に、ROT 修飾パラメーターが付加可能かどうかを返します

書式

ROKOK (目標座標)

ROKOK (基準座標, 目標座標)

パラメーター

目標座標 ROT 修飾パラメーターが付加可能かどうかを、調べる目標座標をポイントデータで指定します。

基準座標 ROT 修飾パラメーターが付加可能かどうかを、調べる基準座標をポイントデータで指定します。

戻り値

ROT 修飾パラメーターを付加可能であれば“True”を、そうでない場合は“False”を返します。

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

解説

ROT 修飾パラメーターが付加可能かを、実際に(ロボットを)動かす前に確認します。

ROT 修飾パラメーターとは、Move などの直線補間動作命令に付加可能で、ツール姿勢変化の加減速を優先することを指定するパラメーターです。

ROKOK 関数では、基準座標から目標座標に Move など、直線補間動作命令で移動する場合に、ROT を付加するとエラーになるかどうかを、動作前に判定できます。

基準座標を省略した場合は、現在位置(Here)を基準として判定した結果を返します。

姿勢変化角度が“0”、または微小変化の場合、ROT 修飾パラメーターを付加すると発生するエラーを判定します。ただし、4242 エラーなど、動作中にロボットの関節速度や関節加速度が、マニピュレーターの限度を超えるかどうかの判定はできません。その場合は、SpeedS, SpeedR, AccelS, AccelR の各値を下げるなどの調整をしてください。

参照

Move

ROKOK 関数使用例

```
If ROTOK(P1) = True Then
  Move P1 ROT
Else
  Move P1
EndIf
```

RSet\$ 関数

文字列の先頭にスペースを追加して、指定の長さにした文字列を返します。

書式

RSet\$ (文字列, 文字列の長さ)

パラメーター

文字列 文字列式を指定します。

文字列の長さ 返される文字列の長さを示す、整数または式を指定します。

戻り値

先頭にスペースを追加した指定文字列を返します。

参照

LSet\$, Space\$

RSet\$ 関数 使用例

```
temp$ = "123"  
temp$ = RSet$(temp$, 10) ' temp$ = "      123"
```

RShift 関数

数値データの右論理シフト

書式

RShift(数値データ, シフトビット数)

パラメーター

数値データ 論理シフトする数値を、式または直接数値で指定します。
シフトビット数 右論理シフトするビット数値 (0~31 の整数値)を指定します。

戻り値

指定した数値データを、右論理シフトした値を返します。

解説

RShift は、指定した数値データを右 (下位方向)に指定ビット数シフトします。シフトした上位のビットは、常に 0 がセットされます。

最も簡単な説明としては、Rshift は数値データを 2 のシフトビット乗で割った数を返します。

注意

数値データ型

数値データは、有効な数値データ型ならば、どんなデータ型もとります。

RShift は、次のデータ型に対応しています。

: Byte 型, Double 型, Int32 型, Integer 型, Long 型, Real 型, Short 型, UByte 型, UInt32 型, UShort 型

参照

And, LShift, LShift64, Not, Or, RShift64, Xor

RShift 使用例

下記は、“0”から始まる Integer 型数値データに対する、すべての Rshift 値を示すプログラムの例です。

```
Function rshiftst
  Integer num, snum, i
  num = 32767
  For i = 1 to 16
    Print "i =", i
    snum = RShift(num, i)
    Print "RShift(32767, ", i, ") = ", snum
  Next i
Fend
```

コマンドウィンドウで Rshift 命令を操作した例です。

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

RShift64 関数

数値データの右論理シフト

書式

RShift64(数値データ, シフトビット数)

パラメーター

数値データ	論理シフトする数値を、式または直接数値で指定します。
シフトビット数	右論理シフトするビット数値 (0~63 の整数値)を指定します。

戻り値

指定した数値データを、右論理シフトした値を返します。

解説

RShift64 は、指定した数値データを右 (下位方向)に指定ビット数シフトします。シフトした上位のビットは、常に 0 がセットされます。

最も簡単な説明としては、Rshift64 は数値データを 2 のシフトビット乗で割った数を返します。

注意

数値データ型

数値型は数種類あります。RShift64 は、Int64 型, UInt64 型の数値に使うことができます。

参照

And, LShift, LShift64, Not, Or, RShift, Xor

RShift 64 使用例

下記は、“0”から始まる UInt64 型数値データに対する、すべての Rshift64 値を示すプログラムの例です。

```
Function rshif64tst
    UInt64 num, snum, i
    num = 18446744073709551615
    For i = 1 to 63
        Print "i =", i
        snum = RShift64(num, i)
        Print "RShift64(18446744073709551615, ", i, ") = ", snum
    Next i
End
```

コマンドウィンドウで Rshift64 命令を操作した例です。

```
> Print RShift64(10,1)
5
> Print RShift64(8,3)
1
> Print RShift64(16,2)
4
```

RTrim\$関数

文字列の右側スペースを削除して返します。

書式

RTrim\$(文字列)

パラメーター

文字列 文字列式または直接文字列で指定します。

戻り値

右側のスペースを削除した文字列

参照

LTrim\$, Trim\$

RTrim\$ 関数使用例

```
str$ = " data "  
str$ = RTrim$(str$) ' str$ = " data"
```

RunDialog

SPEL+ のプログラムから EPSON RC+画面を起動します。

書式

- (1) RunDialog ダイアログ ID
- (2) RunDialog DLG_ROBOTMNG [, ロボット選択ビット]

パラメーター

ダイアログ ID 有効なダイアログ ID を含んだ整数値を指定します。これらの値は、あらかじめ下記の定数で定義されています。

DLG_ROBOTMNG	100	ロボットマネージャダイアログを起動します
DLG_IOMON	102	I/O モニターを起動します
DLG_VGUIDE	110	Vision Guide ダイアログを起動します

ロボット選択ビット ダイアログ ID に DLG_ROBOTMNG を指定したときのみ有効です。ロボットマネージャで選択可能なロボットをビット値で指定します。

設定例	設定値	bit15	bit14	...	bit2	bit1	bit0
ロボット 1	&H0001	Off	Off		Off	Off	On
ロボット 2	&H0002	Off	Off		Off	On	Off
ロボット 1 と 2	&H0003	Off	Off		Off	On	On
:							
ロボット 16	&H1000	On	Off		Off	Off	Off

解説

SPEL+ のタスクから EPSON RC+の画面を起動/表示するときに、RunDialog を使います。タスクは、画面が閉じられるまで一時停止します。

EPSON RC+の画面からロボットコマンドを実行するときは、他に、その画面表示中にロボットを制御するタスクがないことを確認してください。もし他にロボットを制御するタスクが起動している場合は、エラーになります。

参照

InputBox, MsgBox

RunDialog 使用例

```

If Motor = Off Then
  RunDialog DLG_ROBOTMNG
  If Motor = Off Then
    Print "Motors are off, aborting program"
    Quit All
  EndIf
EndIf

```

SafetyOn 関数

安全扉の状態を返します。

書式

SafetyOn

戻り値

安全扉開状態のときは“True”、それ以外のときは“False”を返します。

解説

本関数は、NoPause タスク、NoEmgAbort タスク (Xqt 時に、NoPause や NoEmgAbort を指定して開始された特別なタスク)とバックグラウンドタスクでのみ使用します。

参照

ErrorOn, EstopOn, PauseOn, Wait, Xqt

SafetyOn 関数使用例

下記例は、コントローラーが安全扉開になるのを監視し、安全扉開が発生した場合 I/O をオンオフするプログラムです。

注意

Forced フラグ

本プログラム例では、On/Off 命令に Forced フラグを指定しています。
エラー発生中、および、非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

```
Function main
    Xqt SafetyOnOffMonitor, NoPause
    :
    :
Fend

Function SafetyOnOffMonitor
    Do
        Wait SafetyOn = On
        Print "Saftey Open"
        Off 10, Forced
        On 12, Forced

        Wait SafetyOn = Off
        Print "Saftey Close"
        On 10, Forced
        Off 12, Forced
    Loop
Fend
```


SavePoints

メインメモリーにあるポイントデータを、カレントロボットのディスクファイルに保存します。

書式

SavePoints ファイル名

パラメーター

ファイル名 ポイントデータを保存する先のファイル名を、文字列式で指定します。
拡張子は、“**.pts**”固定です。
パスを指定することはできません。また、ChDisk などの影響を受けません。
詳細は、ChDisk を参照してください。

解説

SavePoints は、指定したファイルにポイントを保存します。拡張子は、“**.pts**”固定です。拡張子を省略した場合、“**.pts**”が追加されます。

また、SavePoints コマンドは、すでにポイントファイルが存在していなければ、カレントロボットのプロジェクトへ追加します。

ポイントデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、SavePoints を実行するとコンパクトフラッシュへの書き込みが発生します。あまりにも頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。ポイントデータの保存が必要な場合にのみ SavePoints を実行することを推奨します。

起こりやすいエラー

ディスクの容量を超えている場合

ディスク容量が残っていない場合、エラーが発生します。

指定したファイルが現在のロボットのファイルでない場合

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。

指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。カレントプロジェクトのファイル名のみ指定できます。

ファイル名のエラー

ファイル名内にスペースや、無効な文字が含まれていた場合、エラーが発生します。

参照

ImportPoints, LoadPoints

SavePoints 使用例

```
ClearPoints
For i = 1 To 10
    P(i) = XY( i, 100, 0, 0 )
Next i
SavePoints "TEST.PTS"
```

Seek

指定したファイルのファイルポインター位置を変更します。

書式

Seek #ファイル番号, ポインター

パラメーター

ファイル番号 30~63 の整数値または式で指定します。

ポインター 0 からファイルサイズまでのファイルポインターを整数値または式で指定します。

参照

BOpen, Read, ROpen, UOpen, Write, WOpen

Seek 使用例

```
Integer fileNum
String data$

fileNumber = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 20
Read #fileNum, data$, 2
Close #fileNum
```

Select...Send

式の値にしたがって、いくつかの文のうちのいずれかに制御を移します。

書式

```
Select 式
    Case 項目
        ステートメント
    [ Case 項目
        ステートメント ]
    [Default
        ステートメント]
Send
```

パラメーター

式	数値式あるいは文字列式を指定します。
項目	式と一致する型の数値または文字列式を指定します。
ステートメント	1 つまたは複数の有効な SPEL+ ステートメント、あるいはマルチステートメントを指定します。

解説

Case 文の項目の中に、Select 文の式の結果と一致するものがあれば、Case ステートメントの後のステートメント群が実行されます。実行後、プログラム制御は、Send ステートメントに続くステートメントに移ります。

Case 文の項目の中に Select 文の式の結果と一致するものがなければ、Default ステートメントが実行され、プログラム制御は Send ステートメントに続くステートメントに移ります。

Case 文の項目の中に Select 文の式の結果と一致するものがなく、Default が省略されると、何も実行されず、プログラム制御は Send ステートメントの次のステートメントに移ります。

Select 文の式には、定数、変数、そして And, Or, Xor などの演算子を指定できます。

Case 文の項目にも 定数、変数、そして And, Or, Xor などの演算子を指定できます。この場合 Case 文の項目の演算結果が Select 文の式と比較されます。また、動作が複雑になることから Case 文の項目に 変数を指定しないでください。

参照

If...Then...Else

Select...Send 使用例

下記に簡単な Select...Send の例を示します。

```
Function Main
  Integer I
  For i = 0 To 10
    Select I
      Case 0
        Off 1;On 2;Jump P1
      Case 3
        On 1;Off 2
        Jump P2;Move P3;On 3
      Case 7
        On 4
      Default
        On 7
    Send
  Next
Fend
```

SelectDB

オープンしているデータベース内のテーブルからデータを検索する関数です。

書式

SelectDB(#データベース番号, テーブル名, Select 条件, Sort 方法)

パラメーター

データベース番号	OpenDB で指定したデータベース番号 (501~508 の整数値)を指定します。
テーブル名	データ検索を行うテーブル名を指定します。 ファイル番号で指定されたデータベースの種別が Excel ブックの場合、Excel シートまたは名前をつけたテーブルを指定します。 Excel シートを指定する場合、シート名の後ろに\$をつけ、[]で囲みます。 Excel シート内の名前前で指定した領域を指定する場合、名前を[]で囲みます。
Select 条件	検索条件を指定します。 AND, OR を使用して複合条件を指定できます。 検索条件を指定しない場合、テーブルの全データが検索されます。
Sort 方法	検索したデータを抽出する順番を指定します。 Sort キーと Sort 順番(昇順[ASC]/降順[DESC])を指定します。 Sort 順番を省略した場合、Sort キーの昇順を指定したことになります。 Sort 方法を省略した場合、オープンされているデータベースにより、順番は決定されます。

戻り値

検索された行の総数を返します。

解説

オープンされたデータベースの指定されたテーブルから Select 条件に一致するデータを Sort 条件により Sort します。

Input#, Print#によりデータの読み込み、書き込みを行う前に必ず実行する必要があります。

オープンされたデータベースが Excel ブックの場合、シートおよび名前前で定義された領域の 1 行目に検索に使用する列名を記述してください。

また、Excel 2007 ブックの場合、シート名を指定してください。名前前で定義された領域へのアクセスは行えません。

注意

- RC+ がインストールされている PC の接続が必要です

参照

OpenDB, CloseDB, UpdateDB, DeleteDB, Input #, Print #

SelectDB 関数使用例

SQL データベースを使用した例

下記に、SQL サーバー 2000 のサンプルデータベース Northwind のテーブル Employees から TitleOfCourtesy が Ms.のデータを EmployeeID の降順に読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
```

```
count = SelectDB(#501, "Employees", "TitleOfCourtesy = 'Ms.'",  
"EmployeeID DESC")  
For i = 0 To count - 1  
    Input #501, eid, Lastname$, Firstname$, Title$  
    Print eid, ",", Lastname$, ",", Firstname$, ",", Title$  
Next  
CloseDB #501
```

Access データベースを使用した例

下記に、Microsoft Access 2007 のサンプルデータベース生徒名簿のテーブル生徒から役職が組長のデータを ID の昇順に読み込む簡単な例を示します。

```
Integer count, i, eid  
String Lastname$, Firstname$, dummy$  
  
OpenDB #502, Access, "c:\MyDataBase\生徒名簿.accdb"  
count = SelectDB(#502, "生徒", "役職 = '組長'", "ID")  
For i = 0 To count - 1  
    Input #502, eid, dummy$, dummy$, Lastname$, dummy$, Firstname$  
    Print eid, ",", Lastname$, ",", Firstname$  
Next  
CloseDB #502
```

Excel ブックを使用した例

下記に、Microsoft Excel ブック生徒名簿の生徒シートから Age が 25 未満のデータを ID の昇順に読み込む簡単な例を示します。

```
Integer count, i, eid  
String Lastname$, Firstname$  
  
OpenDB #503, Excel, "c:\MyDataBase\生徒名簿.xls"  
count = SelectDB(#503, "[生徒$]", "Age < 25", "ID ASC")  
For i = 0 To count - 1  
    Input #503, eid, Lastname$, Firstname$  
    Print eid, ",", Lastname$, ",", Firstname$  
Next  
CloseDB #503
```

Sense

Jump, Jump3, Jump3CP で Sense を指定したときに、目標座標上空で停止させるための条件の設定と表示をします。

書式

Sense [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子 (=, <, >=, >, <=, <=) [整数式]

次の関数や変数をイベントに使うことができます。

関数 : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr
GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out,
AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

変数 : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ
変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

演算子 : And, Or, Xor

<例> Sense Sw(5) = On

Sense Sw(5) = On And Sw(6) = Off

解説

Sense は、Jump コマンド実行中に第 3 関節の下降開始直前のタイミングで入力条件をチェックします。また、Jump3, Jump3CP コマンド実行中に接近動作開始直前のタイミングで入力条件をチェックします。Sense 条件式には、上記の関数を 1 つ以上含んでいなくてはなりません。

Sense 条件式に変数が含まれた場合、その値は、Sense 条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。Sense ステートメントは複数使うこともできます。次の Sense ステートメントに移るまで最後に実行された入力条件が有効です。

Jump と Sense 修飾子

現在の Sense 条件が成立するかチェックします。成立すれば、Jump コマンドは、ロボットが目標座標の上空で停止した状態で完了したことになります。つまり、Sense 条件が“True”のときは、ロボットは目標座標の上空で第 3 関節が下降開始直前の状態で止まります。Sense 条件が“False”のときは、ロボットは目標座標で Jump 命令の動作を完了します。

Jump3, Jump3CP と Sense 修飾子

現在の Sense 条件が成立するかチェックします。成立すれば、Jump3, Jump3CP コマンドは、ロボットが接近開始位置で停止した状態で完了したことになります。

パラメーターを省略すると、現在の Sense 設定を表示します。

注意

電源オン時の Sense 設定

電源オン時の Sense 条件の初期設定は、Sense Sw(0)= On です。入力ビット番号 0 がオンのとき、ロボットが下降動作を行わないように設定されています。

Sense 条件成立をチェックする JS 関数と State 関数

Sense 修飾子を用いた動作コマンドを実行した後で、Sense 条件が成立したかどうかを、JS 関数や State 関数を用いてチェックすることができます。

イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が 0.01 秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1 システム内で使用できる変数待ちの個数は最大 64 個です (Wait などのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数を Byref で参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。

参照

In, JS, Jump, Jump3, Jump3CP, MemIn, MemSw, Stat, Sw, SF_GetStatus

Sense 使用例

Sense 命令の簡単な使用例です。

```
Function test
.
.
TrySense:
    Sense Sw(1) = Off      '入力ビット 1 が Off のとき目標座標上空で停止するように設定
    Jump P1 C2 Sense
    If JS = True Then
        GoSub ERRPRC      'アームが目標座標上空で停止している場合は
        GoTo TrySense     'ERRPRC を実行し TrySense に移動
    EndIf
    On 1; Wait 0.2; Off 1
.
.
Fend
```

<その他の書式例>

```
> Sense Sw(1)=1 And MemSw(1)=1

> Sense Sw(0) Or (Sw(1) And MemSw(1))
```


SetCom

RS-232C ポートのパラメーターを設定または表示します。

書式

SetCom #通信ポート番号 [, 通信速度] [, データビット長] [, ストップビット長] [, パリティ] [, 送受信行末] [, H/W フロー制御] [, S/W フロー制御] [, タイムアウト時間]

パラメーター

通信ポート番号	RS-232C のポート番号を整数値で指定します。 SPEL ⁺ 制御部 1~8 PC 部 1001~1008
通信速度	ボーレートを指定します。有効値は下記のとおりです。省略可能です。 110 2400 19200 300 4800 38400 600 9600 57600 1200 14400 115200 (デフォルト値: 9600) PC 部のポートを使用する場合、19200 以上の通信速度ではデータを取りこぼすケースがあります。
データビット長	1 文字ずつのデータビット長を 7 または 8 の数値で指定します。省略可能です。
ストップビット長	1 文字ずつのストップビット長を 1 または 2 の数値で指定します。省略可能です。
パリティ	パリティを指定します。奇数の場合は O、偶数の場合は E、なしの場合は N を指定します。省略可能です。
送受信行末	CR, LF, CRLF のいずれかの送受信行末を指定します。省略可能です。
H/W フロー制御	ハードウェア制御が有効な場合は RTS、無効な場合は NONE を指定します。省略可能です。
S/W フロー制御	ソフトウェア制御が有効な場合は XON、無効な場合は NONE を指定します。省略可能です。
タイムアウト時間	タイムアウト時間(正の実数値、単位: 秒)を式または数値で指定します。 0 を指定するとタイムアウトは無限になります。省略可能です。

解説

すべてのパラメーターを省略した場合、通信ポートの設定を表示します。

複数のポートを 19200 以上の通信速度に設定し同時に通信を行うと、エラー 2929 または 2922 が発生する場合があります。この場合、より遅い転送速度を選択するか、同時に通信を行わないようにしてください。

PC 部のポートを使用する場合、19200 以上の通信速度ではデータを取りこぼすケースがあります。

取りこぼしが発生した場合は、より遅い転送速度を選択するか、SPEL⁺制御部のポートを使用してください。

パラメーターは、コントローラー内のコンパクトフラッシュに格納されています。このため、SetCom を実行するとコンパクトフラッシュへの書き込みが発生します。あまりにも頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。パラメーターの変更が必要な場合にのみ SetCom を実行することを推奨します。

参照

OpenCom, CloseCom, SetNet

SetCom 使用例

SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0

SetCom #2, 4800

SetLatch

R-I/O 入力によるロボット位置のラッチ機能を設定します。

書式

SetLatch { ポート番号, 入力論理, 連続ラッチ回数 }

パラメーター

ポート番号

トリガー入力信号を接続する R-I/O 入力ポートのポート番号を指定します。

指定できるポート番号は以下のとおりです。対象とするロボットが接続されているユニットのポート番号を指定します。

		ポイント	ポート番号
コントロールユニット	入力	2 点	24,25
	出力	-	-
ドライブユニット 1	入力	2 点	56,57
	出力	-	-
ドライブユニット 2	入力	2 点	280,281
	出力	-	-

ポート番号として以下の定数が定義されています。

定数	ポート番号
SETLATCH_PORT_CU_0	24
SETLATCH_PORT_CU_1	25
SETLATCH_PORT_DU1_0	56
SETLATCH_PORT_DU1_1	57
SETLATCH_PORT_DU2_0	280
SETLATCH_PORT_DU2_1	281

入力論理

R-I/O に接続するトリガー入力信号の論理を指定します。論理は以下の定数で指定することができます。

定数	値	意味
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	負論理
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	正論理

負論理では入力信号の High から Low への切り替わりエッジで、ロボット位置をラッチします。正論理では入力信号の Low から High への切り替わりエッジで、ロボット位置をラッチします。

連続ラッチ回数

R-I/O 入力信号によるロボット位置のラッチの連続回数を指定します。

1,2,3,4 が指定可能です。LatchEnable On 以降、指定した連続ラッチ回数分のポイントデータのラッチが可能となります。最大 4 回までのラッチが可能です。

パラメーターは省略可能です。省略した場合、連続ラッチ回数は 1 回となります。

解説

R-I/O 入力信号によるロボット位置のラッチ条件を設定します。1 台のロボットが複数のポートのトリガー信号を同時に待つことはできません。SetLatch の実行には、約 40 msec の処理時間がかかります。

注意

選択されているロボットと関係ない別ユニットのポート番号を指定するとパラメーター範囲外エラーが発生します。

参照

LatchEnable, LatchState 関数, LatchPos 関数

SetLatch 使用例

```
Function main
    SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE, 4
    '正論理 連続ラッチ 4 回設定

    LatchEnable On          'ラッチ機能有効
    Go P1
    Wait LatchState = True  'トリガー待ち
    Print LatchPos(WithoutToolArm, 1) 'ラッチ位置 1 表示
    Print LatchPos(WithoutToolArm, 2) 'ラッチ位置 2 表示
    Print LatchPos(WithoutToolArm, 3) 'ラッチ位置 3 表示
    Print LatchPos(WithoutToolArm, 4) 'ラッチ位置 4 表示
    LatchEnable Off        'ラッチ機能無効
Fend
```

パラメーターを省略する場合の例:

```
Function main
    SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE '正論理
    LatchEnable On 'ラッチ機能有効
    Go P1
    Wait LatchState = True 'トリガー待ち
    Print LatchPos 'ラッチ位置表示
    LatchEnable Off 'ラッチ機能無効
Fend
```

SetIn

仮想 I/O の入力ポート (8 ビット)を設定します。

書式

SetIn ポート番号, 設定値

パラメーター

ポート番号 I/O 入力バイトを指定します。

設定値 範囲が 0~255 の整数でポート番号を指定します。

解説

仮想 I/O が有効な場合、8 個の入力ビットを同時に設定します。
仮想 I/O が無効な場合、本コマンドはエラーになります。

参照

SetSW, SetInW

SetIn 使用例

```
> setin 0, 1  'ポート 0 の最初のビットをオンに設定
```

SetInW

仮想 I/O の入力ポート (16ビット)を設定します。

書式

SetInW ポート番号, 設定値

パラメーター

ポート番号 I/O 入力ワードを指定します。

設定値 範囲が 0~65535 の整数でワードを指定します。

注意

リアルタイム I/O の入力ビットを含むワードポートに関する規則

リアルタイム I/O の入力ビットに関しては、反映することはできません。

リアルタイム I/O の入力ビットを含むワードポート=1, 3, 17, 19 は、設定値の範囲は 0 から 255 の整数で指定してください。

255 より大きい値は、エラーになります。

解説

仮想 I/O が有効な場合、16 個の入力ビットを同時に設定します。

仮想 I/O が無効な場合、本コマンドはエラーになります。

参照

SetSw, SetIn

SetInW 使用例

> **setinw** 0, 1 'ワード 0 の最初のビットをオンに設定

SetNet

TCP/IP ポートのパラメーターの設定をします。

書式

(1) SetNet #通信ポート番号, ホストアドレス [, TCP/IP ポート番号 [, ターミネーター [, フロー制御 [, タイムアウト時間 [, プロトコル [, CloseNet タイムアウト時間]]]]]]]

(2) SetNet

パラメーター

通信ポート番号	パラメーターを設定する TCP/IP のポート番号を指定します。範囲は 201~216 です。
ホストアドレス	ホストの IP アドレスを指定します。
TCP/IP ポート番号	TCP/IP ポート番号を指定します。
ターミネーター	CR, LF, CRLF のいずれかの行末文字を指定します。
フロー制御	ソフトウェア制御フローです。NONE を指定します。
タイムアウト時間	送受信の最大時間を秒で指定します。0 の場合、タイムアウトは無限となります。
プロトコル	通信のプロトコル (TCP/UDP/UDP_SEND/UDP_RECV) を指定します。 TCP: TCP 通信 UDP: UDP 通信 UDP_SEND: UDP 送信 UDP_RECV: UDP 受信
CloseNet タイムアウト時間	CloseNet でソケットを閉じるまでの時間を、秒単位で指定します。(整数: 0 - 5) 0 を指定すると、シャットダウン要求に対する応答を待たずに、ソケットを閉じます。

解説

パラメーターは、コントローラー内のコンパクトフラッシュに格納されています。このため、SetNet を実行するとコンパクトフラッシュへの書き込みが発生します。あまりにも頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。パラメーターの変更が必要な場合にのみ SetNet を実行することを推奨します。

参照

OpenNet, WaitNet, CloseNet, SetCom

SetNet 使用例

```
SetNet #201, "192.168.0.1", 2001, CRLF, NONE, 0, TCP, 5
```

SetSw

仮想 I/O の入力を設定します。

書式

SetSw ビット番号, 設定値

パラメーター

ビット番号 I/O 入力ビットを指定します。

設定値 0 (オフ)か 1 (オン)の整数を指定します。

解説

仮想 I/O が有効な場合、入力ビットを設定します。

仮想 I/O が無効な場合、本コマンドはエラーになります。

参照

SetIn, SetInW

SetSw 使用例

```
> setsw 2, on '2 つ目の入力ビットをオンに設定
```


SF_GetParam 関数

安全機能パラメーターの情報を返します。

書式

SF_GetParam (インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値、または定数で指定します。

戻り値

指定した情報の整数値を返します。

定数末尾が、“_EN”のインデックスを指定した場合、有効な場合に 1、無効な場合に 0 を返します。

解説

指定した安全機能パラメーター値を返します。

インデックス	定数	解説
1	DRYRUNOFF	ドライラン無効状態
2	SLS_1_HAND_EN	SLS_1 のハンド速度監視状態
3	SLS_1_SPEED	SLS_1 の監視速度設定値
4	SLS_1_ELBOW_EN	SLS_1 の肘 (水平多関節型: J2, 垂直 6 軸型: J3) 速度監視状態 *1
5	SLS_1_JOINT_EN	SLS_1 の関節速度監視状態
6	SLS_1_JOINTSPEED	SLS_1 の監視関節速度設定値
7	SLS_1_WRIST_EN	SLS_1 の手首(垂直 6 軸型: J5)速度監視状態 *1
8	SLS_1_SHOULDER_EN	SLS_1 の肩(垂直 6 軸型: J2)速度監視状態 *1
9	SLS_2_HAND_EN	SLS_2 のハンド速度監視状態
10	SLS_2_SPEED	SLS_2 の監視速度設定値
11	SLS_2_ELBOW_EN	SLS_2 の肘 (水平多関節型: J2, 垂直 6 軸型: J3) 速度監視状態 *1
12	SLS_2_JOINT_EN	SLS_2 の関節速度監視状態
13	SLS_2_JOINTSPEED	SLS_2 の監視関節速度設定値
14	SLS_2_WRIST_EN	SLS_2 の手首(垂直 6 軸型: J5)速度監視状態 *1
15	SLS_2_SHOULDER_EN	SLS_2 の肩(垂直 6 軸型: J2)速度監視状態 *1
16	SLS_3_HAND_EN	SLS_3 のハンド速度監視状態
17	SLS_3_SPEED	SLS_3 の監視速度設定値
18	SLS_3_ELBOW_EN	SLS_3 の肘 (水平多関節型: J2, 垂直 6 軸型: J3) 速度監視状態 *1
19	SLS_3_JOINT_EN	SLS_3 の関節速度監視状態
20	SLS_3_JOINTSPEED	SLS_3 の監視関節速度設定値
21	SLS_3_WRIST_EN	SLS_3 の手首(垂直 6 軸型: J5)速度監視状態 *1
22	SLS_3_SHOULDER_EN	SLS_3 の肩(垂直 6 軸型: J2)速度監視状態 *1
23	SLS_T2_HAND_EN	SLS_T2 のハンド速度監視状態
24	SLS_T2_SPEED	SLS_T2 の監視速度設定値
25	SLS_T2_ELBOW_EN	SLS_T2 の肘 (水平多関節型: J2, 垂直 6 軸型: J3) 速度監視状態 *1
26	SLS_T2_JOINT_EN	SLS_T2 の関節速度監視状態

インデックス	定数	解説
27	SLS_T2_JOINTSPEED	SLS_T2 の監視関節速度設定値
28	SLS_T2_WRIST_EN	SLS_T2 の手首(垂直 6 軸型: J5)速度監視状態 *1
29	SLS_T2_SHOULDER_EN	SLS_T2 の肩(垂直 6 軸型: J2)速度監視状態 *1
30	SLS_T_SPEED	SLS_T の監視速度設定値
31	SLS_T_JOINT_EN	SLS_T の関節速度監視状態
32	SLS_T_JOINTSPEED	SLS_T の監視関節速度設定値
33	SLS_HAND_OFS_X	SLS の X 軸方向 TCP オフセット位置
34	SLS_HAND_OFS_Y	SLS の Y 軸方向 TCP オフセット位置
35	SLS_HAND_OFS_Z	SLS の Z 軸方向 TCP オフセット位置
36	SLS_1_DELAY	SLS_1 の遅延時間設定値
37	SLS_2_DELAY	SLS_2 の遅延時間設定値
38	SLS_3_DELAY	SLS_3 の遅延時間設定値
39	SLS_JOINT_POS_EN	関節角度監視の監視状態
40	SLS_JOINT_POS_ANGLE	監視関節角度設定値
41	SLP_A_XU_EN	SLP_A の XU (壁: X2, 制限領域: X1)位置監視状態 *2
42	SLP_A_XU_POS	SLP_A の XU (壁: X2, 制限領域: X1)監視位置 設定値 *2
43	SLP_A_XL_EN	SLP_A の XL (壁: X1, 制限領域: X2)位置監視状態 *2
44	SLP_A_XL_POS	SLP_A の XL (壁: X1, 制限領域: X2)監視位置 設定値 *2
45	SLP_A_YU_EN	SLP_A の YU (壁: Y2, 制限領域: Y1)位置監視状態 *2
46	SLP_A_YU_POS	SLP_A の YU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
47	SLP_A_YL_EN	SLP_A の YL (壁: Y1, 制限領域: Y2)位置監視状態 *2
48	SLP_A_YL_POS	SLP_A の YL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
49	SLP_A_ZU_EN	SLP_A の Z2 位置監視状態 *2
50	SLP_A_ZU_POS	SLP_A の Z2 監視位置 設定値 *2
51	SLP_A_ZL_EN	SLP_A の Z1 位置監視状態 *2
52	SLP_A_ZL_POS	SLP_A の Z1 監視位置 設定値 *2
53	SLP_B_XU_EN	SLP_B の XU (壁: X2, 制限領域: X1)位置監視状態 *2
54	SLP_B_XU_POS	SLP_B の XU (壁: X2, 制限領域: X1)監視位置 設定値 *2
55	SLP_B_XL_EN	SLP_B の XL (壁: X1, 制限領域: X2)位置監視状態 *2
56	SLP_B_XL_POS	SLP_B の XL (壁: X1, 制限領域: X2)監視位置 設定値 *2
57	SLP_B_YU_EN	SLP_B の YU (壁: Y2, 制限領域: Y1)位置監視状態 *2
58	SLP_B_YU_POS	SLP_B の YU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
59	SLP_B_YL_EN	SLP_B の YL (壁: Y1, 制限領域: Y2)位置監視状態 *2
60	SLP_B_YL_POS	SLP_B の YL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
61	SLP_B_ZU_EN	SLP_B の Z2 位置監視状態 *2
62	SLP_B_ZU_POS	SLP_B の Z2 監視位置 設定値 *2
63	SLP_B_ZL_EN	SLP_B の Z1 位置監視状態 *2
64	SLP_B_ZL_POS	SLP_B の Z1 監視位置 設定値 *2
65	SLP_C_XU_EN	SLP_C の XU (壁: X2, 制限領域: X1)位置監視状態 *2
66	SLP_C_XU_POS	SLP_C の XU (壁: X2, 制限領域: X1)監視位置 設定値 *2
67	SLP_C_XL_EN	SLP_C の XL (壁: X1, 制限領域: X2)位置監視状態 *2
68	SLP_C_XL_POS	SLP_C の XL (壁: X1, 制限領域: X2)監視位置 設定値 *2
69	SLP_C_YU_EN	SLP_C の YU (壁: Y2, 制限領域: Y1)位置監視状態 *2
70	SLP_C_YU_POS	SLP_C の YU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
71	SLP_C_YL_EN	SLP_C の YL (壁: Y1, 制限領域: Y2)位置監視状態 *2
72	SLP_C_YL_POS	SLP_C の YL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
73	SLP_C_ZU_EN	SLP_C の Z2 位置監視状態 *2

インデックス	定数	解説
74	SLP_C_ZU_POS	SLP_C の Z2 監視位置 設定値 *2
75	SLP_C_ZL_EN	SLP_C の Z1 位置監視状態 *2
76	SLP_C_ZL_POS	SLP_C の Z1 監視位置 設定値 *2
77	SLP_J2_MON_RAD	SLP の J2 軸監視範囲半径設定値
78	SLP_J3_MON_RAD	SLP の J3 軸監視範囲半径設定値
79	SLP_J5_MON_RAD	SLP の J5 軸監視範囲半径設定値
80	SLP_J6_MON_RAD	SLP の J6 軸監視範囲半径設定値
81	SLP_J1_RANGE_MAX	ソフト軸制限の J1 軸制限範囲最大設定値
82	SLP_J1_RANGE_MIN	ソフト軸制限の J1 軸制限範囲最小設定値
83	SLP_J2_RANGE_MAX	ソフト軸制限の J2 軸制限範囲最大設定値
84	SLP_J2_RANGE_MIN	ソフト軸制限の J2 軸制限範囲最小設定値
85	SLP_J3_RANGE_MAX	ソフト軸制限の J3 軸制限範囲最大設定値
86	SLP_J3_RANGE_MIN	ソフト軸制限の J3 軸制限範囲最小設定値
87	SLP_J4_RANGE_MAX	ソフト軸制限の J4 軸制限範囲最大設定値
88	SLP_J4_RANGE_MIN	ソフト軸制限の J4 軸制限範囲最小設定値
89	SLP_J5_RANGE_MAX	ソフト軸制限の J5 軸制限範囲最大設定値
90	SLP_J5_RANGE_MIN	ソフト軸制限の J5 軸制限範囲最小設定値
91	SLP_J6_RANGE_MAX	ソフト軸制限の J6 軸制限範囲最大設定値
92	SLP_J6_RANGE_MIN	ソフト軸制限の J6 軸制限範囲最小設定値
93	SIN_1_SLS_1_EN	SAFETY_IN1 に対する SLS_1 機能割り当て状態
94	SIN_1_SLS_2_EN	SAFETY_IN1 に対する SLS_2 機能割り当て状態
95	SIN_1_SLS_3_EN	SAFETY_IN1 に対する SLS_3 機能割り当て状態
96	SIN_1_SLP_A_EN	SAFETY_IN1 に対する SLP_A 機能割り当て状態
97	SIN_1_SLP_B_EN	SAFETY_IN1 に対する SLP_B 機能割り当て状態
98	SIN_1_SLP_C_EN	SAFETY_IN1 に対する SLP_C 機能割り当て状態
99	SIN_1_SG_EN	SAFETY_IN1 に対する保護停止機能割り当て状態
100	SIN_1_ESTOP_EN	SAFETY_IN1 に対する非常停止機能割り当て状態
101	SIN_2_SLS_1_EN	SAFETY_IN2 に対する SLS_1 機能割り当て状態
102	SIN_2_SLS_2_EN	SAFETY_IN2 に対する SLS_2 機能割り当て状態
103	SIN_2_SLS_3_EN	SAFETY_IN2 に対する SLS_3 機能割り当て状態
104	SIN_2_SLP_A_EN	SAFETY_IN2 に対する SLP_A 機能割り当て状態
105	SIN_2_SLP_B_EN	SAFETY_IN2 に対する SLP_B 機能割り当て状態
106	SIN_2_SLP_C_EN	SAFETY_IN2 に対する SLP_C 機能割り当て状態
107	SIN_2_SG_EN	SAFETY_IN2 に対する保護停止機能割り当て状態
108	SIN_2_ESTOP_EN	SAFETY_IN2 に対する非常停止機能割り当て状態
109	SIN_3_SLS_1_EN	SAFETY_IN3 に対する SLS_1 機能割り当て状態
110	SIN_3_SLS_2_EN	SAFETY_IN3 に対する SLS_2 機能割り当て状態
111	SIN_3_SLS_3_EN	SAFETY_IN3 に対する SLS_3 機能割り当て状態
112	SIN_3_SLP_A_EN	SAFETY_IN3 に対する SLP_A 機能割り当て状態
113	SIN_3_SLP_B_EN	SAFETY_IN3 に対する SLP_B 機能割り当て状態
114	SIN_3_SLP_C_EN	SAFETY_IN3 に対する SLP_C 機能割り当て状態
115	SIN_3_SG_EN	SAFETY_IN3 に対する保護停止機能割り当て状態
116	SIN_3_ESTOP_EN	SAFETY_IN3 に対する非常停止機能割り当て状態
117	SIN_4_SLS_1_EN	SAFETY_IN4 に対する SLS_1 機能割り当て状態
118	SIN_4_SLS_2_EN	SAFETY_IN4 に対する SLS_2 機能割り当て状態
119	SIN_4_SLS_3_EN	SAFETY_IN4 に対する SLS_3 機能割り当て状態
120	SIN_4_SLP_A_EN	SAFETY_IN4 に対する SLP_A 機能割り当て状態

インデックス	定数	解説
121	SIN_4_SLP_B_EN	SAFETY_IN4 に対する SLP_B 機能割り当て状態
122	SIN_4_SLP_C_EN	SAFETY_IN4 に対する SLP_C 機能割り当て状態
123	SIN_4_SG_EN	SAFETY_IN4 に対する保護停止機能割り当て状態
124	SIN_4_ESTOP_EN	SAFETY_IN4 に対する非常停止機能割り当て状態
125	SIN_5_SLS_1_EN	SAFETY_IN5 に対する SLS_1 機能割り当て状態
126	SIN_5_SLS_2_EN	SAFETY_IN5 に対する SLS_2 機能割り当て状態
127	SIN_5_SLS_3_EN	SAFETY_IN5 に対する SLS_3 機能割り当て状態
128	SIN_5_SLP_A_EN	SAFETY_IN5 に対する SLP_A 機能割り当て状態
129	SIN_5_SLP_B_EN	SAFETY_IN5 に対する SLP_B 機能割り当て状態
130	SIN_5_SLP_C_EN	SAFETY_IN5 に対する SLP_C 機能割り当て状態
131	SIN_5_SG_EN	SAFETY_IN5 に対する保護停止機能割り当て状態
132	SIN_5_ESTOP_EN	SAFETY_IN5 に対する非常停止機能割り当て状態
133	SOUT_1_STO	SAFETY_OUT1 に対する STO 機能割り当て状態
134	SOUT_1_SLS_1	SAFETY_OUT1 に対する SLS_1 機能割り当て状態
135	SOUT_1_SLS_2	SAFETY_OUT1 に対する SLS_2 機能割り当て状態
136	SOUT_1_SLS_3	SAFETY_OUT1 に対する SLS_3 機能割り当て状態
137	SOUT_1_SLS_T2	SAFETY_OUT1 に対する SLS_T2 機能割り当て状態
138	SOUT_1_SLS_T	SAFETY_OUT1 に対する SLS_T 機能割り当て状態
139	SOUT_1_SLP_A	SAFETY_OUT1 に対する SLP_A 機能割り当て状態
140	SOUT_1_SLP_B	SAFETY_OUT1 に対する SLP_B 機能割り当て状態
141	SOUT_1_SLP_C	SAFETY_OUT1 に対する SLP_C 機能割り当て状態
142	SOUT_1_EP_RC	SAFETY_OUT1 に対する非常停止 (コントローラー)機能割り当て状態
143	SOUT_1_EP_TP	SAFETY_OUT1 に対する非常停止 (ティーチペンダント)機能割り当て状態
144	SOUT_1_EN_SW	SAFETY_OUT1 に対するイネーブルスイッチ機能割り当て状態
145	SOUT_2_STO	SAFETY_OUT2 に対する STO 機能割り当て状態
146	SOUT_2_SLS_1	SAFETY_OUT2 に対する SLS_1 機能割り当て状態
147	SOUT_2_SLS_2	SAFETY_OUT2 に対する SLS_2 機能割り当て状態
148	SOUT_2_SLS_3	SAFETY_OUT2 に対する SLS_3 機能割り当て状態
149	SOUT_2_SLS_T2	SAFETY_OUT2 に対する SLS_T2 機能割り当て状態
150	SOUT_2_SLS_T	SAFETY_OUT2 に対する SLS_T 機能割り当て状態
151	SOUT_2_SLP_A	SAFETY_OUT2 に対する SLP_A 機能割り当て状態
152	SOUT_2_SLP_B	SAFETY_OUT2 に対する SLP_B 機能割り当て状態
153	SOUT_2_SLP_C	SAFETY_OUT2 に対する SLP_C 機能割り当て状態
154	SOUT_2_EP_RC	SAFETY_OUT2 に対する非常停止 (コントローラー)機能割り当て状態
155	SOUT_2_EP_TP	SAFETY_OUT2 に対する非常停止 (ティーチペンダント)機能割り当て状態
156	SOUT_2_EN_SW	SAFETY_OUT2 に対するイネーブルスイッチ機能割り当て状態
157	SOUT_3_STO	SAFETY_OUT3 に対する STO 機能割り当て状態
158	SOUT_3_SLS_1	SAFETY_OUT3 に対する SLS_1 機能割り当て状態
159	SOUT_3_SLS_2	SAFETY_OUT3 に対する SLS_2 機能割り当て状態
160	SOUT_3_SLS_3	SAFETY_OUT3 に対する SLS_3 機能割り当て状態
161	SOUT_3_SLS_T2	SAFETY_OUT3 に対する SLS_T2 機能割り当て状態
162	SOUT_3_SLS_T	SAFETY_OUT3 に対する SLS_T 機能割り当て状態
163	SOUT_3_SLP_A	SAFETY_OUT3 に対する SLP_A 機能割り当て状態

インデックス	定数	解説
164	SOUT_3_SLP_B	SAFETY_OUT3 に対する SLP_B 機能割り当て状態
165	SOUT_3_SLP_C	SAFETY_OUT3 に対する SLP_C 機能割り当て状態
166	SOUT_3_EP_RC	SAFETY_OUT3 に対する非常停止 (コントローラー)機能割り当て状態
167	SOUT_3_EP_TP	SAFETY_OUT3 に対する非常停止 (ティーチペンダント)機能割り当て状態
168	SOUT_3_EN_SW	SAFETY_OUT3 に対するイネーブルスイッチ機能割り当て状態
169	POS_ROT_U	設定平面回転 U_ROT 設定値
170	POS_ROT_V	設定平面回転 V_ROT 設定値
171	POS_ROT_W	設定平面回転 W_ROT 設定値
172	POS_OFS_X	設置位置オフセット X_OFS 設定値
173	POS_OFS_Y	設置位置オフセット Y_OFS 設定値
174	POS_OFS_Z	設置位置オフセット Z_OFS 設定値

*1 安全機能マネージャーの安全速度監視の監視部位 J2, J3, J5 と、本マニュアルで扱う速度超過部位 ハンド, 手首, 肘, 肩との対応は以下のとおりです。

水平多関節型

J2: 肘

J3: 該当なし

J5: 該当なし

Hand: ハンド

垂直 6 軸型

J2: 肩

J3: 肘

J5: 手首

Hand: ハンド

*2 安全機能マネージャーの安全位置監視の監視位置 X1, X2, Y1, Y2, Z1, Z2 と、本マニュアルで扱う監視位置 XL, XU, YL, YU, ZL, ZU との対応は以下のとおりです。

監視位置で「壁」選択時

X1 = XL, X2 = XU

Y1 = YL, Y2 = YU

Z1 = ZL, Z2 = ZU (垂直 6 軸型のみ)

監視位置で「制限領域」選択時

X1 = XU, X2 = XL

Y1 = YU, Y2 = YL

Z1 = ZL, Z2 = ZU (垂直 6 軸型のみ)

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_GetParam関数使用例

```
If SF_GetParam (SLS_1_HAND_EN) = 1 Then
    Print "SLS_1 hand speed monitoring is enabled."
EndIf
```

SF_GetParam\$関数

安全機能パラメーターのテキスト情報を返します。

書式

SF_GetParam\$(インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値、または定数で指定します。

戻り値

指定した情報の文字列値を返します。

解説

指定した安全機能パラメーター値を返します。

インデックス	定数	解説
1	SF_TOOLVERSION	設定ツールバージョン
2	SF_CHECKSUM	安全機能パラメーターチェックサム
3	SF_LAST_MODIFIED	安全機能パラメーター更新日
4	SF_ROBOT_MODEL_NAME	ロボットモデル名
5	SF_ROBOT_CHECKSUM	ロボットパラメーターチェックサム
6	SF_HOFS	物理的なロボットの原点に対するエンコーダーオフセット値 (Hofs)
7	SF_HOFS_LAST_MODIFIED	Hofs 更新日

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_GetParam\$関数使用例

```
String Checksum$
Checksum$ = SF_GetParam$(SF_CHECKSUM)
Print "Safety function parameter Checksum is ", Checksum$

> Print SF_GetParam$(SF_LAST_MODIFIED)
2022/01/01 00:00:00
```

SF_GetStatus 関数

安全機能の状態ビットを返します。

書式

SF_GetStatus (インデックス)

パラメーター

インデックス 検索する情報のインデックスを整数値で指定します。

戻り値

指定したインデックスの情報を整数値で返します。

解説

戻り値のビット情報は次の表のとおりです。

インデックス	ビット	値	解説
0	0-6	-	予約済
	7	&H80	Safety 基板の故障検知
1	0	&H1	SLS_1 機能有効
	1	&H2	SLS_2 機能有効
	2	&H4	SLS_3 機能有効
	3-7	-	予約済
2	0	&H1	SLP_A 機能有効
	1	&H2	SLP_B 機能有効
	2	&H4	SLP_C 機能有効
	3-6	-	予約済
	7	&H80	ソフト軸制限機能有効 (常時有効)
3	0	&H1	SAFETY_IN1 信号 High (機能オフ) *1
	1	&H2	SAFETY_IN2 信号 High (機能オフ) *1
	2	&H4	SAFETY_IN3 信号 High (機能オフ) *1
	3	&H8	SAFETY_IN4 信号 High (機能オフ) *1
	4	&H10	SAFETY_IN5 信号 High (機能オフ) *1
	5-7	-	予約済
4	0	&H1	SAFETY_OUT1 信号 High (機能オフ) *2
	1	&H2	SAFETY_OUT2 信号 High (機能オフ) *2
	2	&H4	SAFETY_OUT3 信号 High (機能オフ) *2
	3-7	-	予約済
5~11	0-7	-	予約済
12	0-7	-	STF_ID
13	0-7	-	STF_DET_L
14	0-7	-	STF_DET_U
15	0-7	-	予約済

*1 安全入力信号は負論理 (Active Low)です。

安全入力に信号レベルHighが入力されると、本関数は1を返し、安全入力に割り当てた機能は動作しません。

安全入力に信号レベルLowが入力されると、本関数は0を返し、安全入力に割り当てた機能が動作します。安全入力に機器を接続しないと、この状態になります。

*2 安全出力信号は負論理 (Active Low)です。

安全出力に割り当てたいいずれかの機能が動作状態の場合、安全出力が有効になり、安全出力の信号レベルはLowになり、本関数は0を返します。

安全出力に割り当てたいいずれの機能も動作状態ではない場合、安全出力が無効になり、安全出力の信号レベルはHighになり、本関数は1を返します。

安全出力にいずれの機能も割り当てない場合、安全出力の信号レベルはLowになり、本関数は0を返します。

STF_ID, および STF_DET_L, STF_DET_Hを使用するとエラー発生原因を確認することができます。

SPEL+を使った確認方法は後述します。

STF_ID, および STF_DET_L, STF_DET_H の情報は次の通りです。

STF_ID	エラー名	STF_DET_L	STF_DET_U
100	停止通知 安全入力	安全入力ポート SAFETY_IN1 0×01 SAFETY_IN2 0×02 SAFETY_IN3 0×04 SAFETY_IN4 0×08 SAFETY_IN5 0×10	不使用
101	停止通知 SLS_1 速度超過 関節	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
102	停止通知 SLS_1 速度超過 部位	部位 ハンド 0×01 / 肘 0×02 手首 0×04 / 肩 0×08 *1	不使用
103	停止通知 SLS_2 速度超過 関節	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
104	停止通知 SLS_2 速度超過 部位	部位 ハンド 0×01 / 肘 0×02 手首 0×04 / 肩 0×08 *1	不使用
105	停止通知 SLS_3 速度超過 関節	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
106	停止通知 SLS_3 速度超過 部位	部位 ハンド 0×01 / 肘 0×02 手首 0×04 / 肩 0×08 *1	不使用
107	停止通知 SLS_T 速度超過 関節	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
108	停止通知 SLS_T 速度超過 部位	部位 ハンド 0×01 / 肘 0×02 手首 0×04 / 肩 0×08 *1	不使用
109	停止通知 SLS_T2 速度超過 関節	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
110	停止通知 SLS_T2 速度超過 部位	部位 ハンド 0×01 / 肘 0×02 手首 0×04 / 肩 0×08 *1	不使用

STF_ID	エラー名	STF_DET_L	STF_DET_U
115	停止通知 SLP_A 位置違反 監視位置	監視位置 YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	関節番号 J6 0×08 J5 0×04 J3 0×02 J2 0×01
116	停止通知 SLP_B 位置違反 監視位置	監視位置 YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	関節番号 J6 0×08 J5 0×04 J3 0×02 J2 0×01
117	停止通知 SLP_C 位置違反 監視位置	監視位置 YL 0×01 / YU 0×02 XL 0×04 / XU 0×08 ZL 0×10 / ZU 0×20 *2	関節番号 J6 0×08 J5 0×04 J3 0×02 J2 0×01
118	停止通知 ソフト軸制限	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
121	停止通知 スイッチ入力	スイッチ番号 イネーブルスイッチ 0×01 非常停止スイッチ(ティーチペン ダント) 0×02 非常停止スイッチ(コントローラー 接続) 0×04	不使用
122	停止通知 モード制御	モード パラメーター通信許可 0×08 安全機能 (Safety 基板) 無効化 0×04 操作モード切替 0×02 パラメーター設定 認証済み 0×01	不使用
123	停止通知 減速監視	検出異常 減速異常 0×08, 0×04 減速完了 0×02 監視時間経過 0×01	不使用
124	停止通知 関節角度監視	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
131	故障停止通知 エンコーダー通信 異常	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
132	故障停止通知 位置異常	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用

STF_ID	エラー名	STF_DET_L	STF_DET_U
133	故障停止通知 入力二重化異常	異常検出部位 安全入力ポート SAFETY_IN 1 0×01 SAFETY_IN 2 0×02 SAFETY_IN 3 0×04 SAFETY_IN 4 0×08 SAFETY_IN 5 0×10 イネーブルスイッチ 0×20 非常停止スイッチ(ティーチペン ダント) 0×40 非常停止スイッチ(コントローラー 接続) 0×80	不使用
134	故障停止通知 出力二重化異常	異常検出部位 安全出力ポート SAFETY_OUT 1 0×01 SAFETY_OUT 2 0×02 SAFETY_OUT 3 0×04 STO 0×80	不使用
135	故障停止通知 Safety 基板異常	異常検出部位 通信バス 0×20 電源(3.3V) 0×08 電源(5V) 0×04 ウォッチドックタイマー検知 0×02 リレー溶着 0×01	不使用
136	故障停止通知 Safety 基板 MCU 異常	異常検出部位 シーケンスモニタ 0×10 CPU 0×08 RAM 0×04 プログラム ROM 0×02 データ ROM 0×01	DET_L = 0×01 のとき (データ ROM) 0×00 - 0×FE データ故障箇所 0×FF パラメーターの故障
137	故障停止通知 Safety 基板 二重化内部異常	異常検出部位 TCP 位置不一致 0×02 状態不一致 0×01	不使用
138	故障停止通知 エンコーダー 内部異常	関節番号 J1 0×01 / J2 0×02 J3 0×04 / J4 0×08 J5 0×10 / J6 0×20	不使用
139	故障停止通知 コントローラー 内部異常	異常検出部位 操作モード受信エラー 0×01	不使用

*1 安全機能マネージャーの安全速度監視の監視部位J2, J3, J5と、本マニュアルで扱う速度超過部位ハンド, 手首, 肘, 肩との対応は以下のとおりです。

水平多関節型

J2: 肘

J3: 該当なし

J5: 該当なし

Hand: ハンド

垂直6軸型

J2: 肩

J3: 肘

J5: 手首

Hand: ハンド

- *2 安全機能マネージャーの安全位置監視の監視位置X1, X2, Y1, Y2, Z1, Z2と、本マニュアルで扱う監視位置XL, XU, YL, YU, ZL, ZUとの対応は以下のとおりです。

監視位置で「壁」選択時

X1 = XL, X2 = XU

Y1 = YL, Y2 = YU

Z1 = ZL, Z2 = ZU (垂直6軸型のみ)

監視位置で「制限領域」選択時

X1 = XU, X2 = XL

Y1 = YU, Y2 = YL

Z1 = ZL, Z2 = ZU (垂直6軸型のみ)

- STF_ID、およびSTF_DET_L、STF_DET_Hを使用したエラー発生原因の確認方法
コマンドウィンドウ等で以下の順番でコマンドを入力、確認します。

> SF_GetStatus (12)

115 「停止通知 SLP_A 位置違反」によるエラーを表します。

> SF_GetStatus (13)

1 「YL」方向の監視位置を越えたことを表します。(STF_ID: 115の内容を確認します。)

> SF_GetStatus (14)

1 「J2軸」が越えたことを表します。(STF_ID: 115の内容を確認します。)

以上の情報をまとめると、「J2軸がSLP_AのYLの監視位置を越えた」ためエラーが発生したことがわかります。

エラー情報はEPSON RC+のシステムヒストリーにも記録されます。エラー発生原因は「追加情報」に記録されます。SPEL+ランゲージリファレンスのSPEL+ エラーメッセージを参照してください。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_GetStatus関数使用例

```
If (SF_GetStatus (3) And &H1) = &H1 Then
  Print "SAFETY_IN1 is High"
Else
  Print "SAFETY_IN1 is Low"
EndIf
```

SF_LimitSpeedS

SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能について速度調整値を設定、表示をします。

書式

SF_LimitSpeedS [SLS 番号 [, 速度調整値]]

パラメーター

SLS 番号 SLS 番号を整数値 (1~3)、または以下に示す定数で指定します。省略可能です。

定数	値	内容
SLS_1	1	SLS_1 有効時の速度調整機能
SLS_2	2	SLS_2 有効時の速度調整機能
SLS_3	3	SLS_3 有効時の速度調整機能
SLS_T	9	SLS_T 有効時の速度調整機能 *1
SLS_T2	10	SLS_T2 有効時の速度調整機能 *1

*1 SLS_TとSLS_T2については、本コマンドで速度調整値を設定することはできません。安全機能マネージャーで設定した監視速度で速度調整します。詳細は以下のマニュアルを参照してください。

ロボットコントローラー 安全機能マニュアル

速度調整値 速度を整数値 (0~10000、単位: mm/sec)で指定します。省略可能です。
0を指定した場合、自動で速度調整値が設定されます。初期値は0です。

解説

SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能について設定、表示します。本機能で速度調整される部位は、Tool コマンドで現在選択されているツール位置です。安全機能パラメーターで設定された TCP 位置ではありませんので注意してください。

指定した SLS 番号の速度調整値[mm/sec]を設定します。第 2 パラメーターが省略されているときは、指定した SLS 番号の速度調整値を表示します。

全パラメーターが省略されているときは、全ての SLS 番号の速度調整値を表示します。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。
安全機能オプションが有効のときのみ使用できます。

注意

SF_LimitSpeedS で速度調整される位置は Tool コマンドで設定されたツール位置です

SF_LimitSpeedS で設定する速度調整値は、Tool コマンドで設定されたツール位置の速度に適用されます。安全機能マネージャーの TCP 位置は、Tool コマンドに自動設定されません。Tool コマンドを利用し、適切な TCP 位置を設定してください。

また、速度の自動調整が正しく機能しない場合は、SF_PeakSpeedS、SF_RealSpeedS を使ってロボットアーム動作速度を計測し、SLS 監視速度 (安全機能パラメーター)を超えないようにロボットアーム動作速度を Speed、SpeedFactor、SpeedS などを使用して制御してください。

SF_LimitSpeedS使用例

- SLS_1の速度調整値を1500mm/secに設定する方法
SF_LimitSpeedS SLS_1, 1500

- SLS_2の速度調整値を表示する方法 (コマンドウィンドウを使用)
> **SF_LimitSpeedS** SLS_2
SLS_2: 400

- 全てのSLS番号の速度調整値を表示する方法 (コマンドウィンドウを使用)
> **SF_LimitSpeedS**
SLS_1: 1500
SLS_2: 400
SLS_3: 750
SLS_T: 250
SLS_T2: 3000

SF_LimitSpeedS 関数

SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能の速度調整値を返します。

書式

SF_LimitSpeedS (SLS 番号)

パラメーター

SLS番号 SLS番号を整数値、または以下に示す定数で指定します。

定数	値	内容
SLS_1	1	SLS_1 有効時の速度調整機能
SLS_2	2	SLS_2 有効時の速度調整機能
SLS_3	3	SLS_3 有効時の速度調整機能
SLS_T	9	SLS_T 有効時の速度調整機能
SLS_T2	10	SLS_T2 有効時の速度調整機能

戻り値

指定したSLS番号の速度調整値[mm/sec]を返します。

解説

SLS有効時に速度調整する機能の指定したSLS番号の速度調整値[mm/sec]を返します。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_LimitSpeedS関数 使用例

```
Integer i
i = SF_LimitSpeedS (SLS_1)
Print "SLS_1 limit speed is ", i
```

SF_LimitSpeedSEnable

SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能について On/Off の設定、設定状態を表示します。

書式

SF_LimitSpeedSEnable [SLS 番号 [, {On | Off}]]

パラメーター

SLS番号 SLS番号を整数値、または以下に示す定数で指定します。省略可能です。

定数	値	内容
SLS_1	1	SLS_1 有効時の速度調整機能
SLS_2	2	SLS_2 有効時の速度調整機能
SLS_3	3	SLS_3 有効時の速度調整機能
SLS_T	9	SLS_T 有効時の速度調整機能 *1
SLS_T2	10	SLS_T2 有効時の速度調整機能 *1

*1 SLS_TとSLS_T2については、本コマンドで速度調整機能を設定することはできません。SLS_Tは操作モードがTEACHおよびTEST T1の時に速度調整機能がOnになります。SLS_T2は操作モードがTEST T2の時に速度調整機能がOnになります。詳細は以下のマニュアルを参照してください。

ロボットコントローラー 安全機能マニュアル

On | Off 速度調整機能のOn/Offを指定します。省略可能です。デフォルト設定は、Onです。

戻り値

なし

解説

指定した SLS 番号が監視状態のときに Tool コマンドで設定されている位置の速度を調整する機能の On/Off を設定、表示します。On にすると、SLS が有効な場合に、ツール設定された位置の速度を調整します。Off にすると、SLS が有効な場合でも速度を調整しません。On/Off 設定に関わらず、SLS が無効な場合は、ツール位置の速度を調整しません。

第 2 パラメーターが省略されているときは、指定した SLS 番号の速度調整状態(On/Off)を表示します。全パラメーターが省略されているときは、全ての SLS 番号の速度調整状態(On/Off)を表示します。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。
安全機能オプションが有効のときのみ使用できます。

SF_LimitSpeedSEnable使用例

- SLS_1の速度調整機能を有効に設定する方法
SF_LimitSpeedSEnable SLS_1, On
- SLS_2の速度調整機能の状態を表示する方法 (コマンドウィンドウを使用)
> **SF_LimitSpeedSEnable** SLS_2
SLS_2: Off
- 全てのSLS番号の速度調整機能の状態を表示する方法 (コマンドウィンドウを使用)
> **SF_LimitSpeedSEnable**
SLS_1: On
SLS_2: Off
SLS_3: Off
SLS_T: On
SLS_T2: On

注意

SF_LimitSpeedSEnable は特異点回避, コンベヤートラッキング, 力制御と同時に使えません

SF_LimitSpeedSEnable のデフォルト設定は On です。特異点回避, コンベヤートラッキング, 力制御の動作中に本機能で速度調整を行うと動作エラー (それぞれ 4093, 4403, 5830)になります。
これらの動作で SLS を使う場合、動作開始前に SF_LimitSpeedSEnable を Off にしてください。
また、SF_PeakSpeedS, SF_RealSpeedS を使ってロボットアーム動作速度を計測し、SLS 監視速度 (安全機能パラメーター)を超えないようにロボットアーム動作速度を Speed, SpeedFactor, SpeedS などを使用して制御してください。

SF_LimitSpeedS で速度調整される位置は、Tool コマンドで設定されたツール位置です

SF_LimitSpeedSEnable が On の場合に適用される速度調整値(SF_LimitSpeedS で設定)は、Tool コマンドで設定されたツール位置の速度に適用されます。安全機能マネージャーの TCP 位置は、Tool コマンドに自動設定されません。Tool コマンドを利用し、適切な TCP 位置を設定してください。
また、速度の自動調整が正しく機能しない場合は、SF_PeakSpeedS, SF_RealSpeedS を使ってロボットアーム動作速度を計測し、SLS 監視速度 (安全機能パラメーター)を超えないようにロボットアーム動作速度を Speed, SpeedFactor, SpeedS などを使用して制御してください。

SF_LimitSpeedSEnable 関数

SLS 有効時に、Tool コマンドで設定されている位置の速度を調整する機能の状態を返します。

書式

SF_LimitSpeedSEnable(SLS 番号)

パラメーター

SLS 番号 SLS 番号を整数値 (1~3)、または以下に示す定数で指定します。

定数	値	内容
SLS_1	1	SLS_1 有効時の速度調整機能
SLS_2	2	SLS_2 有効時の速度調整機能
SLS_3	3	SLS_3 有効時の速度調整機能
SLS_T	9	SLS_T 有効時の速度調整機能
SLS_T2	10	SLS_T2 有効時の速度調整機能

戻り値

指定した SLS 番号の速度調整機能が On の場合、1 を返します。

指定した SLS 番号の速度調整機能が Off の場合、0 を返します。

解説

指定した SLS 番号が監視状態のときに TCP 動作速度を調整する機能の状態を返します。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_LimitSpeedSEnable関数使用例

```
If SF_LimitSpeedSEnable(SLS_1) = 0 Then
    Print "SLS_1 linked speed adjustment function is disabled."
Endif
```

SF_PeakSpeedS

速度監視点のピーク速度値を表示します。

書式

SF_PeakSpeedS [速度監視点番号]

パラメーター

速度監視点番号 速度監視点番号を整数値 (1~4) で指定します。省略可能です。

値	説明
1	ハンド
2	肘
3	手首
4	肩

戻り値

なし

解説

指定した速度監視点のピーク速度値を表示します。

パラメーターが省略されているときは、全ての速度監視点のピーク速度値を表示します。

ハンドの速度は安全機能マネージャーで設定した TCP オフセット位置での速度です。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_PeakSpeedS使用例

- ハンドのピーク速度値を表示する方法 (コマンドウィンドウを使用)

```
> SF_PeakSpeedS 1
250
```

- 全ての速度監視点のピーク速度値を表示する方法 (コマンドウィンドウを使用)

```
> SF_PeakSpeedS
250          150
```

表示順番は以下の通りです。

ハンド 肘

SF_PeakSpeedS 関数

速度監視点のピーク速度を返します。

書式

SF_PeakSpeedS (速度監視点番号)

パラメーター

速度監視点番号 速度監視点番号を整数値 (1~4)で指定します。省略可能です。

値	説明
1	ハンド
2	肘
3	手首
4	肩

戻り値

ピーク速度[mm/sec]を返します。

解説

指定した速度監視点のピーク速度を返します。
 ハンドの速度は安全機能マネージャーで設定した TCP オフセット位置での速度です。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_PeakSpeedS関数使用例

```
Print "Hand peak speed is ", SF_PeakSpeedS (1)
```

SF_PeakSpeedSClear

速度監視点のピーク速度値をクリアし、初期化します。

書式

SF_PeakSpeedSClear [速度監視点番号 1 [, 速度監視点番号 2]]

パラメーター

速度監視点番号 1 1 つ目の速度監視点番号を整数値 (1~4) で指定します。省略可能です。

速度監視点番号 2 2 つ目の速度監視点番号を整数値 (1~4) で指定します。省略可能です。

値	説明
1	ハンド
2	肘
3	手首
4	肩

戻り値

なし

解説

指定した速度監視点のピーク速度値をクリア(初期化)します。

パラメーターを指定しない場合は、全ての速度監視点のピーク速度値がクリアされます。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_PeakSpeedSClear使用例

- ハンドのピーク速度値をクリアする方法
SF_PeakSpeedSClear 1
- 全ての速度監視点のピーク速度値をクリアする方法
SF_PeakSpeedSClear

SF_RealSpeedS

速度監視点の現在速度を表示します。

書式

SF_RealSpeedS [速度監視点番号]

パラメーター

速度監視点番号 速度監視点番号を整数値 (1~4) で指定します。省略可能です。

値	説明
1	ハンド
2	肘
3	手首
4	肩

戻り値

なし

解説

指定した速度監視点の現在速度[mm/sec]を表示します。

ハンドの速度は安全機能マネージャーで設定した TCP オフセット位置での速度です。
パラメーターが省略されているときは、全ての速度監視点の現在速度を表示します。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_RealSpeedS使用例

- ハンドの現在速度を表示する方法 (コマンドウィンドウを使用)

```
> SF_RealSpeedS 1
250
```
- 全ての速度監視点の現在速度を表示する方法 (コマンドウィンドウを使用)

```
> SF_RealSpeedS
250          150
```

表示順番は以下の通りです。

ハンド 肘

SF_RealSpeedS 関数

速度監視点の現在速度を返します。

書式

SF_RealSpeedS(速度監視点番号)

パラメーター

速度監視点番号 速度監視点番号を整数値 (1~4) で指定します。省略可能です。

値	説明
1	ハンド
2	肘
3	手首
4	肩

戻り値

現在速度[mm/sec]を返します。

解説

指定した速度監視点の現在速度[mm/sec]を返します。

ハンドの速度は安全機能マネージャーで設定した TCP オフセット位置での速度です。

本コマンドは、Safety 基板を搭載したコントローラーで使用できます。

SF_RealSpeedS関数使用例

```
Print "Hand real speed is ", SF_RealSpeedS (1)
```

SFree

指定した関節をフリー状態にします。

書式

SFree 関節番号[, 関節番号,...]

パラメーター

関節番号 関節の番号 (1~9 の整数)を、式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

解説

SFree は、指定された関節のモーターパワーを遮断します。このときの状態をフリージョイント状態とよびます。この命令は、ダイレクトティーチングや、指定された関節のみフリージョイント状態にして組み込みを行う時などに使います。その関節のフリージョイント状態を解除するには SLock 命令、Motor On、または Motor Off を実行します。

SFree 命令を実行すると、ロボット制御パラメーターを初期化します。
詳細は、Motor On を参照してください。

注意

SFree で、システム設定の一部が初期化されます

SFree は安全のため、ロボットの動作速度や加減速度にかかわるパラメーター (Speed, SpeedS, Accel, AccelS など)および LimZ パラメーターを初期化します。詳細は、Motor On を参照してください。
ファームウェアバージョン 7.5.1.0 より前では、Motor ON 状態のみで SFree 可能です。
SFree は、ファームウェアバージョンによって、以下のように動作が変わります。

ファームウェア	SFree の可否
7.5.1.0 より前	モーターオン時のみ
7.5.1.0 以降	モーターオン時、およびモーターオフ時

重要事項

SFree を水平多関節型ロボット (RS シリーズを含む) の第 3 関節、および第 4 関節に用いるとき

水平多関節型ロボット (RS シリーズを含む) の第 3 関節には電磁ブレーキがかかるので、SFree を設定しても第 3 関節はすぐに動くわけではありません。手動で第 3 関節を動かすには、ロボットアーム上部にあるブレーキ解除スイッチを押し続ける必要があります。

また、第 4 関節にブレーキがある機種があります。第 4 関節にブレーキがある機種の場合、第 4 関節に電磁ブレーキがかかるので、SFree を設定しても第 4 関節はすぐに動きません。手動で第 4 関節を動かすには、ロボットアーム上部にあるブレーキ解除スイッチを押しながら動かします。

SFree を垂直 6 軸型ロボット (N シリーズを含む) に用いることはできません

垂直 6 軸型ロボット (N シリーズを含む) で SFree を行った場合、エラーが発生します。

手動でアームを動かす場合、Motor Off 後に Brake Off で電磁ブレーキを解除してください。

関節がフリージョイント状態にあるときに動作コマンドを実行する

関節がフリージョイント状態にあるときに動作コマンドを実行すると、コントローラーがデフォルト状態ではエラーになります。1 関節でもフリージョイント状態にあるときに動作命令を行うには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスをチェックします。

コンベヤートラッキング中に SFree を使用しないでください

コンベヤートラッキング中に SFree を使用した場合、エラー 5057, 5058 などが発生します。コマンド Cnv_AbortTrack などコンベヤートラッキングを終了させた後、SFree を使用してください。

参照

Brake, LimZ, Motor, SFree 関数, SLock

SFree 使用例

SFree 命令の簡単な使用例です。この例で作動するには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスがチェックされていない必要があります。

```
Function GoPick
    Speed pickSpeed
    SFree 1, 2    'J1 と J2 をフリージョイント状態にして
                  パーツ取りつけのため Z と U 関節を動かす
    Go pick
    SLock 1, 2    'J1 と J2 のフリージョイント状態を解除する
Fend
```


SFree 関数

指定した関節のフリージョイント状態を返します。

書式

SFree(関節番号)

パラメーター

関節番号 フリージョイント状態をチェックする関節の番号 (整数)を、数値または式で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

True	フリージョイント状態
False	非フリージョイント状態

参照

SFree

SetFree 使用例

```
If SFree(1) Then  
    Print "Joint 1 is free"  
EndIf
```

Sgn 関数

数値の符号を返します。

書式

Sgn(演算数)

パラメーター

演算数 数値を指定します。

戻り値

1 : 演算数は正の値

0 : 演算数は 0

-1 : 演算数は負の値

解説

Sgn 関数は、数値の符号を返します。

参照

Abs, And, Atan, Atan2, Cos, Int, Mod, Or, Not, Sin, Sqr, Str\$, Tan, Val, Xor

Sgn 関数使用例

次の例はコマンドウィンドウで Sgn を使った例です。

```
>print sgn(123)
1
>print sgn(-123)
-1
>
```

Short

Short 型の変数を宣言します。(2 バイト整数型変数)

書式

Short 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

Short は整数型変数の宣言をします。整数型変数の範囲は-32768~32767 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, String, UByte, UInt32, UInt64, UShort

Short 使用例

次は Short を使って整数型の変数を宣言するプログラム例です。

```
Function shortttest
    Short A(10)           'Short 型の 1 次元配列
    Short B(10, 10)       'Short 型の 2 次元配列
    Short C(5, 5, 5)      'Short 型の 3 次元配列
    Short var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

ShutDown

EPSON RC+ をシャットダウンし、Windows をシャットダウン、または再起動します。

書式

ShutDown [モード] [, Forced]

パラメーター

モード 下記に示すモードを整数値で設定します。

定数	値	内容
モード省略	-1	ダイアログが表示され、ユーザーがシャットダウン方法を選びます。
SHUTDOWN_ALL	0	EPSON RC+ および Windows をシャットダウンします。
SHUTDOWN_RESTART	1	EPSON RC+ をシャットダウンし、Windows を再起動します。
SHUTDOWN_EPSONRC	2	EPSON RC+ をシャットダウンします。

Forced 強制シャットダウンするときに設定します。省略可能です。

解説

Shutdown を使用して RC+ を終了したり、プログラムから Windows をシャットダウンまたは再起動することができます。Forced パラメーターを使用すると、強制的にシャットダウンします。

注意

タスク実行中で強制シャットダウンすると、データが損失する可能性があります。
シャットダウンする前にデータを保存してください。

使用可能なコントローラーについて

Shutdown コマンドは、T/VT シリーズでは使用できません。

仮想コントローラーでコントローラー設定が“連携モード”に設定されている場合

仮想コントローラーでコントローラー設定が“連携モード”に設定されている場合、ShutDown を実行すると、バックアップ変数が保存されません。バックアップ変数の保存が必要な場合は、ShutDown は使用しないでください。

参照

Restart

ShutDown 使用例

ShutDown 0 ' EPSON RC+ と Windows のシャットダウン

ShutDown 関数

EPSON RC+ をシャットダウンし、Windows をシャットダウン、または再起動します。

書式

ShutDown(モード[, Forced])

パラメーター

モード 下記に示すモードを整数値で設定します。

定数	値	内容
	-1	ダイアログが表示され、ユーザーがシャットダウン方法を選びます。
SHUTDOWN_ALL	0	EPSON RC+ および Windows をシャットダウンします。
SHUTDOWN_RESTART	1	EPSON RC+ をシャットダウンし、Windows を再起動します。
SHUTDOWN_EPSONRC	2	EPSON RC+ をシャットダウンします。

Forced 強制シャットダウンするときに設定します。省略可能です。

解説

Shutdown を使用して RC+を終了したり、プログラムから Windows をシャットダウンまたは再起動することができます。Forced パラメーターを使用すると、強制的にシャットダウンします。

注意

タスク実行中で強制シャットダウンすると、データが損失する可能性があります。
シャットダウンする前にデータを保存してください。

使用可能なコントローラーについて

Shutdown 関数は、T/VT シリーズでは使用できません。

仮想コントローラーでコントローラー設定が“連携モード”に設定されている場合

仮想コントローラーでコントローラー設定が“連携モード”に設定されている場合、ShutDown 関数を実行すると、バックアップ変数が保存されません。バックアップ変数の保存が必要な場合は、ShutDown 関数は使用しないでください。

戻り値

以下の整数値を返します。

- 1 ダイアログを表示した場合、ユーザーがキャンセルを選択した時に返します。
- 0 シャットダウンが失敗した時に返します。
- 1 シャットダウンが成功した時に返します。

ShutDown 関数使用例

```
If Shutdown(SHUTDOWN_EPSONRC) = 1 Then  
    Print "Shutdown: OK"  
Else  
    Print "Shutdown: NG"  
EndIf
```

Signal

WaitSig 命令を実行中のタスクに信号を送ります。

書式

Signal シグナル番号

パラメーター

シグナル番号 0~63 の、送信するシグナル番号を指定します。

解説

Signal はマルチタスクの同期をとります。WaitSig 実行前の Signal は無視されます。

参照

WaitSig

Signal 使用例

```
Function Main
  Xqt 2, SubTask
  Call InitSys
  Signal 1

Fend

Function SubTask
  WaitSig 1

Fend
```

SimGet

シミュレーターの各種オブジェクトのプロパティの設定値を取得します。

書式

SimGet Object.Property, Var

SimGet Robot.Hand.Propoerty, Var

パラメーター

Object	プロパティの値を取得するオブジェクト名を示す文字列変数
Robot	“Hand”で指定されたハンドが取り付けられているロボット名を示す文字列変数
Hand	プロパティの値を取得するハンド名を示す文字列変数
Property	値を取得するプロパティの名前。プロパティについては後述します。
Var	返される値を示す変数

解説

この命令は、シミュレーターにおける各種オブジェクトのプロパティの設定値を取得するときに使用します。

以下に示すプロパティを指定することで、オブジェクトの設定値を取得することができます。

プロパティ	説明	単位	データタイプ	戻り値
PositionX	X 座標位置を取得します	ミリメートル (mm)	Double	
PositionY	Y 座標位置を取得します	ミリメートル (mm)	Double	
PositionZ	Z 座標位置を取得します	ミリメートル (mm)	Double	
RotationX	X 軸回転角度を取得します	度 (degree)	Double	
RotationY	Y 軸回転角度を取得します	度 (degree)	Double	
RotationZ	Z 軸回転角度を取得します	度 (degree)	Double	
CollisionCheck	衝突検出の有効・無効を取得します	-	Boolean	Trueもしくは False
CollisionCheckSelf	ロボットの自己衝突検出の有効/無効を取得します	-	Boolean	Trueもしくは False
Visible	表示/非表示の状態を取得します	-	Boolean	Trueもしくは False
Type	オブジェクトのタイプを取得します	-	Integer	Layout: 0 Part: 1 Mounted Device: 3
HalfSizeX	Box オブジェクトの X 方向の長さを取得します	ミリメートル (mm)	Double	
HalfSizeY	Box オブジェクトの Y 方向の長さを取得します	ミリメートル (mm)	Double	
HalfSizeZ	Box オブジェクトの Z 方向の長さを取得します	ミリメートル (mm)	Double	
HalfSizeHeight	Plane オブジェクトの長さを取得します	ミリメートル (mm)	Double	

プロパティ	説明	単位	データタイプ	戻り値
HalfSizeWidth	Plane オブジェクトの幅を取得します	ミリメートル (mm)	Double	
PlaneType	Plane オブジェクトのタイプを取得します	-	Integer	Horizontal: 0 Vertical: 1
Radius	Sphere オブジェクトまたは Cylinder オブジェクトの半径を取得します	ミリメートル (mm)	Double	
Height	Cylinder オブジェクトの高さを取得します	ミリメートル (mm)	Double	
Name	オブジェクト名を取得します		String	
Color	オブジェクトの表示色を取得します		String	色名もしくは 16 進カラーコード (ARGB)

以下の表に示す組み合わせでプロパティを取得することができます。

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	○	○	○	○	○	○	○	○
PositionY	○	○	○	○	○	○	○	○
PositionZ	○	○	○	○	○	○	○	○
RotationX	○	○	○	○	○	○	○	○
RotationY	○	○	○	○	○	○	○	○
RotationZ	○	○	○	○	○	○	○	○
CollisionCheck	○	○	○	○	○	○	○	○
CollisionCheckSelf	○	-	-	-	-	-	-	-
Visible	-	○	○	○	○	○	○	○
Type	-	-	○	○	○	○	○	-
HalfSizeX	-	-	○	-	-	-	-	-
HalfSizeY	-	-	○	-	-	-	-	-
HalfSizeZ	-	-	○	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	○	-	-
HalfSizeWidth	-	-	-	-	-	○	-	-
PlaneType	-	-	-	-	-	○	-	-
Radius	-	-	-	○	○	-	-	-
Height	-	-	-	-	○	-	-	-
Name	○	○	○	○	○	○	○	○
Color	-	-	○	○	○	○	-	-

参照

SimSet

SimGet 使用例

  `SBox_1 オブジェクトの X 座標値を取得する

Double boxPosX

SimGet SBox_1.**PositionX**, boxPosX

  `SBox_1 オブジェクトの表示・非表示の状態を取得する

Boolean boxVisible

SimGet SBox_1.**Visible**, boxVisible

  `SBox_1 オブジェクトのタイプを取得する

Integer boxType

SimGet SBox_1.**Type**, boxType

SimSet

シミュレーターの各種オブジェクトのプロパティの設定をします。また、ロボットの動作とオブジェクトの操作、シミュレーターの設定に関する操作をします。

書式

- (1) オブジェクトのプロパティ設定
SimSet Object.Property, Value
SimSet Robot.Hand.Property, Value
- (2) ロボットの動作設定 (Pick および Place)
SimSet Robot.Pick, Object [,Tool]
SimSet Robot.Place, Object
- (3) オブジェクトの操作設定 (親オブジェクトの指定)
SimSet Object.SetParent [, ParentObject]
- (4) シミュレーターの設定 (衝突検出のリセット)
SimSet ResetCollision

パラメーター

- (1) オブジェクトのプロパティ設定

Object	プロパティの値を設定するオブジェクト名を示す文字列変数
Robot	“Hand”で指定されたハンドが取り付けられているロボット名を示す文字列変数
Hand	プロパティの値を設定するハンド名を示す文字列変数
Property	新たに値を設定するプロパティの名前。プロパティについては後述します。
Value	新しい値の式。データタイプはプロパティによります。
- (2) ロボット動作設定 (Pick および Place)

Robot	PickもしくはPlaceするロボット名を示す文字列変数
Object	PickもしくはPlaceされるオブジェクト名を示す文字列変数
Tool	Pick時に利用するTool番号を示す式
- (3) オブジェクトの操作設定 (親オブジェクトの指定)

Object	親オブジェクトを設定するオブジェクト名を示す文字列変数
ParentObject	親オブジェクト名を示す文字列変数

解説

この命令は、シミュレーターにおける各種オブジェクトのプロパティ設定や操作、ロボット動作、シミュレーター設定の変更をするときに使用します。

- (1) オブジェクトのプロパティ設定
 以下に示すプロパティを指定することで、オブジェクトの設定をすることができます。

プロパティ	説明	単位	データタイプ	設定値
PositionX	X座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000
PositionY	Y座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000
PositionZ	Z座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000

プロパティ	説明	単位	データタイプ	設定値
RotationX	X 軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
RotationY	Y 軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
RotationZ	Z 軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
CollisionCheck	衝突検出の有効・無効を設定します	-	Boolean	True もしくは False
CollisionCheckSelf	ロボットの自己衝突検出の有効・無効を設定します	-	Boolean	True もしくは False
Visible	表示・非表示を設定します	-	Boolean	True もしくは False
HalfSizeX	Box オブジェクトの X 方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeY	Box オブジェクトの Y 方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeZ	Box オブジェクトの Z 方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeHeight	Plane オブジェクトの長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeWidth	Plane オブジェクトの幅を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
PlaneType	Plane オブジェクトのタイプを設定します	-	Integer	Horizontal: 0 Vertical: 1
Radius	Sphere オブジェクトまたは Cylinder オブジェクトの半径を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
Height	Cylinder オブジェクトの高さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
Name	オブジェクト名を設定します		String	
Color	オブジェクトの表示色を設定します		String	色名もしくは 16 進カラーコード (ARGB)

以下の表に示す組み合わせでプロパティを設定することができます。

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	○	○	○	○	○	○	○	○
PositionY	○	○	○	○	○	○	○	○
PositionZ	○	○	○	○	○	○	○	○
RotationX	○	○	○	○	○	○	○	○
RotationY	○	○	○	○	○	○	○	○
RotationZ	○	○	○	○	○	○	○	○
CollisionCheck	○	○	○	○	○	○	○	○
CollisionCheckSelf	○	-	-	-	-	-	-	-
Visible	-	○	○	○	○	○	○	○
HalfSizeX	-	-	○	-	-	-	-	-
HalfSizeY	-	-	○	-	-	-	-	-
HalfSizeZ	-	-	○	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	○	-	-
HalfSizeWidth	-	-	-	-	-	○	-	-
PlaneType	-	-	-	-	-	○	-	-
Radius	-	-	-	○	○	-	-	-
Height	-	-	-	-	○	-	-	-
Name	○	○	○	○	○	○	○	○
Color	-	-	○	○	○	○	-	-

(2) ロボットの動作設定 (Pick および Place)

以下に示すロボットの動作を設定することができます。

Pick

“Robot”で指定されたロボットが、“Object”で指定されたオブジェクトを把持する動作を行います。

把持されたオブジェクトは、そのロボットのパーツとして登録されることになります。また、“Tool”に任意のツール番号を指定することで、指定されたツールを利用して把持動作を行います。“Tool”の指定が省略された場合は、Tool0を利用して把持動作を行います。

すでにパーツとして登録されているオブジェクト、もしくはアーム取付機器として設定されているオブジェクトを把持することはできません。また、カメラについても把持することはできません。

Place

“Robot”で指定されたロボットが、“Object”で指定されたオブジェクトを配置する動作を行います。配置されたオブジェクトは、そのロボットのパーツとしての登録が解除されます。

すでにパーツとしての登録が解除されているオブジェクトは配置することはできません。

以下の表に示す組み合わせで、オブジェクトの把持もしくは配置をすることができます。

動作	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Pick	-	-	○	○	○	○	○	-
Place	-	-	○	○	○	○	○	-

- (3) オブジェクトの操作設定 (親オブジェクトの指定)
以下に示すオブジェクトの操作を設定することができます。

SetParent

“Object”で指定されたオブジェクトに対して、“ParentObject”で指定されたオブジェクトを親オブジェクトとして設定します。“ParentObject”は省略することができます。この場合、“Object”で指定されたオブジェクトが親オブジェクトとなります。例えば、“Object”で指定されたオブジェクトが、何らかのオブジェクトの子オブジェクトであった場合には、子オブジェクトとしての設定が解除されることになります。

また、“Object”で指定されたオブジェクトがパーツもしくはアーム取付機器として設定されているオブジェクトの場合は、親オブジェクトの指定をすることはできません。

SetParentを指定可能なオブジェクトは以下の表になります。カメラオブジェクトについては、固定カメラとして設定されているオブジェクトのみ SetParent を利用することができます。

操作	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
SetParent	-	-	○	○	○	○	○	○

また、以下に示す表の組み合わせで SetParent を利用することができます。

		ParentObject (親オブジェクト)							
		Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Object (子オブジェクト)	Robot	-	-	-	-	-	-	-	-
	Hand	-	-	-	-	-	-	-	-
	Box	-	-	○	○	○	○	○	○
	Sphere	-	-	○	○	○	○	○	○
	Cylinder	-	-	○	○	○	○	○	○
	Plane	-	-	○	○	○	○	○	○
	CAD	-	-	○	○	○	○	○	○
	Camera	-	-	○	○	○	○	○	○

- (4) シミュレーターの設定 (衝突検出のリセット)
以下に示すシミュレーター設定の変更ができます。

ResetCollision

衝突検出のリセットを行います。ResetCollision を実行後に、ロボットとオブジェクトの間に衝突が発生していなければ、衝突状態が解除され、シミュレーターの 3D 表示が更新されます。ロボットとオブジェクトの間に衝突が発生している場合は、衝突状態は解除されず、シミュレーターの 3D 表示も更新されません。

参照

SimGet

SimSet 使用例

＼SBox_1 オブジェクトの X 座標値を 100.0mm に設定する

SimSet SBox_1.**PositionX**, 100.0

＼Robot1 で SBox_1 を Tool1 で把持する

SimSet Robot1.**Pick**, SBox_1, 1

＼Robot1 で把持している SBox_1 を配置する

SimSet Robot1.**Place**, SBox_1

＼SBox_1 の親オブジェクトに CAD_1 を設定する

SimSet SBox_1.**SetParent**, CAD_1

＼SBox_1 を親オブジェクトにする

SimSet SBox_1.**SetParent**

＼衝突検出をリセットする

SimSet **ResetCollision**

Sin 関数

指定した数値のサイン値を返します。

書式

Sin(角度)

パラメーター

角度 角度を実数値で指定します。

戻り値

指定した角度を表わすサイン値を数値で返します。

解説

指定した角度 (ラジアン) のサイン値を返します。パラメーターの単位はラジアンでなければなりません。戻り値の範囲は-1~1 です。

ラジアンを角度に変換するためには RadToDeg 関数を使います。

参照

Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sqr, Str\$, Tan, Val

Sin 関数使用例

Sin 関数を使った例です。

```
Function sintest
  Real x
  Print "Please enter a value in radians:"
  Input x
  Print "Sin of ", x, " is ", Sin(x)
Fend
```


SingularityAngle

特異姿勢通過機能に必要な特異点近傍角度を設定します。

書式

SingularityAngle {角度設定値}

パラメーター

角度設定値 垂直 6 軸型ロボット (N シリーズを含む) の手首特異姿勢近傍を判断するための第 5 関節角度 (0.1 以上の実数、単位は deg) を式または数値で指定します。

結果

パラメーターを省略すると、現在の SingularityAngle 設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。
デフォルト値は 10 deg が設定されています。特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より小さい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを変更する必要はありませんが、特異姿勢通過時にエラーが発生するようとき、エラーを抑制できることがあります。

SingularityAngle の設定値を変更した場合、次回コントローラー起動時まで有効です。

参照

AvoidSingularity, SingularityAngle 関数, SingularitySpeed

SingularityAngle 使用例

SingularityAngle 7.0 `特異姿勢近傍角度設定を 7 度にする

SingularityAngle 関数

SingularityAngle の設定値を返します。

書式

SingularityAngle

戻り値

設定されている特異姿勢近傍角度 (単位: deg)を返します。

参照

AvoidSingularity, SingularityAngle, SingularitySpeed, SingularitySpeed 関数

SingularityAngle 関数使用例

```
Real currSingularityAngle  
currSingularityAngle = SingularityAngle
```

SingularityDist

特異姿勢通過機能に必要な特異点近傍距離を設定します。

書式

SingularityDist {距離設定値}

パラメーター

距離設定値	垂直 6 軸型ロボット (N シリーズを含む)および RS シリーズの肩特異姿勢近傍を判断するための P 点と第 1 関節回転軸距離 (0 以上の実数、単位は mm)を式または数値で指定します。

結果

パラメーターを省略すると、現在の SingularityDist 設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。

デフォルト値は 30 mm が設定されています。肩特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より小さい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを変更する必要はありませんが、特異姿勢通過時にエラーが発生するようなとき、エラーを抑制できることがあります。

SingularityDist の設定値を変更した場合、次回コントローラー起動時まで有効です。

参照

AvoidSingularity, SingularityAngle, SingularityAngle 関数, SingularityDist 関数, SingularitySpeed, SingularitySpeed 関数

SingularityDist 使用例

SingularityDist 10.0 `特異姿勢近傍距離設定を 10 mm にする

SingularityDist 関数

SingularityDist の設定値を返します。

書式

SingularityDist

戻り値

設定されている特異姿勢近傍距離 (単位: mm)を返します。

参照

SingularityDist, AvoidSingularity, SingularityAngle, SingularityAngle 関数, SingularitySpeed, SingularitySpeed 関数

SingularityDist 関数使用例

```
Real currSingularityDist  
currSingularityDist = SingularityDist
```

SingularitySpeed

特異姿勢通過機能に必要な特異点近傍角速度を設定します。

書式

SingularitySpeed {角速度設定値}

パラメーター

角速度度設定値 垂直 6 軸ロボット (N シリーズを含む) の手首特異姿勢近傍を判断するための第 4 関節角速度の最大角速度に対する割合 (0.1 以上の実数、単位は%) を式または数値で指定します。

結果

パラメーターを省略すると、現在の SingularitySpeed 設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。
デフォルト値は 10 % が設定されています。特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より大きい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを調整し、変更する必要はありませんが、特異姿勢通過時にエラーが発生するようなとき、小さい値を設定してエラーを抑制できることがあります。

SingularitySpeed の設定値を変更した場合、次回コントローラー起動時まで有効です。

参照

AvoidSingularity 関数, SingularityAngle, SingularitySpeed

SingularitySpeed 使用例

SingularitySpeed 30.0 `特異姿勢近傍角速度の割合を 30 % に設定する

SingularitySpeed 関数

SingularitySpeed の設定値を返します。

書式

SingularitySpeed

戻り値

設定されている特異姿勢近傍角速度の最大角速度に対する割合 (単位: %)を返します。

参照

SingularitySpeed, SingularityAngle, AvoidSingularity

SingularitySpeed 関数使用例

```
Real currSingularitySpeed  
currSingularitySpeed = SingularitySpeed
```

SLock

指定された関節のフリージョイント状態を解除します。

書式

SLock 関節番号 [, 関節番号,...]

パラメーター

関節番号 関節の番号 (1~9 の整数)を式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

解説

SLock は、ダイレクトティーチングやワークの装着などのため、SFree 命令でフリージョイント状態になっていた関節のフリージョイント状態を解除します。

関節番号を省略すると、全関節のフリージョイント状態を解除します。

第 3 関節の SLock を行くと、電磁ブレーキが解除されます。

SLock 命令を実行すると、ロボット制御パラメーターを初期化します。
詳細は、Motor On を参照してください。

垂直 6 軸型ロボット (N シリーズを含む)は SFree 命令でフリージョイント状態にできません。また、SLock を実行した場合、エラーが発生します。

参照

Brake, LimZ, Reset, SFree

SLock 使用例

SLock 命令を使った例です。この例で作動するには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスが有効にされていなければなりません。

```
Function test
.
.
.
SFree 1, 2      'J1 と J2 をフリージョイント状態にして
                パール取りつけのため Z と U 関節を動かす
Go P1
SLock 1, 2    'J1 と J2 のフリージョイント状態を解除する
.
.
.
Fend
```

SoftCP

SoftCP 動作モードを設定します。

書式

SoftCP { On | Off }

パラメーター

On | Off On: SoftCP 動作モードを有効にします。
Off: SoftCP 動作モードを解除します。

解説

SoftCP 動作モードは、高加減速度で CP 動作を実行したときの振動的な動作を抑制します。通常の CP 動作は、軌跡追従性能や等速動作性能を重視する設定がされています。そのため、高加減速度で動作したとき振動的になることがあります。この振動を抑制するためには SpeedS や AccelS といった加減速度設定を低く制限することが有効でした。しかし、アプリケーションによっては、高い軌跡追従性や等速性よりも、高加減速度でより振動が少ない CP 動作を実行することが要求されます。SoftCP 動作モードでは、CP 動作時の軌跡追従性や等速性を通常の CP 動作より抑えることで、高加減速度が設定された CP 動作の振動をより少なくします。

SoftCP 動作モードは、以下の CP 動作命令に有効です。
Move, BMove, TMove, Arc, Arc3, CVMove, Jump3CP

通常の CP 動作によって振動が問題にならない場合や、軌跡追従性や等速性が重視されるアプリケーションでは SoftCP 動作モードを使用しないでください。

重要事項

CP On 時に CP 動作と PTP 動作を接続する場合

下記のように CP 動作と PTP 動作を接続する場合、SoftCP を有効にしてください。有効にしない場合、動作によってはロボットから異音が発生することがあります。接続終了後、SoftCP を無効にしてください。

```
SoftCP On
Go P1 CP
Move P2
SoftCP Off
```

参照

SoftCP 関数

SoftCP 使用例

```
SoftCP On
Move P1
Move P2
SoftCP Off
```


SoftCP 関数

SoftCP 動作モードの状態を返します。

書式

SoftCP

戻り値

- 0 = SoftCP 動作モードオフ
- 1 = SoftCP 動作モードオン

参照

SoftCP

SoftCP 関数使用例

```
If SoftCP = Off Then  
    Print "SoftCP is off"  
EndIf
```

Space\$ 関数

指定した数だけのスペース文字列を返します。

書式

Space\$(指定数)

パラメーター

指定数 戻り値として返すスペースの数を指定します。

戻り値

指定した数のスペース文字列

解説

Space\$ は指定した数のスペース文字列を返します。Space\$ は最大 255 文字のスペース (文字列変数の許容範囲)を返します。

Space\$ は、別の文字列の前、後、中間にスペースを加えるために使います。

参照

Asc, Chr\$, InStr, Left\$, Len, LSet\$, Mid\$, Right\$, RSet\$, Str\$, Val

Space\$ 関数使用例

```
> Print "XYZ" + Space$(1) + "ABC"
XYZ ABC

> Print Space$(3) + "ABC"
ABC

>
```

Speed

Go, Jump, Pulse 命令などによる PTP 動作の、速度設定と表示をします。

書式

- (1) Speed 速度設定値 [, 退避速度, 接近速度]
- (2) Speed

パラメーター

速度設定値	最大動作速度 (PTP 動作)に対する割合 (1~100 の整数、単位: %)を式または数値で指定します。
退避速度	Jump 命令時の退避動作の速度 (1~100 の整数、単位: %)を、式または数値で指定します。省略可能です。Jump コマンド時のみ設定可能です。
接近速度	Jump 命令時の接近動作の速度 (1~100 の整数、単位: %)を、式または数値で指定します。省略可能です。Jump コマンド時のみ設定可能です。

結果

パラメーターを省略すると、現在の Speed 設定値を表示します。

解説

Speed は、すべての PTP 動作命令について、速度を指定します。これには Go, Jump, Pulse などの動作命令に関する速度設定が含まれます。速度設定は、最大速度に対する割合 (%)を 1~100 の整数で指定します。“100”を指定すると、最大速度で動作します。

退避速度および接近速度は、Jump コマンドのみに対して適用されます。省略すると、速度設定値が、退避速度と接近速度の設定値となります。

下記のいずれかの場合、Speed 値は、初期化されます。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

ローパワーモード時には、Speed 設定値は、デフォルト値よりも低い値になります。コマンドウィンドウから、またはプログラム中で、それよりも高い値が入れられても、デフォルト値が設定されます。ハイパワーモード時には、動作速度は、Speed で設定された値になります。

より速い動作速度が必要な場合は、Power High でハイパワーモードを設定し、安全扉を閉めます。安全扉が開いていると、Speed 値は、デフォルト値に変更されます。

ローパワーモード時に **Speed** が実行されると、メッセージが表示されます。下記の例では、ロボットがローパワーモードにあるので、**Speed** の設定値が “80” であっても、ロボットはデフォルト値 “5” で動作します。

```
> speed 80
> speed
Low Power Mode
  80
  80      80
>
```

参照

Accel, Go, Jump, Power, Pass, Pulse, SpeedS

Speed 使用例

Speed は、コマンドウィンドウでも、プログラム中でも、用いることができます。下記に両方の場合の使用例を示します。

```
Function speedtst
  Integer slow, fast, i
  slow = 10
  fast = 100
  For i = 1 To 10
    Speed slow
    Go P0
    Go P1
    Speed fast
    Go P0
    Go P1
  Next i
Fend
```

コマンドウィンドウから **Speed** 値を設定する例です。

```
> Speed 100,100,50      'Z 関節の降下速度を 50%に設定
> Speed 50
> Speed
Low Power State: Speed is limited to 5
  50
  50      50
>
```

Speed 関数

現在設定されている PTP 動作速度を返します。

書式

Speed [(パラメーター番号)]

パラメーター

パラメーター番号 下記に示す番号から、設定値番号を 1 つ指定します。
省略した場合は 1 が指定されたとみなします。

- 1: PTP 動作速度
- 2: JUMP 上昇速度
- 3: JUMP 下降速度

戻り値

1~100 の整数値を返します。

参照

Speed

Speed 関数使用例

```
Function SpeedEx
  Integer savSpeed

  savSpeed = Speed(1)
  Speed 50
  Go pick
  Speed savSpeed
End
```

SpeedFactor

ロボット動作の速度係数を設定、および設定値を返す。

書式

- (1) SpeedFactor 速度比率
- (2) SpeedFactor

パラメーター

速度比率 ロボット動作の速度比率を (1~100 の整数、単位: %)を式または数値で指定します。

結果

パラメーターが省略されているときは、現在の SpeedFactor 値を表示します。

解説

SpeedFactor は、コントローラーに設定されているすべてのロボット、すべての動作に掛かる速度係数です。通常 SpeedFactor は 100%が設定されており、各ロボット、各動作命令の速度は Speed, SpeedS 等で設定します。SpeedFactor はすべてのロボットのすべての動作に対し一定の速度比率で速度を変更したときに有効です。例えば、Speed 80%の動作は、速度比率 50%を設定したとき、動作速度 40%となります。

SpeedFactor はロボット動作の加減速度のバランスを考慮するため加速度についても同じ割合で変更します。

SpeedFactor は、オペレーターウィンドウの速度比率の設定と等価であり、連動して変化します。

SpeedFactor は、コントローラー起動時にデフォルト値 100%に初期化されます。

参照

SpeedFactor 関数

SpeedFactor 使用例

```
Function main
  Motor On
  Power High
  SpeedFactor 80

  Speed 100; Accel 100,100
  Go P1           ' Speed 80; Accel 80,80 で動作する

  Speed 50; Accel 50,50
  Go P2           ' Speed 40; Accel 40,40 で動作する
Fend
```

SpeedFactor 関数

SpeedFactor 命令の設定値を返します。

書式

SpeedFactor

戻り値

SpeedFactor 命令の設定値を整数値で返します。

参照

SpeedFactor

SpeedFactor 関数使用例

```
Real savSpeedFactor

savSpeedFactor = SpeedFactor
SpeedFactor 80
Go P1
Go P2
SpeedFactor savSpeedFactor
```

SpeedR

ROTを使っているときの CP 動作のツール姿勢変化速度の設定と表示をします。

書式

- (1) SpeedR 速度設定値
- (2) SpeedR

パラメーター

速度設定値 CP 動作時のツール姿勢変化速度 (0.1 以上の実数、単位は deg/sec)を式または数値で指定します。

パラメーターの有効設定値: 0.1~1000

結果

パラメーターを省略すると、現在の SpeedR 設定値を表示します。

解説

この命令は、Move, Arc, Arc3, BMove, TMove, Jump3CP の動作命令で ROT 修飾パラメーターを指定したときのみ有効となります。

下記のいずれかの場合、SpeedR 値は、デフォルト値 (低速)に初期化されます。

コントローラー起動時 Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

参照

AccelR, Arc, Arc3, BMove, Jump3CP, Power, SpeedR 関数, TMove

SpeedR 使用例

SpeedR 200

SpeedR 関数

ツール姿勢変化速度を返します。

書式

SpeedR

戻り値

設定された速度を実数値 (単位: deg/sec)で返します。

参照

AccelR, SpeedR

SpeedR 関数使用例

```
Real currSpeedR
```

```
currSpeedR = SpeedR
```

SpeedS

Move, Arc, Arc3, Jump3, Jump3CP など CP 動作時のアームの速度の設定と表示を行います。

書式

- (1) SpeedS 速度設定値 [, 退避速度, 接近速度]
 (2) SpeedS

パラメーター

- 速度設定値** 速度を式または数値 (整数、単位: mm/sec) で指定します。
退避速度 Jump3 の退避速度を示す実数または式 (単位: mm/sec) を指定します。省略可能です。
接近速度 Jump3 の接近速度を示す実数または式 (単位: mm/sec) を指定します。省略可能です。

パラメーターの有効設定値:

N シリーズ以外: 0.1~2000

N シリーズ: 0.1~1120

結果

パラメーターを省略すると、現在の SpeedS 値を表示します。

解説

SpeedS は、CP 動作 (Move および Arc) 命令時の速度を指定します。

SpeedS 値は、ロボットの速度を指定した値です。指定する単位は mm/sec です。デフォルト値は、ロボットの機種によって異なります。SpeedS のデフォルト値については、各ロボットのマニピュレーターマニュアルを参照してください。コントローラーの電源オン時は、この値が常に自動的に設定されます。

下記のいずれかの場合、SpeedS 値は、初期化されます。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

ローパワーモード時には、SpeedS 値は、デフォルト値か設定値のどちらか低い方の速度が有効になります。より高い速度設定を、コマンドウィンドウから、またはプログラム中で指定すると、速度はデフォルト値に設定されます。ハイパワーモード時には、動作速度は、SpeedS 設定値になります。

より速い動作速度が必要な場合は、Power High でハイパワーモードを設定し、安全扉を開めます。安全扉が開いていると、SpeedS 値は、デフォルト値に変更されます。

参照

AccelS, Arc, Jump3, Move, Speed

SpeedS 使用例

SpeedS は、コマンドウィンドウでも、プログラム中でも、いずれでも用いることができます。下記に両方の場合の使用例を示します。

```
Function speedtst
  Integer slow, fast, i
  slow = 50
  fast = 500
  For i = 1 To 10
    SpeedS slow
    Move P0
    Move P1
    SpeedS fast
    Move P0
    Move P1
  Next i
End
```

SpeedS は、コマンドウィンドウからも設定できます。

```
> speeds 1000
> speeds 500
> speed 30      ' PTP 動作速度を設定
> go p0        ' PTP 動作で移動
> speeds 100   ' 直線速度を mm/sec で設定
> move P1      ' 直線で移動
```

SpeedS 関数

SpeedS 設定を返します。

書式

SpeedS [(パラメーター番号)]

パラメーター

パラメーター番号 返される SpeedS 値を、下記の中から整数または式で指定します。省略可能です。

1: CP 速度

2: Jump3 退避速度

3: Jump3 接近速度

戻り値

実数値を返します。単位は mm/sec です。

参照

SpeedS

SpeedS 関数使用例

```
Real savSpeeds  
  
savSpeeds = SpeedS  
  
Print "Jump3 depart speed = ", SpeedS(2)
```

Sqr 関数

平方根を返します。

書式

Sqr(数値)

パラメーター

数値 実数値または式で指定します。

戻り値

平方根を返します。

解説

平方根を返します。

起こりやすいエラー

負数演算子

数値が負数の場合、エラーが発生します。

参照

Abs, And, Atan, Atan2, Cos, Int, Mod, Not, Or, Sgn, Sin, Str\$, Tan, Val, Xor

Sqr 関数使用例

コマンドウィンドウから Sqr 関数を使用した例です。

```
>print sqr(2)
1.414214
>
```

Sqrを使用した簡単なプログラム例です。

```
Function sqrtest
  Real x
  Print "Please enter a numeric value:"
  Input x
  Print "The Square Root of ", x, " is ", Sqr(x)
Fend
```

ST 関数

指定された付加軸の座標値をポイントデータとして返します。

書式

ST(s 座標要素, t 座標要素)

パラメーター

s 座標要素	s 軸の座標を実数値で指定します。
t 座標要素	t 軸の座標を実数値で指定します。

戻り値

指定された付加軸の座標値をポイントデータとして返します。

解説

付加軸(S 軸, T 軸)を使用する場合のみ本関数は使用します。

Go ST(10,20) というような使い方をした場合、付加軸は指定の座標値へ移動しますが、ロボットは動きません。ロボットも同時に動作させるようなケースでは Go XY(60,30,-50,45) : ST(10,20) のような指定をしてください。

詳細は、ユーザズガイド「21. 付加軸」を参照してください。

参照

XY 関数

ST 関数使用例

```
P10 = ST(10, 20)
```

StartMain

バックグラウンドタスクからメインファンクションを実行します。
本命令は、上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

StartMain メインファンクション名

パラメーター

メインファンクション名 実行するメインファンクション名 (main~main63) を指定します。

解説

本命令をプログラムから実行するには、EPSON RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。

バックグラウンドタスクから Xqt 命令でタスクを実行した場合、実行されたタスクもバックグラウンドタスクになります。本命令を使用すると、バックグラウンドタスクから、メインファンクションを通常タスクとして実行することができます。

すでにメインファンクションを実行済みの場合や、通常タスクから本命令を実行した場合はエラーが発生します。



注意

■ プログラムからStartMainコマンドを実行する場合、コマンドの仕様を理解して、システムとしてメインファンクション実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

Xqt

StartMain 使用例

```
Function bgmain
:
If Sw(StartMainSwitch) = On And Sw(ErrSwitch) = Off Then
StartMain main
EndIf
:
Fend
```

Stat 関数

コントローラーの状態を返します。

書式

Stat(アドレス)

パラメーター

アドレス コントローラーの状態を示すアドレス (0~2 の整数)を指定します。

戻り値

コントローラーの状態を示す 4 バイトの値を返します。(下記表を参照してください。)

解説

Stat 命令は、下記表に示されたデータを返します。各ビットの内容を参照してください。

アドレス	Bit		ビットがオンのとき示されるコントローラーの状態
0	0-15	&H1-&H8000	タスク 1 から 16 が 実行中 (Xqt)または Halt 状態
	16	&H10000	タスク実行中
	17	&H20000	ポーズ (一時停止)状態
	18	&H40000	エラー状態
	19	&H80000	TEACH モード
	20	&H100000	非常停止状態
	21	&H200000	ローパワーモード (Power Low)
	22	&H400000	安全扉入力が開
	23	&H800000	Enable スイッチが開
	24	&H1000000	未定義
	25	&H2000000	未定義
	26	&H4000000	テストモード
	27	&H8000000	T2 モード状態
	28-31		未定義
1	0	&H1	Jump...Sense ステートメントの条件成立で目標座標上空停止の履歴。 (この履歴は次の Jump ステートメントが実行されると消去されます。)
	1	&H2	Go/Jump/Move...Till ステートメントの条件成立で動作途中停止した履歴。 (この履歴は次の Go/Jump/Move...Till ステートメントが実行されると消去されます。)
	2	&H4	未定義
	3	&H8	Trap ステートメントの条件成立で動作途中停止した履歴
	4	&H10	Motor On 状態
	5	&H20	現在は、ホーム位置
	6	&H40	ローパワー状態
	7	&H80	未定義
	8	&H100	第 4 関節の励磁中
	9	&H200	第 3 関節の励磁中
	10	&H400	第 2 関節の励磁中
	11	&H800	第 1 関節の励磁中
	12	&H1000	第 6 関節の励磁中
	13	&H2000	第 5 関節の励磁中

アドレス	Bit		ビットがオンのとき示されるコントローラーの状態
	14	&H4000	第 T 関節の励磁中
	15	&H8000	第 S 関節の励磁中
	16	&H10000	第 7 関節の励磁中
	17-31		未定義
2	0-15	&H1-&H8000	タスク 17 から 32 が 実行中(Xqt)または Halt 状態

参照

EStopOn 関数, TillOn 関数, PauseOn 関数, SafetyOn 関数

Stat 関数使用例

```
Function StatDemo

Integer rbt1_sts
rbt1_sts = RShift((Stat(0) And &H070000), 16)
Select TRUE
    Case (rbt1_sts And &H01) = 1
        Print "Tasks are running"
    Case (rbt1_sts And &H02) = 2
        Print "Pause Output is ON"
    Case (rbt1_sts And &H04) = 4
        Print "Error Output is ON"
Send
Fend
```

Str\$ 関数

数値を文字列化して返す関数です。

書式

Str\$(数値)

パラメーター

数値 文字列に変換する数値を、式または直接数値で指定します。

戻り値

数値を文字列化して返します。

解説

Str\$ は、数値 (正負とも可能)を文字列化します。

参照

Abs, Asc, Chr\$, InStr, Int, Left\$, Len, Mid\$, Mod, Right\$, Sgn, Space\$, Val

Str\$ 関数使用例

下記例は、いくつかの異なる数値を文字列化してスクリーンに表示します。

```
Function strtest
  Integer intvar
  Real realvar
  '
  intvar = -32767
  Print "intvar = ", Str$(intvar)
  '
  realvar = 567.9987
  Print "realvar = ", Str$(realvar)
  '
End
```

コマンドウィンドウから Str\$ 命令を行った結果です。

```
> Print Str$(999999999999999)
1.000000E+014

> Print Str$(25.999)
25.999
```

String

変数を文字列変数に宣言します。

書式

String 変数名\$[(配列の最大要素番号)] [,変数名\$[(配列変数の最大要素番号)]...]

パラメーター

変数名\$ 文字列型と宣言する変数の名前を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	200
バックアップ変数(Global Preserve)	400
グローバル変数とモジュール変数	10,000

解説

String は、変数の文字列型宣言のために用います。文字列型変数は、最大 255 文字までです。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

String 演算子

文字列変数には、下記の演算子を使うことができます。

- + 複数の文字列変数をマージします。文字列変数への割当ステートメントや Print 命令に使えます。

Example: name\$ = fname\$ + " " + lname\$

- = 複数の文字列変数を比較します。Case を含む 2 つの文字列が完全に一致するとき、True が返されます。

Example: If temp1\$ = "A" Then GoSub test

- <> 複数の文字列変数を比較します。2 つの文字列中、1 つ以上の文字が異なるときに True を返します。

Example: If temp1\$ <> "A" Then GoSub test

注意

変数名の最後には"\$"をつけてください

String 型の変数名は、最後に"\$"をつけてください。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, UByte, UInt32, UInt64, UShort

String 使用例

```
String password$  
String A$(10)           'String 型の 1 次元配列  
String B$(10, 10)      'String 型の 2 次元配列  
String C$(5, 5, 5)     'String 型の 3 次元配列
```

```
Print "Enter password:"  
Input password$  
If UCase$(password$) = "EPSON" Then  
    Call RunMaintenance  
Else  
    Print "Password invalid!"  
EndIf
```

Sw 関数

指定された入力ビットの状態を返す関数です。

書式

Sw(ビット番号)

パラメーター

ビット番号 I/O の入力ビットを、整数または式で指定します。

戻り値

指定された入力が入オンのとき“1”を、オフのとき“0”を返します。

解説

Sw は、I/O 入力の状態チェックを行います。Sw は、最も一般的には、I/O で動くローダー、コンベヤー、グリッパー、その他の周辺装置に接続するセンサーを、チェックするために使われます。Sw 命令でチェックされる入力状態には、“1”か“0”があります。それぞれ、デバイスがオン (1)かオフ (0)かを示します。

参照

In, InBCD, MemOn, MemOff, MemSw, Off, On OpBCD, Oport, Out, Wait

Sw 関数使用例

下記例は、入力ビット 5 をチェックして、その結果に基づいてプログラムを枝分かれさせる例です。より明確にするため、“1”の代わりに“On”が使われています。

```
Function main
  Integer i, feed5Ready
  feed5Ready = Sw(5)
  'Feeder の準備ができているか確認
  If feed5Ready = On Then
    Call mkpart1
  Else
    Print "Feeder #5 is not ready. Please reset and"
    Print "then restart program"
  EndIf
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> print sw(5)
1
>
```

SyncLock

相互排他ロックを用いて複数のタスクを同期させます。

書式

SyncLock シグナル番号 [, タイムアウト]

パラメーター

シグナル番号 受信する信号番号 (0~63 の整数)を、式または数値で指定します。
タイムアウト ロックまで待つ最長待機時間を、式または数値 (実数)で指定します。省略可能です。

解説

SyncLock は、共通のリソースを 1 度に 1 タスクだけが使うよう、同期ロックするために使います。あるタスクが共通リソースを使い終わったら、SyncUnlock を呼び出してロック解除し、他のタスクがそのリソースを使えるようにします。

1 つのタスクがロック解除することができるのは、あらかじめ自分がロックした syncID だけです。

タスクは、ロック解除するために SyncUnlock を実行しなくてははいけません。
タスクが終了された場合は、そのタスクが行ったロックは解除されます。

同じシグナル番号に対して、2 回連続で SyncLock を実行するとエラーになります。

タイムアウトパラメーターが指定された場合は、Tw 関数を使って、ロックが成功したかどうかをチェックできるようにしてください。

注意

EPSON RC+ 5.0 ではタスクが終了しても自動的にロックは解除されませんが、EPSON RC+ 6.0, RC+ 7.0 では自動的に解除されます。

参照

Signal, SyncLock, Tw, Wait, WaitPos

SyncLock 使用例

下記例では、SyncLockとSyncUnlockを使って、1度に1タスクだけが、ログファイルにメッセージを書き込めるように設定しています。

```
Function Main

    Xqt Func1
    Xqt Func2
Fend

Function Func1
    Long count
    Do
        Wait .5
        count = count + 1
        LogMsg "Msg from Func1, " + Str$(count)
    Loop
Fend

Function Func2
    Long count
    Do
        Wait .5
        count = count + 1
        LogMsg "Msg from Func2, " + Str$(count)
    Loop
Fend

Function LogMsg(msg$ As String)
    SyncLock 1
    OpenCom #1
    Print #1, msg$
    CloseCom #1
    SyncUnlock 1
Fend
```

下記は、オプションのタイムアウトを使った **SyncLock** の例です。Twを使ってロックが成功したかどうかをチェックしています。タイムアウトを使うことで、リソースをロックするのを待っている間、他のプログラムを実行することができます。

```
Function MySyncLock(syncID As Integer)
    Do
        SyncLock syncID, .5
        If Tw = 0 Then
            Exit Function
        EndIf
        If Sw(1) = On Then
            Off 1
        EndIf
    Loop
Fend
```

SyncUnlock

あらかじめ SyncLock でロックされた、シグナル番号のロックを解除します。

書式

SyncUnlock シグナル番号

パラメーター

シグナル番号 受信する信号番号 (1~63 の整数)を、式または数値で指定します。

解説

SyncUnlock は、あらかじめ SyncLock でロックされたシグナル番号のロックを解除します。
あらかじめロックされたシグナル番号でなければ、ロック解除することはできません。

参照

Signal, SyncLock, Wait, WaitPos

SyncUnlock 使用例

```
Function Main

    Xqt task
    Xqt task
    Xqt task
    Xqt task
Fend

Function task
    Do
        SyncLock 1
        Print "resource 1 is locked by task", MyTask
        Wait .5
        SyncUnlock 1
    Loop
Fend
```


SyncRobots

動作予約中のロボットの動作を開始します。

書式

SyncRobots ロボット番号 [, ロボット番号] [...]

SyncRobots All

パラメーター

ロボット番号 動作を開始するロボット番号を式または数値 (整数) で指定します。

All 動作予約中のすべてのロボットを指定します。

解説

SyncRobots は、各動作コマンドにサポートされている SYNC パラメーターを使って動作予約されたロボットの動作を開始するために使います。SyncRobots で指定されたロボットは同じタイミングで動作が開始します。通常のマルチタスクプログラムの I/O 信号のイベント待ちを使った同期よりも、タスクスイッチングの影響がないため、より正確にロボットの動作開始を同期できます。

動作予約されていないロボット番号を指定した場合はエラーになります。

参照

SyncRobots 関数

SyncRobots 使用例

下記例では、動作コマンドの SYNC パラメーターと SyncRobots を使って、2 台のロボットを同時に動作開始します。

```
Function Main
  Xqt Func1
  Xqt Func2
  Do
    Wait 0.1
    If (SyncRobots And &H03) = &H03 Then
      Exit Do
    EndIf
  Loop
  SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

SyncRobots 関数

動作予約中のロボットの状態を返す関数です。

書式

SyncRobots

戻り値

ロボットが動作予約中のとき、そのロボット番号に対応するビットを“1”、予約中でないとき“0”とする整数値を返します。

ビット 0: ロボット番号 1
 ビット 1: ロボット番号 2
 :
 ビット 15: ロボット番号 16

解説

SyncRobots 関数は、ロボット動作コマンドの SYNC パラメーターによるロボットの動作予約の状態チェックを行います。SyncRobots でチェックされる状態は、ロボット番号に対応するビットの状態で表されます。それぞれ、各ビットはロボットが動作予約中のとき (1)か予約中でないとき (0)かを示します。予約中のロボットは SyncRobots 命令で動作を開始することができます。

参照

SyncRobots

SyncRobots 関数使用例

下記例では、動作コマンドの SYNC パラメーターと SyncRobots を使って、2 台のロボットを同時に動作開始します。

```
Function Main
  Xqt Func1
  Xqt Func2
  Do
    Wait 0.1
    If (SyncRobots And &H03) = &H03 Then
      Exit Do
    EndIf
  Loop
  SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

SysConfig

システム設定パラメーターを表示します。

書式

SysConfig

結果

システム設定パラメーターを表示します。

解説

システム制御データとして現在設定されている値を表示します。ロボットの納入時やデータを変更したときは、このデータを保存してください。保存するには、EPSON RC+ の[ツール]-[メンテナンス]メニュー-[コントローラー設定バックアップ]で行います。

表示されるのは、次のデータです。(次の数値は例で、実際の数値はコントローラーの機種によって異なります。)

```
' Version:
'   Firmware 1, 0, 0, 0

' Options:
'   External Control Point
'   RC+ API

' HOUR: 414.634

' Controller:
'   Serial #: 0001

' ROBOT 1:
' Name: Mnp01
' Model: PS3-AS10
' Serial #: 0001
' Motor On Time: 32.738
'   Motor 1: Enabled, Power = 400
'   Motor 2: Enabled, Power = 400
'   Motor 3: Enabled, Power = 200
'   Motor 4: Enabled, Power = 50
'   Motor 5: Enabled, Power = 50
'   Motor 6: Enabled, Power = 50

ARCH 0, 30, 30
ARCH 1, 40, 40
ARCH 2, 50, 50
ARCH 3, 60, 60
ARCH 4, 70, 70
ARCH 5, 80, 80
ARCH 6, 90, 90
ARMSET 0, 0, 0, 0, 0, 0
HOFs 0, 0, 0, 0, 0, 0
HORDR 63, 0, 0, 0, 0, 0
RANGE -7427414, 7427414, -8738134, 2621440, -3145728, 8301227, -
5534152, 5534152, -3640889, 3640889, -6553600, 6553600
BASE 0, 0, 0, 0, 0, 0
WEIGHT 2, 0
INERTIA 0.1, 0
```

XYLIM 0, 0, 0, 0, 0, 0

```
' Extended I/O Boards:
'   1: Installed
'   2: Installed
'   3: None installed
'   4: None installed

' Fieldbus I/O Slave Board:
'   Installed
'   Type: PROFIBUS

' Fieldbus I/O Master Board:
'   None installed

' RS232C Boards:
'   1: Installed
'   2: None installed

' PG Boards:
'   1: None installed
'   2: None installed
'   3: None installed
'   4: None installed
```

SysConfig 使用例

> **SysConfig**

SysErr 関数

最新のエラーステータス、またはワーニングステータスを返します。

書式

SysErr [(情報番号)]

パラメーター

情報番号 エラーコード、またはワーニングコードのどちらを取得するかを整数値で指定します。
(省略可)
0: エラーコード (省略時)
1: ワーニングコード

戻り値

コントローラーのエラーコード、またはワーニングコードを整数値で返します。

解説

本関数は、NoEmgAbort タスク(Xqt 時に、NoEmgAbort を指定して開始された特別なタスク)とバックグラウンドタスクでのみ使用します。
コントローラーのエラーコード、ワーニングコードとは LCD に表示されているエラーコード、ワーニングコードです。
エラーやワーニングが発生していない場合は “0” を返します。

参照

ErrMsg\$, ErrorOn, Trap, Xqt

SysErr 関数使用例

下記例は、コントローラーがエラー状態になるのを監視し、エラーが発生した場合、エラー番号に応じて I/O をオンオフするプログラムです。

注意

Forced フラグ

本プログラム例では、On/Off 命令に Forced フラグを指定しています。
エラー発生中、および、非常停止中や安全扉開時に I/O 出力が変化するのでシステム設計上の注意が必要です。

エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

```
Function main

    Xqt ErrorMonitor, NoEmgAbort
    :
    :
Fend

Function ErrorMonitor
    Wait ErrorOn
    If 4000 < SysErr Then
        Print "Mortion Error = ", SysErr
        Off 10, Forced
        On 12, Forced
    Else
        Print "Other Error = ", SysErr
        Off 11, Forced
        On 13, Forced
    EndIf
Fend
```

Tab\$ 関数

指定した数だけのタブ文字列を返します。

書式

Tab\$(指定数)

パラメーター

指定数 タブの数を整数値で指定します。

戻り値

タブ文字を含む文字列を返します。

解説

指定した数だけのタブ文字列を返します。

参照

Left\$, Mid\$, Right\$, Space\$

Tab\$ 関数使用例

```
Print "X", Tab$(1), "Y"  
Print  
For i = 1 To 10  
    Print x(i), Tab$(1), y(i)  
Next i
```

Tan 関数

指定した数値のタンジェント値を返します。

書式

Tan(角度)

パラメーター

角度 角度を表わす実数値を指定します。

戻り値

指定した角度を表わす数値のタンジェント値を、数値で返します。

解説

Tan はタンジェント値を返します。パラメーターの単位はラジアンでなければなりません。
ラジアンを角度に変換するためには RadToDeg 関数を使います。

参照

Abs, Atan, Atan2, Cos, Int, Mod, Not, Sgn, Sin, Sqr, Str\$, Val

Tan 関数使用例

```
Function tantest
    Real num
    Print "Enter number in radians to calculate tangent for:"
    Input num
    Print "The tangent of ", num, "is ", Tan(num)
Fend
```

次はコマンドウィンドウから Tan 関数を使う例です。

```
> print tan(0)
0.00
> print tan(45)
1.6197751905439
>
```


TargetOK 関数

現在位置から目標座標への PTP (point to point) 動作が可能かどうかを返します。

書式

TargetOK(目標座標)

パラメーター

目標座標 動作可能かどうかを調べる目標座標をポイントデータで指定します。

戻り値

現在位置から目標座標までの PTP 動作が可能であれば“True”を、そうでない場合は“False”を返します。

解説

目標座標や姿勢に到達できるかを、実際に (ロボットを)動かす前に確認します。ただし、目標座標にたどり着くまでの軌道は考慮されていません。

参照

CurPos, FindPos, InPos, WaitPos

TargetOK 関数使用例

```
If TargetOK(P1) Then  
  Go P1  
EndIf  
  
If TargetOK(P10 /L /F) Then  
  Go P10 /L /F  
EndIf
```

TaskDone 関数

タスクの終了を確認する関数です。

書式

TaskDone (タスク識別子)

パラメーター

タスク識別子 タスク名またはタスク番号を整数値、または式で指定します。
タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。
タスク番号の指定 (整数)
通常タスク : 1~32
バックグラウンドタスク : 65~80
トラップタスク : 257~267

戻り値

タスクが終了している場合は“True”、それ以外は“False”を返します。

解説

TaskDone はタスクが終了しているか確認するために使います。

参照

TaskState, TaskWait

TaskDone 関数使用例

```
Xqt 2, conveyor  
Do  
  .  
  .  
Loop Until TaskDone (conveyor)
```

TaskInfo 関数

タスクの状態情報を返します。

書式

TaskInfo(タスク識別子, インデックス)

パラメーター

タスク識別子 タスク名またはタスク番号を整数値で指定します。
タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

通常タスク : 1~32

バックグラウンドタスク : 65~80

トラップタスク : 257~267

インデックス 検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報を整数値で返します。

解説

インデックス	解説
0	タスク番号
1	0 - バックグラウンドタスク 1 - 通常のタスクか、NoPause タスクか、NoEmgAbort タスク
2	タスクタイプ 0 - 通常のタスク Xqt で何も指定しないか、Normal を指定して開始したタスク 1 - NoPause タスク Xqt で NoPause を指定して開始したタスク 2 - NoEmgAbort タスク Xqt で NoEmgAbort を指定して開始したタスク 3 - Trap タスク 4 - バックグラウンドタスク
3	-1 - 指定したタスクは実行していない 1 - 指定したタスク実行中 2 - 指定したタスクはイベント待ち 3 - 指定したタスクは一時停止または Halt 中 4 - 指定したタスクはクイックポーズ状態 5 - 指定したタスクはエラー状態
4	イベント待機中にタイムアウト発生 (TW と同じ)
5	イベント待ち時間 (msec)
6	タスクが選択したロボット番号
7	タスクが使用中のロボット番号

参照

CtrlInfo, RobotInfo, TaskInfo\$ 関数

TaskInfo 関数使用例

```
If (TaskInfo(1, 3) <> 0 Then
    Print "Task 1 is runnning"
Else
    Print "Task 1 is not running"
EndIf
```

TaskInfo\$ 関数

タスクのテキスト情報を返します。

書式

TaskInfo\$(タスク識別子, インデックス)

パラメーター

タスク識別子 タスク名またはタスク番号を整数値で指定します。
タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

通常タスク : 1~32

バックグラウンドタスク : 65~80

トラップタスク : 257~267

インデックス 検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報を文字列で返します。

解説

TaskInfo\$で検索できる情報は次の表のとおりです。

インデックス	解説
0	タスク名
1	開始日時
2	現在実行しているファンクション名
3	ファンクションを含んでいる、プログラムの行番号

参照

CtrlInfo, RobotInfo, TaskInfo

TaskInfo\$ 関数使用例

```
Print "Task 1 started: "TaskInfo$(1, 1)
```

TaskState 関数

タスクの現在の状態を取得する関数です。

書式

TaskState(タスク識別子)

パラメーター

タスク識別子 タスク名またはタスク番号を整数値で指定します。
タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

通常タスク	: 1~32
バックグラウンドタスク	: 65~80
トラップタスク	: 257~267

戻り値

- 0: 指定タスク未実行
- 1: 指定タスク実行中
- 2: 指定タスクイベント待機中
- 3: 指定タスク一時停止中
- 4: 指定タスククイックポーズ中
- 5: 指定タスクエラー状態

解説

TaskState はタスク番号、またはタスク名で指定したタスクの現在状態を取得します。

参照

TaskDone, TaskWait

TaskState 関数使用例

```
If TaskState(conveyor) = 0 Then  
    Xqt 2, conveyor  
EndIf
```

TaskWait

指定のタスクの終了を待ちます。

書式

TaskWait (タスク識別子)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。

タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

通常タスク : 1~32

バックグラウンドタスク : 65~80

トラップタスク : 257~267

参照

TaskDone, TaskState

TaskWait 使用例

```
Xqt 2, conveyor  
TaskWait conveyor
```

TC

トルク制御モードの設定および現在のモードを表示します。

書式

- (1) TC { On | Off }
- (2) TC

パラメーター

On | Off On : トルク制御モードを有効にします。
 Off : トルク制御モードを無効にします。

戻り値

パラメーター省略時、現在のトルク制御モードを表示します。

解説

TC On/Off でトルク制御モードを有効、無効にします。
 トルク制御モードは、モーター出力の制限値を設定することで一定の力を発生させるためのモードで、ハンドを対象物に一定の力で押しつける、ハンドと対象物の接触を保ちながら対象物の動きに合わせてな
 らい動作をさせる場合などに利用します。
 トルク制御を有効にする前に、**TCLim** でトルク制限値を設定してください。
 トルク制御モード中も、動作コマンド実行時には、目標位置に位置決めするようにロボットが動作します。
 ロボットが対象物に接触して、モーター出力がトルク制限値に達したところで、ロボットは停止し、一定トルクを保持します。

下記のいずれかの場合、トルク制御モードを無効にします。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

参照

TCLim , TCSpeed

TC 使用例 1

```
Speed 5
Go ApproachPoint

'Z 軸トルク制限値を 20%に設定します。
TCLim -1, -1, 20, -1

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```


注意

Safety 基板を搭載したコントローラーで位置異常が検出される場合

「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。

大きく離れてしまうと、Safety 基板が故障と判断し、「エラーNo.9801 Safety 基板が位置異常を検出した」エラーが出ます。(Safety 基板を搭載したコントローラー)

ロボットの現在位置は RealPos 関数で取得できます。

ロボットの現在の動作目標位置は CurPos 関数で取得できます。

スカラの場合、RealPos の差と CurPos の差は J1:10 度, J2:10 度, J3:40mm, J4:90 度程度以下で使用することを奨励します。

エラーが発生する値は機種ごとに異なります。エラーにならない範囲で機種ごとに大きく設定が可能です。

TCSpeed で速度制限すると、対象物に接触する前に RealPos と CurPos に差が生じます。

TCSpeed を利用しないか、利用する場合は Speed と同じ値にしてください。

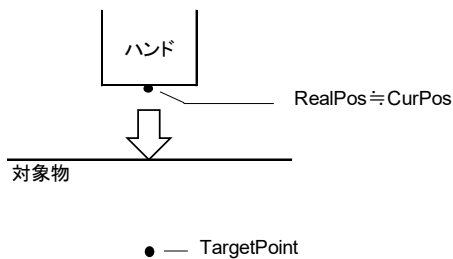


図 1. 対象物に接触していない場合

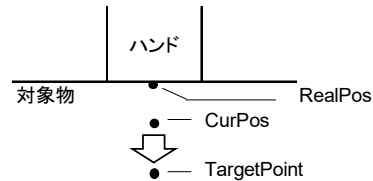


図 2. 対象物に接触している場合

1. 対象部に接触していない場合

ロボットの現在位置 (RealPos) と、ロボットの現在の動作目標位置 (CurPos) は近い位置です。

2. 対象部に接触している場合

ロボットの現在位置 (RealPos) は、対象物に接触した位置を維持します。

ロボットの現在の動作目標位置 (CurPos) は、TargetPoint に向かって進行します。

この距離が一定以上開くとエラーNo.9801 が発生します。

Safety 基板の位置異常を発生させずに利用するサンプルプログラムを以下に示します。

Till コマンドを使い、「ロボットの現在位置」と、「ロボットの現在の動作目標位置」の差が一定以上で次の処理に進むことによりエラーを回避できます。

差をなるべく小さい値にすることで、早期に次の処理に移るため、タクトタイム向上につながります。

TC 使用例 2

```
Speed 5
Go ApproachPoint
```

'Z 軸トルク制限値を 20%に設定します。

```
TCLim -1, -1, 20, -1
```

Xqt posDiffChk(10.0) '位置差分チェックタスク起動。J3 軸 10.0mm 以上の差で、フラグを立てる

```
TC On
Go TargetPoint Till MemSw(0)
Wait 3
Go ApproachPoint
TC Off
```

```
Function posDiffChk(Zth As Double)
    Do
        If (Abs(CZ(RealPos) - CZ(CurPos)) > Zth) Then
            '「ロボットの現在位置」と、「ロボットの現在の目標位置」の差が閾値を超過したか?
            MemOn (0) ' 位置偏差大フラグ ON
        Else
            MemOff (0) ' 位置偏差大フラグ クリア
        EndIf
        Wait 0.01
    Loop
Fend
```

TCLim

トルク制御モードのための各関節のトルク制限値を設定します。

書式

TCLim [第 1 関節トルク制限値, 第 2 関節トルク制限値, 第 3 関節トルク制限値,
第 4 関節トルク制限値 [, 第 5 関節トルク制限値] [, 第 6 関節トルク制限値]
[, 第 7 関節トルク制限値] [, 第 8 関節トルク制限値] [, 第 9 関節トルク制限値]]

パラメーター

- 第 1 関節トルク制限値** 瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 2 関節トルク制限値** 瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 3 関節トルク制限値** 瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 4 関節トルク制限値** 瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 5 関節トルク制限値** 省略可。瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 6 関節トルク制限値** 省略可。瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 7 関節トルク制限値** 省略可。瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 8 関節トルク制限値** 省略可。S 軸の瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。
- 第 9 関節トルク制限値** 省略可。T 軸の瞬時最大トルクに対する割合 (1~100 の整数、単位: %)を式または数値で指定します。-1 のときトルク制限値を無効とし、通常的位置制御モードとなります。

戻り値

パラメーター省略時、現在のトルク制限値を表示します。

解説

トルク制限値の設定は TC On のときに有効となります。

設定値が低すぎるとロボットが動作せず、目標位置に到達する前に動作コマンドが終了します。

下記のいずれかの場合、TCLim 設定値は、初期化されます。

コントローラー起動時 Motor On 実行 SFree, SLock, Brake 実行 Reset, Reset Error 実行 停止ボタン, Quit All 実行などによるタスクの終了

参照

TC, TCLim 関数, TCSpeed

TCLim 使用例

```
Speed 5  
Go ApproachPoint
```

```
・ Z 軸トルク制限値を 20%に設定します。  
TCLim -1, -1, 20, -1
```

```
TC On  
Go TargetPoint  
Wait 3  
Go ApproachPoint  
TC Off
```

注意

Safety 基板を搭載したコントローラーで位置異常が検出される場合

「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。

大きく離れてしまうと、Safety 基板が故障と判断し、「エラーNo.9801 Safety 基板が位置異常を検出した」エラーが出ます。(Safety 基板を搭載したコントローラー)

詳細は、TC コマンドの説明を参照してください。

TCLim 関数

指定した関節のトルク制限値を返します。

書式

TCLim(関節番号)

パラメーター

関節番号 トルク制限値を取得したい関節の関節番号を式または数値で指定します。
付加軸の S 軸は 8、T 軸は 9 になります。

戻り値

現在のトルク制限値をあらわす 1~100 の整数を返します。トルク制限値が無効の場合、-1 を返します。

参照

TC, TCLim, TCSpeed

TCLim 関数使用例

```
Print "現在の Z 軸トルク制限値:", TCLim(3)
```

TCPSpeed 関数

算出された現在のツール先端 (TCP)速度を返します。

書式

TCPSpeed

戻り値

算出された現在のツール先端速度を実数で返します。(単位: mm/秒)

解説

CP 動作コマンドを実行したときの、算出されたツール先端速度を mm/秒で返します。

CP 動作コマンドとは、Move, TMove, Arc, Arc3, CVMove, および Jump3CP です。この速度は実際の速度と異なります。この速度は、ファンクションを呼び出したときにシステムが動作計画したツール先端速度です。

モーターの追従遅れが考慮されません。

ロボットが PTP 動作コマンドを実行している場合は、“0”を返します。

付加軸を使用している場合でも、ロボットの移動速度のみを返します。

例えば、付加軸を走行軸に使用するケースでも付加軸移動速度を考慮しません。

参照

AccelS, CurPos, InPos, SpeedS

TCPSpeed 関数使用例

```
Function MoveTest
  AccelS 4000, 4000
  SpeedS 200
  Xqt ShowTCPSpeed
Do
  Move P1
  Move P2
Loop
Fend

Function ShowTCPSpeed
Do
  Print "Current TCP speed is: ", TCPSpeed
  Wait .1
Loop
Fend
```

TCSpeed

トルク制御中の速度制限値を設定します。

書式

TCSpeed [速度]

パラメーター

速度 最大速度に対する割合 (1~100 の整数、単位: %)を式または数値で指定します。

解説

トルク制御中は、Speed コマンドなどの速度設定に関わらず、TCSpeed で設定された速度に制限されます。

トルク制御中に制限値以上の速度を検出するとエラーが発生します。

下記のいずれかの場合、TCSpeed 設定値は、100%に初期化されます。

コントローラー起動時
 Motor On 実行
 SFree, SLock, Brake 実行
 Reset, Reset Error 実行
 停止ボタン, Quit All 実行などによるタスクの終了

参照

TC, TCLim, TCSpeed 関数

TCSpeed 使用例

```
Speed 5
Go ApproachPoint
```

'Z 軸トルク制限値を 20%に設定します。

```
TCLim -1, -1, 20, -1
```

'トルク制御中の速度を Speed 設定と同じ 5%に設定します。

```
TcSpeed 5
```

```
TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

注意

Safety 基板を搭載したコントローラーで位置異常が検出される場合

「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。

TCSpeed で速度制限すると、「ロボットの現在位置」と「ロボットの現在の目標位置」に差が生じます。

TCSpeed を利用しないか、利用する場合は Speed と同じ値にしてください。

大きく離れてしまうと、Safety 基板が故障と判断し、「エラーNo.9801 Safety 基板が位置異常を検出した」エラーが出ます。(Safety 基板を搭載したコントローラー)

詳細は、TC コマンドの説明を参照してください。

TCSpeed 関数

トルク制御中の速度制限値を返します。

書式

TCSpeed

戻り値

現在の速度制限値を表す 1~100 の整数を返します。

参照

TC, TCSpeed, TCLim

TCSpeed 使用例

```
Integer var  
var = TCSpeed
```

TeachOn 関数

ティーチモードの状態を返します。

書式

TeachOn

戻り値

ティーチモードのときは“True”、それ以外のときは“False”を返します。

解説

本関数は、バックグラウンドスクでのみ使用します。

参照

ErrorOn, EstopOn, SafetyOn, Xqt

TeachOn 関数使用例

下記例は、コントローラーがティーチモードになるのを監視し、ティーチモードに移行した場合 I/O をオンオフするプログラムです。

```
Function BGMain
  Do
    Wait 0.1
    If TeachOn = True Then
      On teachBit
    Else
      Off teachBit
    EndIf
    If SafetyOn = True Then
      On safetyBit
    Else
      Off safetyBit
    EndIf
    If PauseOn = True Then
      On PauseBit
    Else
      Off PauseBit
    EndIf
  Loop
Fend
```

TGo

現在のツール座標系での、オフセット PTP 動作を実行します。

書式

TGo 目標座標 [CP] [Till | Find] [!並列処理!] [SYNC]

パラメーター

目標座標	ポイントデータを用いて動作の目標位置を指定します。
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
! 並列処理!	動作中に、I/O その他のコマンドを実行するよう、並列処理ステートメントを付加することができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

現在のツール座標系での、オフセット PTP 動作を実行します。

ポイントデータで与えられた姿勢フラグは無視され、現在の姿勢フラグが維持されます。ただし、垂直 6 軸型ロボット (N シリーズを含む) では、関節移動量が小さくなるよう姿勢フラグを自動的に変更します。

Till 修飾子を使うことで、Till 条件成立時にロボットを動作途中で減速、停止して、TGo 動作を完了することができます。

Find 修飾子を使って、動作中に Find 条件が真になったときに、ポイントを FindPos に保存します。

!並列処理!を使って、動作と並列に他の処理を実行することができます。

CP パラメーターをつけた場合には、動作の減速開始時に次の動作命令の加速を重ね合わせることが可能です。この場合には、目標座標には位置決めされません。

参照

Accel, CP, Find, !並列処理!, P# = ポイント指定, Speed, Till, TMove, Tool

TGo 使用例

```
> TGo XY(100, 0, 0, 0)  ' (ツール座標系で) X 方向に 100 mm 移動  
Function TGoTest
```

```
Speed 50  
Accel 50, 50  
Power High
```

```
Tool 0  
P1 = XY(300, 300, -20, 0)  
P2 = XY(300, 300, -20, 0) /L
```

```
Go P1  
Print Here  
TGo XY(0, 0, -30, 0)  
Print Here
```

```
Go P2  
Print Here  
TGo XY(0, 0, -30, 0)  
Print Here
```

```
Fend
```

```
[出力結果]
```

```
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /R /0  
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /R /0  
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0  
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /L /0
```

Till

Jump, Go, Move, その他の動作命令で Till 指定したときに、動作途中で停止し、処理を終了させる条件の設定と表示を行います。

書式

Till [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子 (=, <, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

関数 : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr
GetRobotInsideBox, GetRobotInsidePlane, Force, AIO_In, AIO_InW,
AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

変数 : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数、グローバル変数、モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

演算子 : And, Or, Xor

<例> Till Sw(5) = On

Till Sw(5) = On And Till(6) = Off

解説

Till ステートメントは、単独で記述するか、動作命令ステートメントの修飾子として記述してください。

Till 条件式には、上記の関数を 1 つ以上含んでいなくてはなりません。

Till 条件式に変数が含まれたときは、その値は、Till 条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。複数の Till ステートメントを記述することもできます。その場合、最後に実行された Till 条件が有効となります。

パラメーターを省略すると、現在の Till 設定が表示されます。

注意

電源オン時の Till 設定

電源オン時の Till 条件の初期設定は、Till Sw(0) = On です。入力ビット番号 0 がオンのとき、減速停止するように設定されています。

Till 条件成立をチェックする Stat 関数と TillOn 関数

Till 修飾子を用いた動作コマンドを実行した後で、Till 条件が成立したかどうかを、Stat 関数または TillOn 関数を用いてチェックすることができます。

イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort) です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が 0.01 秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1 システム内で使用できる変数待ちの個数は最大 64 個です (Wait などのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。

- 変数待ちを行う変数を Byref で参照渡しするとエラーになります。
 - イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。
-

参照

Find, Go, In, InW, Jump, MemIn, MemSw, Move, Stat, Sw, TillOn, SF_GetStatus

Till 使用例

下記は、プログラム中で Till 命令を用いた例です。

Till Sw(1) = Off	'Till 条件を設定 (入力ビット 1 オフ)
Go P1 Till	'前の行の条件を満たす場合は停止
Till Sw(1) = On And Sw(\$1) = On	'新しい Till 条件を設定
Move P2 Till	'前の行の条件を満たす場合は停止
Move P5 Till Sw(10) = On	'この行の条件を満たす場合は停止

TillOn 関数

Till の状態を返します。

書式

TillOn

戻り値

Till を使った直前の動作コマンドで、Till の条件が成立したら **True** を返します。

解説

Till の条件が成立したら **True** を返します。

TillOn は下記と同様です。

```
((Stat(1) And 2) <> 0)
```

参照

EStopOn, SafetyOn, Sense, Stat, Till

TillOn 関数使用例

```
Go P0 Till Sw(1) = On
If TillOn Then
    Print "Till condition occurred during move to P0"
EndIf
```

Time

時間の表示をします。

書式

Time

解説

現在の時間を 24 時間表記で表示します。

参照

Date, Time\$

Time 使用例

コマンドウィンドウでの実行例

```
> Time  
10:15:32
```


Time 関数

コントローラーの積算通電時間を返します。

書式

Time(単位指定)

パラメーター

単位指定 0~2 の整数値を指定します。0~2 でコントローラーの積算通電時間の単位を示します。

0: 時
1: 分
2: 秒

解説

コントローラーの積算通電時間を整数値で返します。

参照

Hour

Time 関数使用例

次の例はコマンドウィンドウから実行します。

```
Function main
  Integer h, m, s

  h = Time(0)    '通電時間を時間で返す
  m = Time(1)    '通電時間を分で返す
  s = Time(2)    '通電時間を秒で返す
  Print "This controller has been used:"
  Print h, "hours, ",
  Print m, "minutes, ",
  Print s, "seconds"
Fend
```

Time\$ 関数

現在のシステム時刻を返します。

書式

Time\$

戻り値

時間を 24 時間表記の文字列で返します。

書式は hh:mm:ss [時間:分:秒] です。

参照

Date, Date\$, Time

Time\$ 関数使用例

```
Print "The current time is: ", Time$
```

TLClr

ツール座標系の設定をクリアします。

書式

TLClr ツール番号

パラメーター

ツール番号 クリアするツールを、整数または式で指定します。
(ツール 0 はデフォルトなので、クリアできません。)

解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TLSet

TLClr使用例

TLClr 1

TLDef 関数

ツールの設定状態を返します。

書式

TLDef(ツール番号)

パラメーター

ツール番号 状態を返すツールを、整数または式で指定します。

戻り値

指定したツールが設定されている場合“True”、設定されていない場合“False”を返します。

参照

Arm, ArmClr, ArmSet, ECPSet, Local, LocalClr, Tool, TlClr, TlSet

TLDef 関数使用例

```
Function DisplayToolDef(toolNum As Integer)

    If TlDef(toolNum) = False Then
        Print "Tool ", toolNum, "is not defined"
    Else
        Print "Tool ", toolNum, ": ",
        Print TlSet(toolNum)
    EndIf
End
```

TLSet

ツール座標系を設定、表示します。

書式

- (1) TLSet ツール座標系番号, ツール設定データ
- (2) TLSet ツール座標系番号
- (3) TLSet

パラメーター

- ツール座標系番号** 設定するツールを 1~15 の整数値で指定します。(Tool 0 はデフォルトツールで変更することはできません。)
- ツール設定データ** 設定するツール座標系の原点および向きを、P 番号, P(式), ポイントラベル, ポイント式のいずれかで指定します。

結果

パラメーターをすべて省略すると、すべての TLSet 設定を表示します。
 ツール番号だけを指定すると、指定の TLSet 設定を表示します。

解説

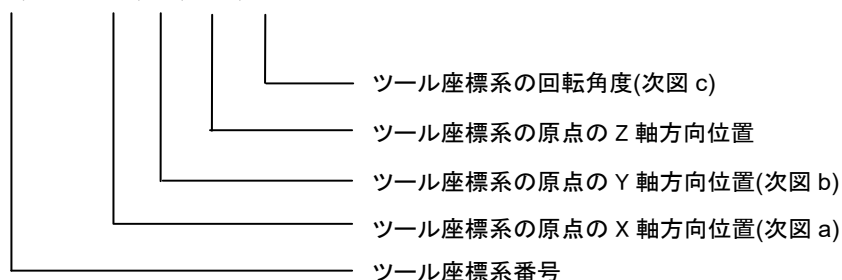
Tool 0 座標系 (ハンド座標系) に対する相対原点位置と相対回転角度を指定して、ツール座標系 Tool 1, Tool 2, Tool 3 を定義します。

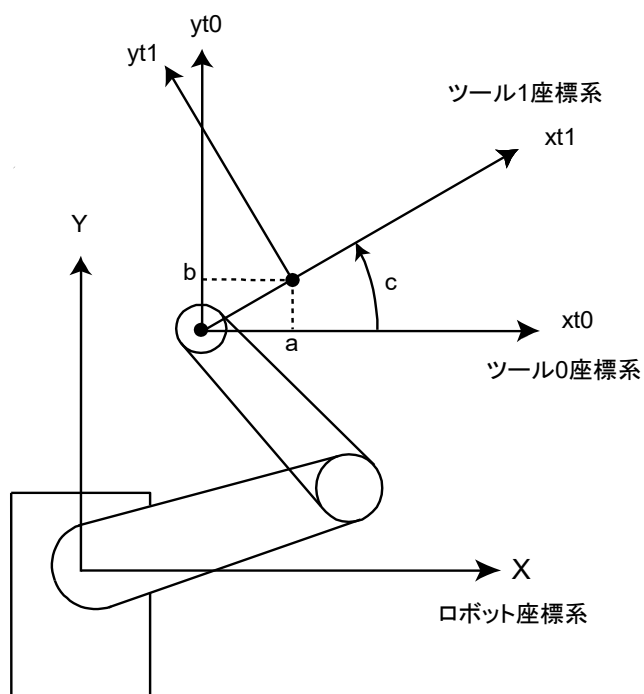
```
TLSet 1, XY(50,100,-20,30)
```

```
TLSet 2, P10 +X(20)
```

上記の場合、座標値 P10 が参照され、X 値に 20 を加えています。アーム属性とローカル座標系番号は無視されます。

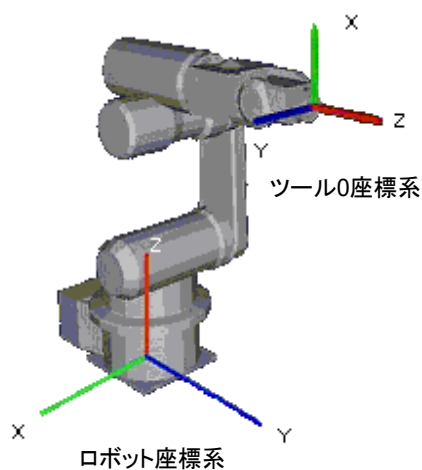
```
TLSET 1, XY(100,60,-20,30)
```





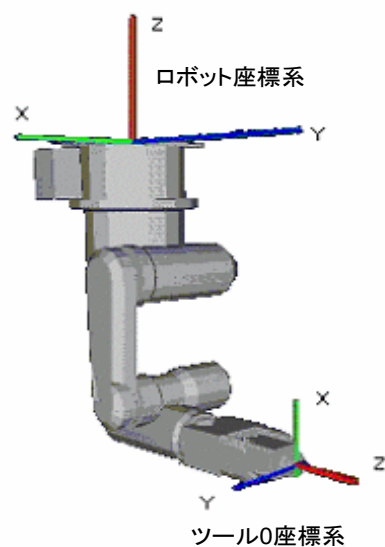
垂直 6 軸型ロボットでの TLSet の使用

垂直 6 軸型ロボットでは、すべての関節を 0 度の位置にしたとき、第 6 関節のフランジ面中心を原点とし、垂直上方向に X 軸、ロボット座標系 X 軸方向に Y 軸、第 6 関節フランジ面に対して鉛直方向に Z 軸を持つ座標系がツール 0 座標系です。(下図参照)

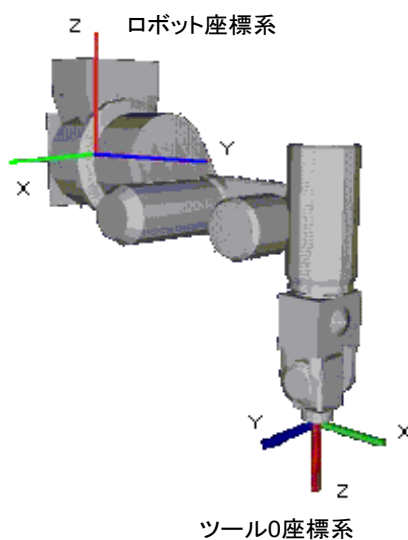


垂直 6 軸型ロボットの設置方法により、ツール 0 座標系の定義が異なります。

天吊設置

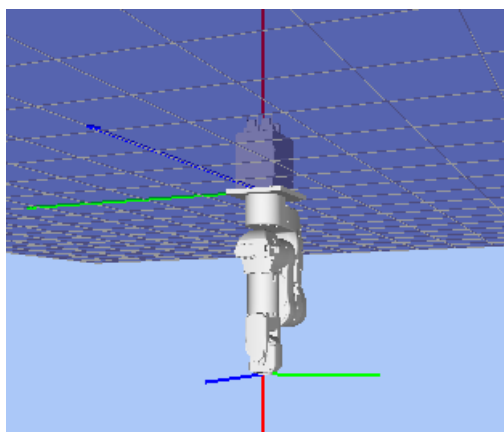


壁掛け設置



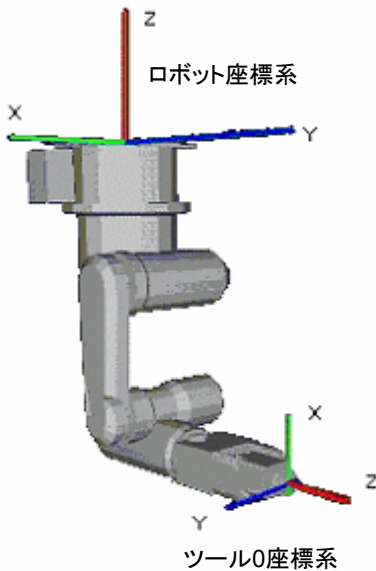
N シリーズでの TLSet の使用

N シリーズでは、すべての関節を 0 度の位置にしたとき、ロボット座標系-X 軸方向に X 軸、ロボット座標系 Y 軸方向に Y 軸、ロボット座標系-Z 軸方向に Z 軸を持つ座標系がツール 0 座標系です。(下図参照)

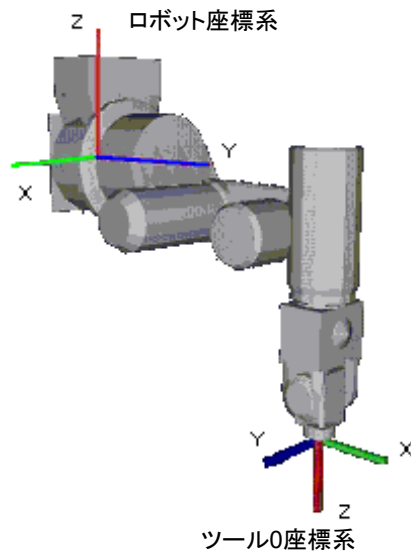


Nシリーズの設置方法により、ツール 0 座標系の定義が異なります。

天吊設置



壁掛け設置



解説

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

TLSet 値は保持されます

TLSet 値は保持されています。ツール定義をクリアするには、TLClrを使用します。

参照

Tool, Arm, ArmSet, TLSet 関数, TLClr

TLSet 使用例

下記は、ツール定義をした場合の動作と、ツール定義をしない場合の動作の違いを理解するために、コマンドウィンドウから操作を行った例です。

> TLSet 1, XY(100, 0, 0, 0)	'Tool 1 (ハンドの座標系から X 方向に 100 mm)を 'ツール座標系で定義
> Tool 1	'TLSet で定義した Tool 1 を選択
> TGo P1	'Tool 1 の目標座標を P1 に設定
> Tool 0	'今後の動作でツールを使用しないように設定
> Go P1	'U 関節の中央を P1 に設定

TLSet 関数

設定されたツール座標系データを返します。

書式

TLSet(ツール座標系番号)

パラメーター

ツール座標系番号 ツール番号を整数値で指定します。

戻り値

ツール座標系データを返します。

参照

TLSet

TLSet 関数使用例

```
P1 = TLSet(1)
```

TMOut

Wait 命令実行時のタイムアウトエラー (エラー2280)発生までの時間を設定します。

書式

TMOut 秒

パラメーター

秒 タイムアウト時間を整数値で指定します。範囲は 0~2147483 です。(単位: 秒)

解説

TMOut は Wait 命令実行時のタイムアウトエラー (エラー2280)発生までの時間を設定します。タイムアウト時間を 0 秒と設定した場合、タイムアウトは働かず、Wait 命令は指定した条件が成立するまで無期限に待機します。

TMOut のデフォルト値は “0”です。

参照

In, MemSw, OnErr, Sw, TW, Wait

TMOut 使用例

```
TMOut 5
Wait MemSw(0) = On
```

TMove

現在のツール座標系で、オフセット直線補間動作を実行します。

書式

TMove 目標座標 [ROT] [CP] [Till | Find] [! 並列処理!] [SYNC]

パラメーター

目標座標	ポイントデータを用いて動作の目標位置を指定します。
ROT	ツール姿勢変化を優先させ、動作の速度、加減速度を決めます。省略可能です。
CP	パスモーションを指定します。省略可能です。
Till Find	Till または Find 式を記述します。省略可能です。 Till Find Till Sw(式) = {On Off} Find Sw(式) = {On Off}
! 並列処理!	動作中に I/O その他のコマンドを実行するために、並列処理ステートメントを付け加えることができます。省略可能です。
SYNC	動作コマンドを予約します。SyncRobots による動作開始まで、ロボットは動作しません。

解説

現在のツール座標系で、オフセット直線補間動作を実行します。

ポイントデータで与えられた姿勢フラグは無視され、現在の姿勢フラグが維持されます。ただし、垂直 6 軸型ロボット (N シリーズを含む) では姿勢フラグを関節移動量が小さくなるよう自動的に変更します。これは、Move 命令にて LJM 修飾パラメーターを指定した場合と同等です。このため、180 度以上の姿勢変化を行わせたい場合、複数回に分けて実行してください。

TMove の速度と加減速度はそれぞれ SpeedS と AccelS の設定値を使用します。速度と加減速度の関係については、“注意”の“TMove を CP とともに用いる”を参照してください。ただし、ROT 修飾パラメーターを使っているときの速度と加減速度はそれぞれ SpeedR と AccelR の設定値を使用します。その場合、SpeedS と AccelS の設定値は無効となります。

通常、移動距離が“0”で、姿勢関節のみが動作するような場合にはエラーとなります。ROT 修飾パラメーターを付けてツール姿勢変化の加減速を優先することでエラーなく動作が可能になります。ROT 修飾パラメーターを付けたとき、姿勢変化がなく、移動距離が“0”でない場合には、エラーとなります。

また、移動距離に対してツール姿勢変化速度が大きすぎる場合や、指定された回転速度がマニピュレーターの限度を越えている場合もエラーとなります。その場合は指定速度を落とすか、または ROT 修飾パラメーターを付けて姿勢変化の加減速度を優先するようにしてください。

Till 修飾子を用いることで、Till 条件成立時に、動作途中でロボットを減速停止させて、TMove を完了することができます。

Find 修飾子を用いることで、動作中に Find 条件の値が真(True)になったとき、ポイントデータを FindPos に保存します。

!並列処理!を使って、動作と並列に他の処理を実行することができます。

注意**TMove を CP とともに用いる**

CP パラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。CP 指定なしの TMove 命令では、アームは必ず減速して、指定された目標座標に停止します。

参照

AccelS, CP, Find, ! 並列処理 !, P# = ポイント指定, SpeedS, TGo, Till, Tool

TMove 使用例

```
> TMove XY(100, 0, 0, 0) '(ツール座標系で)X 方向に 100 mm 移動
```

```
Function TMoveTest
```

```
Speed 50
```

```
Accel 50, 50
```

```
SpeedS 100
```

```
AccelS 1000, 1000
```

```
Power High
```

```
Tool 0
```

```
P1 = XY(300, 300, -20, 0)
```

```
P2 = XY(300, 300, -20, 0) /L
```

```
Go P1
```

```
Print Here
```

```
TMove XY(0, 0, -30, 0)
```

```
Print Here
```

```
Go P2
```

```
Print Here
```

```
TMove XY(0, 0, -30, 0)
```

```
Print Here
```

```
Fend
```

```
[出力結果]
```

```
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /R /0
```

```
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /R /0
```

```
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0
```

```
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /L /0
```

Tmr 関数

Tmr 関数は、タイマーがスタートしてからの経過時間を、単位秒で返します。

書式

Tmr(タイマー番号)

パラメーター

タイマー番号 64 のタイマーのうち、どのタイマーをチェックするか、整数 (0~63) を式または数値で指定します。

戻り値

指定したタイマーの経過時間を実数値 (秒単位) で返します。タイマーの範囲は、0~約 1.7E+31 です。タイマ解像度は、0.001 秒です。

解説

指定したタイマーがスタートしてからの経過時間を返します。ElapsedTime 関数と異なり、プログラムが一時停止状態となっている時間も経過時間としてカウントします。

タイマーは TmReset でリセットすることができます。

```
Real overhead
```

```
TmReset 0
overHead = Tmr(0)
```

参照

ElapsedTime 関数, TmReset

Tmr 関数使用例

```
TmReset 0                    'タイマー0をリセット
For i = 1 To 10            '10回実行
    GoSub Cycle
Next
Print Tmr(0) / 10    'サイクルタイムを計算して表示
```

TmrReset

Tmr 関数で使用するタイマーをリセットします。

書式

TmrReset タイマー番号

パラメーター

タイマー番号 64 個のタイマーのうち、リセットするタイマーの番号を整数 (0~63) で指定します。

解説

タイマー番号で指定したタイマーをリセットし、スタートします。

Tmr 関数は、指定したタイマーの経過時間を取得します。

参照

Tmr

TmrReset 使用例

```
TmrReset 0            'タイマー0をリセット
For i = 1 To 10        '10回実行
    GoSub CYL
Next
Print Tmr(0)/10        'サイクルタイムを計算して表示
```

Toff

LCD に実行行の表示をオフします。

書式

Toff

解説

Toff を実行するとタスクの実行行が LCD に表示されません。

注意

使用可能なコントローラーについて

RC90/T/VT シリーズでは使用できません。

参照

Ton

Toff 使用例

下記は、コマンドウィンドウからテストする例です。ツール設定時と未設定時の動作の違いを理解することができます。

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

Ton

LCD に実行行を表示するタスクを指定します。

書式

Ton タスク識別子
Ton

パラメーター

タスク識別子 タスク名またはタスク番号を整数値、または式で指定します。
タスク名は、Xqt ステートメントで使われたファンクション名か、Run ウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。
タスク番号の指定 (整数)
通常タスク : 1~32

注意

使用可能なコントローラーについて

RC90/T/VT シリーズでは使用できません。

解説

初期状態では、タスク番号1の実行行が表示されます。
Tonを使用すると指定したタスクの実行行をLCDに表示することができます。
タスク識別子を省略した場合、Tonを実行したタスクの実行行がLCDに表示されます。

参照

Toff

Ton 使用例

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```


Tool

ツールの選択、または選択されているツール番号の表示。

書式

- (1) Tool ツール番号
- (2) Tool

パラメーター

ツール番号 次にくる動作命令で、16 個あるツール (整数値 0~15)のうち、どれを使うか指定します。省略可能です。

結果

パラメーターを省略すると、現在設定されているツール番号を表示します。

解説

Tool は、ツール番号で指定されたツールを選択します。ツール番号“0”のときは、選択ツールはなく、すべての動作は、先端関節 (回転関節)中心に対して行われます。しかし、ツール番号“1”, “2”, “3”などが選択されたときは、ツール設定されたツールの先端に対して動作が行われます。

注意

電源オフ時のツール選択への影響

電源をオフしても、選択されたツール座標系は変更されません。

コンパクトフラッシュの寿命

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

参照

TGo, TLSet, TMove

Tool 使用例

下記は、コマンドウィンドウからテストする例です。ツール設定時と未設定時の動作の違いを理解することができます。

>tlset 1, 100, 0, 0, 0	'Tool 1 (ハンドの座標系から X 方向に 100mm)を 'ツール座標系で定義
>tool 1	'TLSet で定義した Tool 1 を選択
>tgo p1	'Tool 1 の目標座標を P1 に設定
>tool 0	'今後の動作でツールを使用しないように設定
>go p1	'U 関節の中央を P1 に設定

Tool 関数

現在設定されているツール番号を返します。

書式

Tool

戻り値

ツール番号を整数値で返します。

参照

Tool

Tool 関数使用例

```
Integer savTool
```

```
savTool = Tool
```

```
Tool 2
```

```
Go P1
```

```
Tool savTool
```

Trap (ユーザー定義トリガー)

割込み、および割込み発生時の処理の定義を行います。

Trap 命令を使用すると、イベント発生により、ラベルにジャンプする、または、ファンクションを呼び出すことができます。Trap 命令には 2 つのタイプがあります。

- ユーザーの定義する入力状態をトリガーにする 4 つの Trap
- システム状態をトリガーにする 7 つの Trap です。

本項目ではユーザー定義トリガーの Trap を説明します。

書式

Trap トラップ番号, イベント条件式 GoTo ラベル
 Trap トラップ番号, イベント条件式 Call ファンクション名
 Trap トラップ番号, イベント条件式 Xqt ファンクション名
 Trap トラップ番号

パラメーター

トラップ番号 トラップ番号 (1~4 の整数) を、式または直接数値で指定します。(SPEL+では同時に有効な Trap を 4 つまでサポートします。)

イベント条件式 トリガーとする入力状態を指定します。
 [イベント] 比較演算子(=, <, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

関数 : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr
 GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out,
 AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

変数 : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ
 変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

演算子: And, Or, Xor

<例> Trap 1, Sw(5) = On Call, TrapFunc

 Trap 1, Sw(5) = On And Till(6) = Off, Call TrapFunc

ラベル Trap 条件が成立した場合に、プログラムの実行を移行させる先のラベルです。

ファンクション名 Trap 条件が成立したとき Call 又は Xqt されるファンクションです。
 引数のあるファンクションを指定することはできません。

解説

Trap は、条件成立時に GoTo, Call, Xqt など指定された割込み処理を実行します。

Trap 条件式には、上記の関数を 1 つ以上含んでいなくてはなりません。

Trap 条件式に変数が含まれたときは、その値は Trap 条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。

いったん割込み処理が実行されると、その Trap 設定はクリアされます。同じ割込み処理を行うためには、再度 Trap 命令を実行しなくてはなりません。

Trap 設定をキャンセルするためには、トラップ番号パラメーターだけを指定して Trap 命令を実行します。

(例) “Trap 3” は、Trap #3 をキャンセルします。

また、Trap Goto は宣言したファンクションから抜けると自動的にキャンセルされます。

Trap Call は宣言したタスクが終了するとキャンセルされます。

Trap Xqt はすべてのタスクが終了するまでキャンセルされません。

GoTo を指定したとき

Trap を設定したタスクで、実行中のコマンドは下記のように処理され、制御は、指定先のラベルに移行します。

- アーム動作を直ちに一時停止(クイックポーズ)します。
- Wait や Input コマンドによる待機状態を中断します。
- その他のコマンドはすべて、制御が移る前に完了します。

Call を指定したとき

GoTo と同じ処理を実行した後、制御は、指定先のファンクションに移行します。

ファンクションが終了すると、プログラムは割込み発生時の次のステートメントに戻ります。

トラップ処理のファンクションには Call を使うことはできません。

また、トラップ処理のファンクション中でエラーが発生した場合、OnErr によるエラーハンドリングは無効となり、必ずエラーが発生します。

Xqt が指定されたとき

プログラム制御は指定されたファンクションを割込み処理専用のタスクとして生成します。この場合、Trap コマンドを実行するタスクはそのまま継続されます。

割込み処理タスクから Xqt でさらにタスクを実行することはできません。

注意

EPSON RC+ 4.x ユーザーの方へ

EPSON RC+ 4.x までの Trap Call の機能は、EPSON RC+ 7.0 では Trap Xqt に置き換えられています。

EPSON RC+ 4.x までの Trap GoSub の機能は、EPSON RC+ 7.0 では削除されました。Trap Call を使うことにより代用してください。

イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort) です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が 0.01 秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1 システム内で使用できる変数待ちの個数は最大 64 個です (Wait などのイベント条件式で使用された変数待ちも含みます)。最大数を越えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数を Byref で参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数が含まれたときは、その値は Trap 条件設定時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。

参照

Call, GoTo, Xqt, SF_GetStatus

Trap 使用例

<例 1> ユーザー定義のエラー処理

Sw(0) Input は、ユーザー定義のエラー入力とします。

```
Function Main
    Trap 1, Sw(0) = On GoTo EHandle    'Trap を定義
    .
    .
    .
EHandle:
    On 31                                'シグナルタワー点灯
```

```

    OpenCom #1
    Print #1, "Error is issued"
    CloseCom #1
Fend

```

<例 2> マルチタスク的な使用

```

Function Main
    Trap 2, MemSw(0) = On Or MemSw(1) = On Call Feeder
    .
    .
    .
Fend
.

```

```

Function Feeder
    Select TRUE
        Case MemSw(0) = On
            MemOff 0
            On 2
        Case MemSw(1) = On
            MemOff 1
            On 3
    Send

    '次のサイクルの Trap を再設定
    Trap 2, MemSw(0) = On Or MemSw(1) = On Call Feeder
Fend

```

<例 3> グローバル変数をイベント条件に使用

```

Global Integer gi

Function main
    Trap 1, gi = 5 GoTo THandle
    Xqt sub
    Wait 100
    Exit Function

THandle:
    Print "IN Trap ", gi

Fend

Function sub
    For gi = 0 To 10
        Print gi
        Wait 0.5
    Next
Fend

```

Trap (システム状態トリガー)

割込み、および割込み発生時の処理の定義を行います。

Trap 命令を使用すると、イベント発生により、ラベルにジャンプする、または、ファンクションを呼び出すことができます。Trap 命令には 2 つのタイプがあります。

- ユーザーの定義する入力状態をトリガーにする 4 つの Trap
- システム状態をトリガーにする 7 つの Trap

本項目ではシステム状態トリガーの Trap を説明します。

書式

Trap {Emergency | Error | Pause | SGOpen | SGClose | Abort | Finish } Xqt ファンクション名

Trap {Emergency | Error | Pause | SGOpen | SGClose | Abort | Finish }

パラメーター

Emergency	非常停止が発生したとき、指定のファンクションが実行されます。
Error	エラーが発生したとき、指定のファンクションが実行されます。
Pause	一時停止状態に入ったとき、指定のファンクションが実行されます。
SGOpen	安全扉回路が開いたとき、指定のファンクションが実行されます。
SGClose	安全扉回路が閉じたとき、指定のファンクションが実行されます。
Abort	ユーザー、またはシステムによってすべてのタスク (バックグラウンドタスクをのぞく) が停止したとき (Abort All に相当する命令が実行されたときや、中断ボタンが押されたとき)、指定のファンクションが実行されます。
Finish	すべてのタスク (バックグラウンドタスクをのぞく) が終了したとき、指定のファンクションが実行されます。Trap Abort が実行される条件では実行されません。
ファンクション名	システム状態が成立したとき Xqt される割込み処理タスクのファンクションです。 引数のあるファンクションを指定することはできません。 ただし、パラメーターに“Error”が指定されている場合は、3 つの引数を指定できます。

注意

EPSON RC+ 4.x までの Trap *** Call の機能は、EPSON RC+ 7.0 では Trap *** Xqt に置き換えられています。

解説

システム状態が成立したとき指定された割込み処理タスクを実行します。

割込み処理タスクを実行しても、その Trap 設定はクリアされません。

Trap 設定をクリアするためには、ファンクション名を省略して Trap 命令を実行します。

(例) “Trap Emergency” は、“Trap Emergency”をクリアします。

通常タスクがすべて終了してコントローラーが Ready 状態になるとすべての Trap 設定はクリアされます。

割込み処理タスクから Xqt でさらにタスクを実行することはできません。



注意

■ Forcedフラグ

On, Off などの I/O 出力命令に、Forced フラグを指定することで、非常停止中、安全扉開時、ティーチモード中、エラー発生時にも I/O 出力のオンオフが可能になります。

Forced フラグを指定する I/O 出力には、アクチュエーターのような機械的動作を伴う外部機器を絶対に接続しないでください。非常停止中、安全扉開時、ティーチモード中、エラー発生時に外部機器が動作することになり大変危険です。

Forced フラグを指定する I/O 出力は、状態表示 LED のような機械的動作をとみなわない外部機器が接続されることを想定しています。

Emergency を指定したとき

非常停止が発生したとき、指定のファンクションが NoEmgAbort タスク属性で実行されます。

割り込み処理タスクから実行可能なコマンドは NoEmgAbort タスクが実行可能なコマンドです。

非常停止の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready 状態になりません。割り込み処理タスクから Reset コマンドを実行し、自動的に非常停止を解除することはできません。

割り込み処理タスクから I/O のオンオフをタスクが行う場合は、[コントローラー設定]-[環境設定]-[非常停止で出力ポートを OFF]ボックスのチェックをはずしてください。この設定がチェックされたままの場合、コントローラーによる I/O の Off とタスクによる I/O の On のどちらが先に実行されるか保障されません。

Error を指定したとき

エラーが発生したとき、指定のファンクションが NoEmgAbort タスク属性で実行されます。

割り込み処理タスクから実行可能なコマンドは NoEmgAbort タスクが実行可能なコマンドです。

エラーの割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready 状態になりません。

ユーザファンクションに 3 つの省略可能なパラメーター (エラー番号, ロボット番号, 関節番号) を指定できます。これらのパラメーターを使いたいときはトラップファンクションに、3 つの byval 整数パラメーターをつけてください。

モーション系のエラーが発生した場合、エラー番号, ロボット番号, 関節番号が設定されます。

モーション系以外のエラーが発生した場合は、ロボット番号, 関節番号には“0”が設定されます。

Pause を指定したとき

一時停止状態に入ったとき、指定のファンクションが NoPause タスク属性で実行されます。

SGOpen を指定したとき

安全扉回路が開いたとき、指定のファンクションが NoPause タスク属性で実行されます。

SGClose を指定したとき

安全扉回路が閉じてラッチされたとき、指定のファンクションが NoPause タスク属性で実行されます。

割り込み処理タスクから Cont 命令を実行した場合はエラーとなります。

Abort を指定したとき

ユーザー、またはシステムによってすべてのタスク (バックグラウンドタスクをのぞく) が停止したとき (Abort All に相当する命令が実行されたときや中断ボタンが押されたとき)、指定のファンクションが NoPause タスク属性で実行されます。

中断の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready 状態になりません。

Trap Abort で実行されるタスクにおいてエラーが発生しても Trap Error の処理タスクは実行されません。

Shutdown 命令や Restart 命令でタスクが中断された場合は、Trap Abort の処理タスクも Trap Finish の処理タスクも実行されません。

Finish を指定したとき

すべてのタスク (バックグラウンドタスクをのぞく) が終了したとき、指定のファンクションが NoPause タスク属性で実行されます。Trap Abort の処理タスクが実行される条件では実行されません。

終了の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready 状態になりません。

参照

Era, Erl, Err, Ert, ErrMsg\$, OnErr, Reset, Restart, SysErr, Xqt

Trap 使用例

```
Function main
:
  Trap Error Xqt suberr
:
Fend
```

```
Function suberr
  Print "Error =", Err
  On ErrorSwitch
Fend
```

```
Function main

  Trap Error Xqt trapError

FEnd
```

```
Function trapError(errNum As Integer, robotNum As Integer, jointNum
As Integer)
  Print "error number = ", errNum
  Print "robot number = ", robotNum
  Print "joint number = ", jointNum
  If Ert = 0 Then
    Print "system error"
  Else
    Print "task error"
    Print "function = ", Erf$(Ert)
    Print "line number = ", Erl(Ert)
  EndIf
Fend
```


Trim\$ 関数

前後にスペースを含まずに、指定された文字列と同じ文字列を返します。

書式

Trim\$(文字列)

パラメーター

文字列 文字列式を指定します。

戻り値

指定された文字列が前後にスペースを含む場合、スペースは削除されます。

参照

LTrim\$, RTrim\$

Trim\$ 関数使用例

```
str$ = "  data  "  
str$ = Trim$(str$) ' str$ = "data"
```

TW 関数

Wait 命令, WaitNet 命令, WaitSig 命令の状態を返します。

書式

TW

戻り値

時間内に Wait 状態が成立した場合は“False”を返します。
タイムアウトが発生した場合は“True”を返します。

解説

直前の Wait 命令の条件が成立した場合は“False”、タイムアウトが発生した場合は“True”を返します。

参照

TMOut, Wait, Hand_TW

TW 関数使用例

```
Wait Sw(0) = On, 5      ' 入力ビット 0 がオンになるまで 5 秒間待機
If TW = True Then
    Print "Time Up"      ' 5 秒以上の場合は“Time Up”と表示
EndIf
```

UBound 関数

指定した配列に設定できる要素番号の最大値を返します。

書式

UBound (配列変数名 [, 次元])

パラメーター

配列変数名 通常の変数名の名前のつけ方にしたがって、名前をつけます。

次元 要素番号の最大値を返す次元を、下記の整数値で設定します。
省略可能で、省略すると、“1”と仮定されます。

1	1 番目の次元
2	2 番目の次元
3	3 番目の次元

参照

Redim

UBound 関数使用例

```
Integer i, a(10)

For i = 0 to UBound(a)
    a(i) = i
Next
```

UByte

UByte 型の変数を宣言します。(符号なし整数型、サイズ: 2 バイト)

書式

UByte 変数名 [(配列変数の最大要素番号)] [, 変数名[(配列変数の最大要素番号)]...]

パラメーター

変数名 UByte 型と宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

UByte は、変数を UByte 型と宣言するときに使います。UByte 変数の範囲は、0~255 までです。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UInt32, UInt64, UShort

UByte 使用例

下記の例では、UByte 型変数を宣言して、その変数に値を与えています。変数 test_ok の最上位ビットが 1 か 0 かを見えています。その結果はディスプレイに表示されます。(この例では、変数に値 15 が与えられているため、変数 test_ok 値の高いビットが常に設定されています。)

```
Function Test
    UByte A(10)           'UByte 型の 1 次元配列
    UByte B(10, 10)       'UByte 型の 2 次元配列
    UByte C(5, 5, 5)      'UByte 型の 3 次元配列
    UByte test_ok
    test_ok = 15
    Print "Initial Value of test_ok = ", test_ok
    test_ok = (test_ok And 8)
    If test_ok <> 8 Then
        Print "test_ok high bit is ON"
    Else
        Print "test_ok high bit is OFF"
    EndIf
Fend
```

UCase\$ 関数

文字列を小文字から大文字化して返します。

書式

UCase\$ (文字列)

パラメーター

文字列 大文字化する文字列を指定します。

戻り値

大文字化した文字列を返します。

参照

LCase\$, LTrim\$, Trim\$, RTrim\$

UCase\$ 関数使用例

```
str$ = "Data"  
str$ = UCase$(str$)    ' str$ = "DATA"
```

UInt32

UInt32 型の変数を宣言します。(符号なし 4 バイト整数型変数)

書式

UInt32 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数(Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

UInt32 は整数型変数の宣言をします。整数型変数の範囲は 0~4294967295 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt64, UShort

UInt32 使用例

次は UInt32 を使って整数型の変数を宣言するプログラム例です。

```
Function uint32test
    UInt32 A(10)           'UInt32 型の 1 次元配列
    UInt32 B(10, 10)       'UInt32 型の 2 次元配列
    UInt32 C(5, 5, 5)      'UInt32 型の 3 次元配列
    UInt32 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

UInt64

UInt64 型の変数を宣言します。(符号なし 8 バイト整数型変数)

書式

UInt64 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

UInt64 は整数型変数の宣言をします。整数型変数の範囲は 0~18446744073709551615 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UShort

UInt64 使用例

次は UInt64 を使って整数型の変数を宣言するプログラム例です。

```
Function uint64test
    UInt64 A(10)          'UInt64 型の 1 次元配列
    UInt64 B(10, 10)      'UInt64 型の 2 次元配列
    UInt64 C(5, 5, 5)     'UInt64 型の 3 次元配列
    UInt64 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

UOpen

読み出しと書き込みの両方のモードでファイルを開きます。

書式

UOpen ファイル名 As #ファイル番号

・
・

Close #ファイル番号

パラメーター

ファイル名	パスを含んだファイル名の文字列を指定します。 ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。 詳細は、ChDisk を参照してください。
ファイル番号	30~63 の整数値または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは指定したファイルに書き込んだり、データを読み出しするときに使います。

注意

ネットワークパスが使用できます。

存在しないファイルを指定すると、そのファイルを作成し、データを書き込みます。存在するファイルを指定すると、既存データの先頭から、データを読み書きします。

Seek 命令で、ファイルの読み込み/書き込み位置 (ポインター) を切り替えられます。読み込みアクセスと書き込みアクセスを切り替える場合は、Seek 命令でファイルポインターを再設定してください。

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号を使うことはできません。ファイル番号は、ファイル操作命令 (Print#, Input#, Read, Write, Seek, Eof, Flush, Close) で使用します。

Close ステートメントでファイルを閉じ、ファイル番号を解放します。

複数のタスクで同一の番号を使用しないように、FreeFile 関数でファイル番号を取得してください。

参照

Close, Print #, Input#, AOpen, BOpen, ROpen, WOpen, FreeFile, Seek

UOpen 使用例

```
Integer fileNum, i, j

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 10
Input #fileNum, j
Print "data = ", j
Close #fileNum
```

UpdateDB

オープンしているデータベース内の検索されたテーブルのデータを更新します。

書式

UpdateDB #データベース番号, 項目, 値

パラメーター

データベース番号	OpenDB で指定したデータベース番号 (501~508 の整数値)を指定します。
項目	更新するテーブルの項目名を指定します。
値	更新する値を指定します。

解説

オープンされたデータベースの Select されたテーブルのデータを指定された項目の値で更新します。
データの更新前に必ず SelectDB を実行し、更新するレコードを選択する必要があります。

注意

- RC+ がインストールされている PC の接続が必要です

参照

OpenDB, CloseDB, SelectDB, DeleteDB

UpdateDB 使用例

SQL データベースを使用した例

下記に、SQL サーバー2000 のサンプルデータベース Northwind のテーブル Employees にデータを登録し、登録したデータの項目を更新する簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees", "TitleOfCourtesy = 'Mr.'")
Print #501, "Epson", "Taro", "Engineer", "Mr."
count = SelectDB(#501, "Employees", "LastName = 'Epson' and
FirstName = 'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
UpdateDB #501, "Title", "Chief Engineer"
count = SelectDB(#501, "Employees", "LastName = 'Epson' and
FirstName = 'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
CloseDB #501
```

UShort

UShort 型の変数を宣言します。(符号なし 2 バイト整数型変数)

書式

UShort 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名 変数宣言する変数名を指定します。

配列変数の最大要素番号

配列変数の最大要素番号で、最高 3 次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号 1, [最大要素番号 2], [最大要素番号 3])

要素番号は 0 から始まるので、要素数は最大要素番号に 1 加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

ローカル変数	2,000
バックアップ変数 (Global Preserve)	4,000
グローバル変数とモジュール変数	100,000

解説

UShort は整数型変数の宣言をします。整数型変数の範囲は 0~65535 です。ローカル変数は、Function の先頭で宣言します。グローバル変数とモジュール変数は、Function の外で宣言します。

参照

Boolean, Byte, Double, Global, Int32, Int64, Integer, Long, Real, Short, String, UByte, UInt32, UInt64

UShort 使用例

次は UShort を使って整数型の変数を宣言するプログラム例です。

```
Function ushorttest
    UShort A(10)          'UShort 型の 1 次元配列
    UShort B(10, 10)      'UShort 型の 2 次元配列
    UShort C(5, 5, 5)     'UShort 型の 3 次元配列
    UShort var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
End
```

Val 関数

数字からなる指定文字列を数値に変換し、その値を返します。

書式

Val(文字列)

パラメーター

文字列 数字だけからなる文字列式を指定します。文字列にはプレフィックスも含まれます
: &H (16 進法), &O (8 進法), &B (2 進法)

戻り値

整数、または入力された文字列式による浮動小数点数を返します。小数点を含む入力文字列式は、浮動小数点数に変換されます。それ以外で、戻り値は、整数値が返されます。

解説

Val は、数字からなる文字列式を数値に変換します。結果は、整数または浮動小数点数です。Val 命令に小数点を含む入力文字列式が与えられると、浮動小数点数が返されます。それ以外は、整数値が返されます。

参照

Abs, Asc, Chr\$, Int, Left\$, Len, Mid\$, Mod, Right\$, Sgn, Space\$, Str\$

Val 関数使用例

下記は、いくつかの異なる文字列式を数値に変換し、その結果を画面に表示するプログラムの例です。

```
Function ValDemo
    String realstr$, intstr$
    Real realsqr, realvar
    Integer intsqr, intvar

    realstr$ = "2.5"
    realvar = Val(realstr$)
    realsqr = realvar * realvar
    Print "The value of ", realstr$, " squared is: ", realsqr

    intstr$ = "25"
    intvar = Val(intstr$)
    intsqr = intvar * intvar
    Print "The value of ", intstr$, " squared is: ", intsqr
End
```

下記は、コマンドウィンドウからの操作例です。

```
> Print Val("25.999")
25.999
>
```

VSD

スカラロボットの可変速度 CP 動作機能を設定します。

書式

VSD { ON | Off }

パラメーター

On | Off On: スカラロボットの可変速度 CP 動作機能を有効にします。
Off: スカラロボットの可変速度 CP 動作機能を無効にします。

解説

VSD は下記の命令で有効です。

Move, Arc, Arc3

本命令は、スカラ型ロボットのみ有効です。

スカラ型ロボット以外の機種では、AvoidSingularity SING_VSD を使用してください。

可変速度 CP 動作機能は、スカラロボットが CP 動作実行中に、加速度エラーや過速度エラーが発生するのを防止します。動作軌跡を維持したまま、関節速度を自動的に制限して動作する機能です。関節速度が制限された場合、SpeedS で設定されたツール先端速度は維持されませんが、関節速度が制限を下回ると元のツール先端速度に回復します。等速度が優先される場合には、AccelS, DecelS, SpeedS を小さくして、エラーの発生をなくしてください。

VSD を使用しても加速度エラーや過速度エラーが発生する場合は、AccelS, DecelS, SpeedS を小さくしてください。

VSD の設定値を変更した場合、次回コントローラー起動時まで有効です。コントローラー起動時は、VSD はオフになります。

参照

VSD 関数

VSD 使用例

```
VSD On ' 可変速度 CP 動作を有効にして動作させる
Move P1
Move P2
VSD Off
```

VSD 関数

スカルロボットの可変速度 CP 動作機能の設定を返します。

書式

VSD

戻り値

On = 可変速度 CP 動作機能が有効

Off = 可変速度 CP 動作機能が無効

参照

VSD

VSD 関数使用例

```
If VSD = Off Then
    Print "Variable Speed Drive is off"
EndIf
```

VxCalib

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータを作成します。

書式

- (1) VxCalib CalNo
- (2) VxCalib CalNo, CamOrient, P(pixel_st : pixel_ed), P(robot_st : robot_ed) [, TwoRefPoints]
- (3) VxCalib CalNo, CamOrient, P(pixel_st : pixel_ed), P(robot_st : robot_ed), P(ref0) [, P(ref180)]

パラメーター

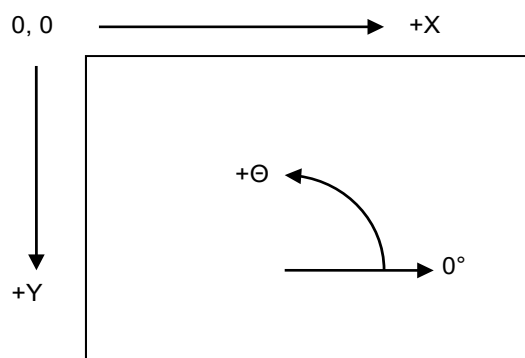
- CalNo** キャリブレーションデータの番号を整数値で指定します。0~15 の整数で、合計 16 まで定義することができます。
- CamOrient** カメラ取付方向を下記の整数値で指定します。
 1~3 は、書式 (2)でのみ指定できます。4~7 は、書式 (3)でのみ指定できます。
 1 : 固定カメラ
 2 : 下向き固定カメラ
 3 : 上向き固定カメラ
 4 : 可動カメラ 第 2 軸搭載
 5 : 可動カメラ 第 4 軸搭載
 6 : 可動カメラ 第 5 軸搭載
 7 : 可動カメラ 第 6 軸搭載
- P(pixel_st : pixel_ed)**
 ピクセル座標 (X, Y のみ)を連続ポイントデータで指定します。
- P(robot_st : robot_ed)**
 ロボット座標を連続ポイントデータで指定します。
 指定するポイントデータは、CamOrient で指定するカメラ取付方向によって異なります。
 CamOrient = 1~3 の場合、
 ロボット座標は、現在の TOOL と ARM で設定してください。
 CamOrient = 4~7 の場合、
 ロボット座標は、TOOL: 0、ARM: 0 で設定してください。
- TwoRefPoints** 書式 (1)の場合に、指定できます。
 2 つの測定ポイントを使用する場合は“True”、1 つの測定ポイントを使用する場合は
 “False”を指定します。2 つの測定ポイントを指定することにより、より正確なキャリブレーションを実施することができます。デフォルトは“False”です。省略可能です。
- P(ref0)** 書式 (3)の場合に、指定できます。
 基準点のロボット座標をポイントデータで指定します。
- P(ref180)** 書式 (3)の場合に、指定できます。
 2 つめの基準点のロボット座標をポイントデータで指定します。基準点を 2 つ指定することにより、より正確なキャリブレーションを実施することができます。省略可能です。

解説

指定したカメラ姿勢を使った指定キャリブレーション番号、引数で渡されたピクセル座標、ロボット座標、基準点 (可動カメラのみ)から、ビジョンキャリブレーションデータを計算します。

CalNo のみを指定した場合、定義時のポイントデータなどが表示されます (コマンドウインドウのみ)。

ピクセル座標の座標系は、次の表のようになります。単位はピクセルです。



ピクセル座標, ロボット座標は、画面上の上左位置をポイント 1 とし、以下表の順番にしたがい、下右位置がポイント 9 になるように設定してください。CamOrient, TwoRefPoints 引数により、4 種類に分類されます。

1) CamOrient = 1~3 (固定, 下向き固定, 上向き固定)、TwoRefPoints = False の場合

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標 1	測定点座標 1
2	上中	検出座標 2	測定点座標 2
3	上右	検出座標 3	測定点座標 3
4	中右	検出座標 4	測定点座標 4
5	中中	検出座標 5	測定点座標 5
6	中左	検出座標 6	測定点座標 6
7	下左	検出座標 7	測定点座標 7
8	下中	検出座標 8	測定点座標 8
9	下右	検出座標 9	測定点座標 9

2) CamOrient = 2 (下向き固定), TwoRefPoints = True の場合

ツールを正確に定義してある場合、TwoRefPoints は必要ありません。“False”に設定してください。

TwoRefPoints を“True”にすると、2 つの測定ポイントを使用するため、より正確なキャリブレーションを実施することができます。

ロボット座標のみ、U 軸: 0 度 / 180 度の 18 ポイントが必要になります。

1~9 の測定点座標を設定した後、U 軸を 180 度回転し、ハンド (ロッドなど) をキャリブレーションターゲット位置に合わせた測定点座標 10~18 を設定してください。

データ順序	画面位置	ピクセル座標	ロボット座標	U 軸
1	上左	検出座標 1	測定点座標 1	0 度
2	上中	検出座標 2	測定点座標 2	
3	上右	検出座標 3	測定点座標 3	
4	中右	検出座標 4	測定点座標 4	
5	中中	検出座標 5	測定点座標 5	
6	中左	検出座標 6	測定点座標 6	
7	下左	検出座標 7	測定点座標 7	
8	下中	検出座標 8	測定点座標 8	
9	下右	検出座標 9	測定点座標 9	
10	上左	---	測定点座標 10	180 度
11	上中	---	測定点座標 11	
12	上右	---	測定点座標 12	
13	中右	---	測定点座標 13	
14	中中	---	測定点座標 14	
15	中左	---	測定点座標 15	
16	下左	---	測定点座標 16	
17	下中	---	測定点座標 17	
18	下右	---	測定点座標 18	

3)CamOrient = 3 (上向き固定), TwoRefPoints = True の場合

ツールを正確に定義してある場合、TwoRefPoints は必要ありません。“False”に設定してください。

TwoRefPoints を“True”にすると、2つの検出ポイントを使用するため、より正確なキャリブレーションを実施することができます。

ピクセル座標のみ、U 軸: 0 度 / 180 度の 18 ポイントが必要になります。

各測定点座標で 1~9 の検出座標を設定した後、U 軸を 180 度回転した位置の検出座標 10~18 を設定してください。

データ順序	画面位置	ピクセル座標	ロボット座標	U 軸
1	上左	検出座標 1	測定点座標 1	0 度
2	上中	検出座標 2	測定点座標 2	
3	上右	検出座標 3	測定点座標 3	
4	中右	検出座標 4	測定点座標 4	
5	中中	検出座標 5	測定点座標 5	
6	中左	検出座標 6	測定点座標 6	
7	下左	検出座標 7	測定点座標 7	
8	下中	検出座標 8	測定点座標 8	
9	下右	検出座標 9	測定点座標 9	
10	上左	検出座標 10	---	180 度
11	上中	検出座標 11	---	
12	上右	検出座標 12	---	
13	中右	検出座標 13	---	
14	中中	検出座標 14	---	
15	中左	検出座標 15	---	
16	下左	検出座標 16	---	
17	下中	検出座標 17	---	
18	下右	検出座標 18	---	

4)CamOrient = 4~7 の場合

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標 1	測定点座標 1
2	上中	検出座標 2	測定点座標 2
3	上右	検出座標 3	測定点座標 3
4	中右	検出座標 4	測定点座標 4
5	中中	検出座標 5	測定点座標 5
6	中左	検出座標 6	測定点座標 6
7	下左	検出座標 7	測定点座標 7
8	下中	検出座標 8	測定点座標 8
9	下右	検出座標 9	測定点座標 9

注意

上表の他に、基準点のロボット座標を指定してください。

2つの基準点を使用すると、より正確なキャリブレーションを実施することができます。その場合、U 軸 0 度 / 180 度の 2 ポイントが必要になります。

1 つめの基準点座標を設定した後、U 軸を 180 度回転し、ハンド (ロッドなど) をキャリブレーションターゲット位置に合わせた 2 つめの基準点座標を設定してください。ツールを正確に定義してある場合、2 つの基準点を使用する必要はありません。

参照

VxTrans 関数, VxCalInfo 関数, VxCalDelete, VxCalSave, VxCalLoad

VxCalib 使用例

```
Function MobileJ2
```

```
    Integer i  
    Double d(8)
```

```
    Robot 1  
    LoadPoints "MobileJ2.pts"
```

```
    VxCalib 0, 4, P(21:29), P(1:9), P(0)
```

```
    If (VxCalInfo(0, 1) = True) Then  
        For i = 0 To 7  
            d(i) = VxCalInfo(0, i + 2)  
        Next i  
        Print "Calibration result:"  
        Print d(0), d(1), d(2), d(3), d(4), d(5), d(6), d(7)
```

```
        P52 = VxTrans(0, P51, P50)  
        Print "Coordinates conversion result:"  
        Print P52  
        SavePoints "MobileJ2.pts"  
        VxCalSave "MobileJ2.caa"
```

```
    Else  
        Print "Calibration failed"  
    EndIf
```

```
End
```

VxCALDelete

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータを削除します。

書式

VxCALDelete CalNo

パラメーター

CalNo キャリブレーションデータの番号を整数値で指定します。
0~15 の整数で、合計 16 まで定義することができます。

解説

指定したキャリブレーション番号で定義してあるキャリブレーションデータを削除します。

参照

VxCALib, VxTrans 関数, VxCALInfo 関数, VxCALSave, VxCALLoad

VxCALDelete 使用例

```
VxCALDelete "MobileJ2.caa"
```

VxCaLLoad

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータをファイルから読み込みます。

書式

VxCaLLoad FileName

パラメーター

FileName キャリブレーションデータを読み込むファイル名を文字列式で指定します。
拡張子は、“**.caa**”固定です。省略した場合、“**.caa**”が追加されます。
“**.caa**”以外の場合、“**.caa**”に変換されます。
パスを指定することはできません。

解説

カレントプロジェクト内にある、指定したファイルからキャリブレーションデータを読み込みます。

参照

VxCaLib, VxTrans 関数, VxCaInfo 関数, VxCaDelete, VxCaSave

VxCaLLoad 使用例

```
VxCaLLoad "MobileJ2.caa"
```

VxCalInfo 関数

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションの終了状態、キャリブレーションデータを返します。

書式

VxCalInfo (CalNo, CalData)

パラメーター

- CalNo** キャリブレーションデータの番号を整数値で指定します。
0~15 の整数で、合計 16 まで定義することができます。
- CalData** 取得したいキャリブレーションデータ種類を下表の整数値で指定します。

CalData	キャリブレーションデータ種類
1	キャリブレーションの終了状態
2	X 方向の平均偏差 [mm]
3	X 方向の最大偏差 [mm]
4	X 方向の 1 ピクセル当たりの長さ [mm]
5	X 方向の傾き
6	Y 方向の平均偏差 [mm]
7	Y 方向の最大偏差 [mm]
8	Y 方向の 1 ピクセル当たりの長さ [mm]
9	Y 方向の傾き

戻り値

指定されたキャリブレーションデータを、CalData = 1 の場合は Bool 型値で、CalData = 2~9 の場合は Double 型値で返します。

解説

どのキャリブレーション番号でキャリブレーションデータが定義されているか確認することができます。
キャリブレーションデータの値を取得することもできます。

参照

VxCalib, VxTrans 関数, VxCalDelete, VxCalSave, VxCalLoad

VxCalInfo 関数 使用例

```
Print VxCalInfo(0, 1)
```

VxCaISave

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータをファイルへ保存します。

書式

VxCaISave FileName

パラメーター

FileName キャリブレーションデータを保存するファイル名を文字列式で指定します。
拡張子は、“**.caa**”固定です。省略した場合、“**.caa**”が自動的に追加されます。
“**.caa**”以外の場合、“**.caa**”に自動的に変換されます。
パスを指定することはできません。

解説

指定したファイル名でキャリブレーションデータを保存します。ファイルはカレントプロジェクト内に保存されます。同じ名前のファイルが存在する場合、キャリブレーションデータは上書きされます。

参照

VxCaIib, VxTrans 関数, VxCaIInfo 関数, VxCaIDelete, VxCaILoad

VxCaISave 使用例

VxCaISave "MobileJ2.caa"

VxTrans 関数

このコマンドは、Vision Guide 以外の、お客様がご用意したビジョンシステムでご使用ください。

ピクセル座標からロボット座標への座標変換を行い、変換したポイントデータを返します。

書式

VxTrans (CalNo, P(pixel)[, P(camRobot)]) As Pose

パラメーター

CalNo	キャリブレーションデータの番号を整数値で指定します。 0~15 の整数で、合計 16 まで定義することができます。
P(pixel)	ビジョンピクセル座標 (X, Y, U のみ)をポイントデータで指定します。
P(camRobot)	省略可能。可動カメラの場合、撮像時のロボットの位置を指定します。 省略した場合、ロボットの現在位置が使用されます。 ポイントデータは、TOOL: 0, ARM: 0 で設定してください。

戻り値

算出されたロボット座標をポイントデータで返します。

解説

指定したキャリブレーション番号のキャリブレーションデータを使用し、ピクセル座標からロボット座標への座標変換を行います。

可動カメラで現在値以外を指定する場合、撮像時のロボット座標 P(camRobot)を指定します。

P(camRobot)は TOOL: 0, ARM: 0 で設定してください。設定したロボット座標の第 4 関節の角度, 第 6 関節の角度が計算に使用されます。

参照

VxCalib, VxCalInfo 関数, VxCalDelete, VxCalSave, VxCalLoad

VxTrans 関数 使用例

```
P52 = VxTrans (0, P51, P50)
```

Wait

MemSw や Sw を使用して、指定したイベント条件が成立するまで、または指定時間の間、プログラムを待機します。(Sw のかわりに Oport を使って、I/O 出力をチェックすることもできます。) グローバル変数の値が変化するのを待機することもできます。

書式

- (1) Wait 時間
- (2) Wait イベント条件式
- (3) Wait イベント条件式, 時間

パラメーター

時間 待機時間を 0~2147483 の実数 (単位: 秒) で指定します。最小有効桁は 0.01 秒です。

イベント条件式 以下の書式でイベント条件を指定します。
[イベント] 比較演算子(=, <, >=, >, <=, <=) [整数式]

次の関数や変数をイベントに使うことができます。

関数 : Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, MCalComplete, Motor, LOF, ErrorOn, SaftyOn, EstopOn, TeachOn, Cnv_QueueLen, WindowsStatus, AtHome, LatchState, WorkQueue_Len, PauseOn, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus

変数 : Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクをかけることや、複合的に組み合わせることができます。

演算子 : And, Or, Xor, Mask

解説

- (1) 時間だけを指定した場合
タイマーとして Wait 命令を使うときは、指定時間だけプログラムを一時停止させ、その後プログラムの実行を継続します。
- (2) イベント条件式だけを指定した場合
条件的インターロック機能として Wait 命令を使うときは、指定されたイベント条件が成立するまで、プログラムを待機します。TMout 命令でタイムアウト指定している場合は、その指定時間が経過してもイベント条件式が成立しないとエラーになります。And, Mask, Or, または Xor 命令などを用いて、1 つの Wait 命令に対し複数の条件をチェックできます。使用例を参照してください。
- (3) 時間とイベント条件式を指定した場合
イベント条件式と時間を指定したときは、条件が成立するか、指定時間が経過するかの、どちらかが成立したときに次のコマンドを実行します。Tw 関数を使って、イベント条件式が成立したか、指定時間が経過したかを確認することができます。

注意

Wait にタイムアウトを併用する場合

Wait 命令に待機時間の指定がないとき、タイムアウトを指定すると指定された状態を待つ制限時間を設定することができます。タイムアウトは、TMOut 命令を使って指定します。詳細は、この TMOut 命令の項を参照してください。(TMOut 命令のデフォルト値は、“0”で、これは時間制限がないことを意味します。)

Wait で変数待ちを行う場合

- 変数待ちできる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が 0.01 秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1 システム内で使用できる変数待ちの個数は最大 64 個です (Till などのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数を Byref で参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数または関数が含まれたときは、その値は Wait 条件設定時に演算されます。意図しない条件になる場合があるので、整数式には変数または関数を使用しないことを推奨します。

PC COM ポート (1001~1008)を使用している場合

- Wait 命令で Lof 関数を使用することはできません。

Wait 実行中に、プログラムが一時停止した場合

Wait 命令実行中に、プログラムが一時停止 (Pause 状態)となった場合でも、Wait 命令は停止しません。イベント条件式が成立するか、指定時間が経過すると、Wait 命令は終了します。

Wait 命令で時間が指定されている場合、指定時間が経過する前に、継続実行でプログラムを再開すると、それまでの経過時間はリセットされ、プログラムは、指定された待機時間の間、待機します。

参照

AtHome, Cnv_QueueLen, Ctr, ErrorOn, EstopOn, GetRobotInsideBox, GetRobotInsidePlane, In, InW, LatchState, LOF, Mask, MCalComplete, MemIn, , MemInW, MemSw, Motor, Oport, Out, OutW, PauseOn, SaftyOn, Sw, TeachOn, TMOut, WindowsStatus, Tw, WorkQueue_Len, SF_GetStatus

Wait 使用例

下記例では、それぞれ動作命令を起動できる 2 つのタスクに対し、他方がロボット制御をしていないときに限ってロボット制御できるインターロック機能が働いています。これにより、それぞれのタスクが指定された動作を順序よく行うことができます。MemSw を併用した Wait 命令が、メモリー I/O ビット 1 が適切な値になるまでプログラムを待機し、安全になってから動作を再開させます。

```
Function main
  Integer I
  MemOff 1
  Xqt !2, task2
  For i = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
```

```
Integer i
For i = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
Next i
Fend

'入力 0 オンまで待機
Wait Sw(0) = On

'60.5 秒待機した後に継続実行
Wait 60.5

'入力 0 がオフになり入力 1 がオンになるまで待機
Wait Sw(0) = Off And Sw(1) = On

'メモリービット 0 がオンまたはメモリービット 1 がオンになるまで待機
Wait MemSw(0) = On Or MemSw(1) = On

'1 秒待機して出力 1 をオン
Wait 1; On 1

'入力ポート 0 の下位 3 ビットが 1 になるまで待機
Wait In(0) Mask 7 = 1

'グローバル Integer 型の変数 giCounter の値が 10 を超えるまで待機
Wait giCounter > 10

'グローバル Long 型の変数 glCheck の値が 30000 になるまで 10 秒間待機
Wait glCheck = 30000, 10
```

WaitNet

TCP/IP ポートの接続が確立するまで待ちます。

書式

WaitNet # ポート番号 [, タイムアウト時間]

パラメーター

ポート番号 接続待ちする TCP/IP のポートの番号を 201~216 の整数値で指定します。

タイムアウト時間 最大接続待ち時間を指定します。省略可能です。

参照

OpenNet, CloseNet

WaitNet 使用例

次は、2 つのコントローラーの TCP/IP 設定の例です。

Controller #1:

Port: #201

Host Name: 192.168.0.2

TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
Fend
```

Controller #2:

Port: #201

Host Name: 192.168.0.1

TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
Fend
```

WaitPos

パスモーションが有効な状態でも、次のステートメントを実行する前にロボットが減速停止するのを待ちます。

書式

WaitPos

解説

通常、パスモーションが有効な状態 (CP On や CP パラメーター指定) の場合、動作命令は減速開始と同時に制御を次のステートメントに移します。

しかし、直後に WaitPos 命令を挿入しておくと、減速動作を完了させてから次の動作に移ります。

参照

Wait, WaitSig, CP

WaitPos 使用例

```
Off 1
CP On
Move P1
Move P2
WaitPos 'ロボットが減速するまで待機
On 1
CP Off
```

WaitSig

他のタスクからの Signal 命令による同期シグナルを待ちます。

書式

WaitSig シグナル番号 [, タイムアウト時間]

パラメーター

シグナル番号 受信するシグナル番号を整数値 (0~63)で指定します。

タイムアウト時間 最大待ち時間を実数値で指定します。省略可能です。

解説

他のタスクからのシグナルを待ちます。WaitSig 実行後にシグナル待ちとなり、それ以前のシグナルは無視されます。

参照

Wait, WaitPos, Signal

WaitSig 使用例

```
Function Main
  Xqt SubTask
    Wait 1
    Signal 1
    .
    .
  Fend

Function SubTask
  WaitSig 1
  Print "signal received"
  .
Fend
```

Weight

PTP 動作時の、速度、加減速度を補正するためのパラメーター設定と表示をします。

書式

Weight [ハンド質量 [, アーム長さ | S | T]]

Weight

パラメーター

ハンド質量	アームにかかるハンドの質量を指定します。(単位: kg 小数点以下 2 桁) 省略可能 (ハンド質量のみを省略することは不可能)です。
アーム長さ	水平多関節型ロボット (RS シリーズを含む)のみ有効です。第 2 関節中心から、第 3 関節中心までを指定します。(単位: mm) 省略可能です。
S	付加軸 S 関節に対する負荷質量を指定します。(単位: kg 小数点以下 2 桁)
T	付加軸 T 関節に対する負荷質量を指定します。(単位: kg 小数点以下 2 桁)

結果

パラメーターを省略すると、現在の Weight 設定値を表示します。

[アーム長さ]を省略すると、入力された [ハンド質量]が設定され、デフォルト値の [アーム長さ]が設定されます。

[ハンド質量]のみを省略することはできません。

解説

PTP 動作における最大加減速度を演算するためのパラメーターを指定します。Weight 命令では、ハンドとワークの質量を設定します。アーム長指定は、水平多関節型ロボット (RS シリーズを含む)の場合のみ必要となります。これは、第 2 関節中心から第 3 関節中心までの距離です。水平多関節型ロボット (RS シリーズを含む)以外では無効になります。ロボットに付加軸が設定されている場合、S、T パラメーターを使って付加軸が持つ負荷質量を個別に設定する必要があります。

設定した値から計算される等価搬送質量が最大可搬質量を超えると、エラーになります。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

“負荷、イナーシャ、偏心/オフセット測定ユーティリティ”による設定も可能です。

詳細は、以下のマニュアルに記載しています。

EPSON RC+ 7.0 ユーザーズガイド 6.18.12 負荷、イナーシャ、偏心/オフセット測定ユーティリティ

起こりやすいエラー

最大許容負荷質量をこえた場合

設定した値から計算される等価負荷質量が最大許容負荷質量を超えると、エラーになります。

マニピュレーターアームへの損傷

Weight のハンド質量を、実際の質量からあまりにかけ離れて軽い値で設定すると、過剰な加速値および減速値が設定され、マニピュレーターに損傷を与えることがありますので注意してください。

注意

Weight の設定値は、電源オフでも変更されません

Weight の設定値は、一度設定するとコントローラーに記憶されます。電源をオフしても変更されません。何も設定しない場合は、前回設定された値のままになります。

参照

Accel, Inertia

ハンドについては、Hand 機能マニュアルを参照してください。

Weight 使用例

コマンドウィンドウから **Weight** 命令で現在の設定値を表示させる例です。

```
> weight  
2.000, 200.000  
>
```

Weight 命令でハンドの質量 (3 kg)を設定する例です。

```
Weight 3.0
```

Weight 命令で付加軸 S の負荷質量 (30 kg)を設定する例です。

```
Weight 30.0, S
```

Weight 関数

Weight 命令で設定したハンド質量とアーム長さを返します。

書式

Weight(パラメーター番号)

パラメーター

パラメーター番号 パラメーター番号を次に示す整数値で設定します。

- 1: ハンド質量
- 2: アーム長
- 3: 付加軸 S 関節負荷質量
- 4: 付加軸 T 関節負荷質量

戻り値

パラメーターを実数値で返します。

参照

Inertia, Weight

ハンドについては、Hand 機能マニュアルを参照してください。

Weight 関数使用例

```
Print "The current Weight parameters are: ", Weight(1)
```


Where

ロボットの現在の位置データを表示します。

書式

Where [ローカル番号]

パラメーター

ローカル番号 ローカル座標系番号を指定します。デフォルトは“Local 0”です。省略可能です。

参照

Joint, PList, Pulse

Where 使用例

表示される形式は、ロボットのタイプや付加軸の有無で変わります。

以下の例はスカラ型ロボットで付加軸無しの場合です。

```
>where
WORLD: X: 350.000 mm Y: 0.000 mm Z: 0.000 mm U: 0.000 deg V: 0.000 deg
W: 0.000 deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 mm 4: 0.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 0 pls

> local 1, 100,100,0,0

> where 1
WORLD: X: 250.000 mm Y:-100.000 mm Z: 0.000 mm U: 0.000 deg V: 0.000
deg W: 0.000 deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 mm 4: 0.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 0 pls
```

WindowsStatus 関数

Windows の起動状態を返します。

書式

WindowsStatus

戻り値

現在の Windows 起動状態を整数値で返します。Windows 起動状態はビットイメージで返され、以下の状態を示します。

機能名	システム予約	RC+機能可能	PC 機能可能
ビット番号	15 - 2	1	0
利用可能機能の詳細		Vision Guide (フレームグラバタイプ) RC+ API フィールドバスマスター	PC ファイル PC RS-232C データベースアクセス DLL 呼び出し

注意

使用可能なコントローラーについて

T/VT シリーズでは使用できません。

解説

本関数は、コントローラー設定を“独立モード”に設定した場合に、コントローラーの起動状態を確認するために使用します。コントローラー設定が“連携モード”に設定されている場合は、RC+機能、PC 機能の両方が使用可能になるまでプログラム実行を開始できません。

WindowsStatus 関数使用例

```
Print "The current PC Booting up Status is: ", WindowsStatus
```

WOpen

ファイルを書き込みモードで開きます。

書式

WOpen ファイル名 As #ファイル番号

・
・

Close #ファイル番号

パラメーター

ファイル名	パスを含んだファイル名の文字列を指定します。 ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。 詳細は、ChDisk を参照してください。
ファイル番号	30~63 の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは、指定したファイルを開いてデータを書き込むときに使います。(すでにあるファイルにデータを書き加えるときは、AOpen についての説明を参照してください。)

指定したファイルがないときは、新規でファイルを作成して、そこに書き込みます。指定したファイルがあれば、既存のデータをすべて削除して、新たに書き込みを行います。

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号を使えません。ファイル番号は、ファイル操作命令 (Print#, Write, Seek, Flush, Close) で使用します。

Close ステートメントでファイルを閉じ、ファイル番号を解放します。

複数のタスクで同一の番号を使用しないように、FreeFile 関数でファイル番号を取得してください。

注意

ネットワークパスが使用できます。

ファイルへの書き込みはバッファリングされます。

バッファリングされたデータは Flush ステートメントで書き込むことができます。Close ステートメントでファイルを閉じた場合も書き込まれます。

参照

AOpen, BOpen, Close, Print#, ROpen, UOpen, FreeFile

WOpen 使用例

```
Integer fileNum, i, j

fileNum = FreeFile
WOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Close #fileNum

fileNum = FreeFile
ROpen "TEST.DAT" As #fileNum
For i = 0 to 100
    Input #fileNum, j
    Print "data = ", j
Next i
Close #fileNum
```

WorkQue_Add

指定したワークキューにワークキューデータ (ポイントデータとユーザーデータ)を追加します。

書式

WorkQue_Add ワークキュー番号, ポイントデータ [, ユーザーデータ]

パラメーター

ワークキュー番号	ワークキューの番号を整数値 (1~16)で指定します。
ポイントデータ	ワークキューに追加するポイントデータを指定します。
ユーザーデータ	ポイントデータと一緒に登録するユーザーデータを実数値で指定します。 省略可能です。省略すると、0 (実数)がユーザーデータとして登録されます。

解説

ポイントデータとユーザーデータはワークキューの最後尾に追加されます。
ただし、WorkQue_SortによりSort方法が設定されている場合は、その設定されたSort方法にしたがって登録されます。

WorkQue_Rejectにより2重登録防止距離が設定されている場合は、既に登録済みのポイントデータとの距離が計算され、その距離以下のポイントデータであれば、ポイントデータとユーザーデータはワークキューに追加されません。この場合、エラーにはなりません。

ワークキューデータの上限值は“1000”です。ワークキューデータを使い終わったときにWorkQue_Removeにより、ワークキューデータを削除します。

参照

WorkQue_AutoRemove, WorkQue_Len, WorkQue_Reject, WorkQue_Remove, WorkQue_Sort

WorkQueAdd 使用例

```
Integer x, y
Real u

P0 = XY(300, 300, 300, 90, 0, 180)
P1 = XY(200, 280, 150, 90, 0, 180)
P2 = XY(200, 330, 150, 90, 0, 180)
P3 = XY(-200, 280, 150, 90, 0, 180)

Pallet 1, P1, P2, P3, 10, 10
x = 1
y = 1
u = 5.3
WorkQue_Add 1, Pallet(1, x, y), u
```

WorkQue_AutoRemove

指定したワークキューに自動削除機能を設定します。

書式

WorkQue_AutoRemove ワークキュー番号,{True | False}

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

True | False True: 自動削除機能を有効にします。
False: 自動削除機能を無効にします。

解説

ワークキューに自動削除機能を設定します。自動削除機能を有効にした場合、WorkQue_Getによりワークキューからポイントデータを取得したときに、ポイントデータとユーザーデータがワークキューから自動削除されます。

自動削除機能を無効にした場合は、ポイントデータとユーザーデータは削除されません。削除するには、WorkQue_Removeを使います。

WorkQue_UserDataにより、ユーザーデータを取得した場合は、自動削除されません。

ワークキュー毎に、自動削除機能の有効/無効が設定可能です。

参照

WorkQue_AutoRemove 関数, WorkQue_Get

WorkQue_AutoRemove 使用例

```
WorkQue_AutoRemove 1, True
```

WorkQue_AutoRemove 関数

ワークキューに設定されている自動削除機能の状態を返します。

書式

WorkQue_AutoRemove (ワークキュー番号)

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

戻り値

指定ワークキューの自動削除機能が有効の場合は“True”、無効の場合は“False”を返します。

参照

WorkQue_AutoRemove, WorkQue_Get

WorkQue_AutoRemove 関数使用例

```
Boolean autoremove  
  
autoremove = WorkQue_AutoRemove (1)
```

WorkQue_Get 関数

指定したワークキューからポイントデータを返します。

書式

WorkQue_Get(ワークキュー番号 [, インデックス])

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16) で指定します。
インデックス 取得するキューデータのインデックスを整数値で指定します。
 (先頭インデックス番号は 0 です。)省略可能です。

戻り値

指定したワークキューからポイントデータを返します。

解説

WorkQue_Get は、ワークキューからポイントデータを取得します。インデックスを省略した場合、キューデータの先頭データを返します。インデックスを設定した場合は、設定したインデックスのポイントデータを返します。

WorkQue_AutoRemove により、キューデータの自動削除機能を有効にしている場合は、WorkQue_Get により、ポイントデータとユーザーデータを削除します。
 無効にしている場合は、ポイントデータとユーザーデータは削除しません。削除するには WorkQue_Remove を使います。

参照

WorkQue_AutoRemove, WorkQue_Len, WorkQue_Reject, WorkQue_Remove, WorkQue_Sort

WorkQue_Get 関数使用例

```
・ キューの先頭のワークへ Jump し、追従する
Jump WorkQue_Get (1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
WorkQueRemove 1
```


WorkQue_Len 関数

指定したワークキューに登録された有効なワークキューデータの数を返します。

書式

WorkQue_Len(ワークキュー番号)

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

戻り値

有効なワークキューデータの登録数を整数値で返します。

解説

有効なワークキューデータ数の登録数を返します。

Wait 命令の引数としても使うことが可能です。

参照

WorkQue_Add, WorkQue_Get, WorkQue_Remove

WorkQue_Len 関数使用例

```
Do
  Do While WorkQue_Len(1) > 0
    WorkQue_Remove 1, 0
  Loop
  If WorkQue_Len(1) > 0 Then
    Jump WorkQue_Get(1, 0) C0
    On gripper
    Wait .1
    WorkQue_Remove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

WorkQue_List

指定したワークキューのワークキューデータ一覧 (ポイントデータとユーザーデータ)を表示します。

書式

WorkQue_List ワークキュー番号 [, 表示数]

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

表示数 表示数データ数を整数値で指定します。省略可能です。
省略すると、すべてのキューデータを表示します。

注意

このコマンドはコマンドウィンドウのみ実行可能です。

参照

WorkQue_Add, WorkQue_Get, WorkQue_Remove

WorkQue_List 使用例

コマンドウィンドウでの操作例

```
> WorkQue_List 1
Queue 0 = XY( 1.000, 1.000, 0.000, 0.000 ) /R /0 ( 0.000)
Queue 1 = XY( 3.000, 1.000, 0.000, 0.000 ) /R /0 ( 2.000)
Queue 2 = XY( 4.000, 1.000, 0.000, 0.000 ) /R /0 ( 3.000)
Queue 3 = XY( 5.000, 1.000, 0.000, 0.000 ) /R /0 ( 4.000)
Queue 4 = XY( 6.000, 1.000, 0.000, 0.000 ) /R /0 ( 5.000)
```

WorkQue_Reject

指定したワークキューにポイントデータの2重登録防止のための最小距離を設定、表示します。

書式

WorkQue_Reject ワークキュー番号 [, 2重登録防止距離]

パラメーター

- ワークキュー番号** ワークキューの番号を整数値 (1~16)で指定します。
- 2重登録防止距離** 2重登録防止距離の最小値を実数値 (単位: mm)で指定します。マイナス値を指定した場合は、0 mm が設定されます。コマンドウインドウから実行する場合、省略可能です。省略すると、現在の2重登録防止距離が表示されます。

解説

ポイントデータの2重登録防止のための最小距離を設定します。WorkQue_Addにより、最小距離以下のポイントデータを登録しても、ワークキューには登録されません。WorkQue_Rejectは2重登録防止のシステムフィルターとなります。デフォルトは0 mmです。

WorkQue_Addにより、ワークキューデータ (ポイントデータとユーザーデータ)を追加する前に、WorkQue_Rejectを実行する必要があります。

ワークキュー毎に、2重登録防止距離が設定可能です。

参照

WorkQue_Add, WorkQue_Reject 関数

WorkQue_Reject 使用例

```
WorkQue_Reject 1, 2.5
```

WorkQue_Reject 関数

指定したワークキューに設定されている 2 重登録防止距離を返します。

書式

WorkQue_Reject (ワークキュー番号)

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

戻り値

実数値 (単位: mm)を返します。

参照

WorkQue_Add, WorkQue_Reject

WorkQue_Reject 関数使用例

```
Real rejectDist  
  
RejectDist = WorkQue_Reject(1)
```

WorkQue_Remove

指定したワークキューからワークキューデータ (ポイントデータとユーザーデータ)を削除します。

書式

WorkQue_Remove ワークキュー番号 [, インデックス | All]

パラメーター

ワークキュー番号 コンベヤーの番号を整数値 (1~16)で指定します。

インデックス 削除するワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は 0 です。)省略可能です。ワークキューからすべてのワークキューデータを削除する場合は All で指定します。

解説

ワークキューデータから 1 つ以上のワークキューデータ (ポイントデータとユーザーデータ)を削除します。ワークキューデータを使い終わったときに、キューデータを削除します。

参照

WorkQue_Add

WorkQue_Remove 使用例

```
Jump WorkQue_Get (1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
```

```
・ ワークキューからデータを削除
WorkQue_Remove 1
```

WorkQue_Sort

指定したワークキューに Sort 方法を設定、表示します。

書式

WorkQue_Sort ワークキュー番号 [, Sort 方法]

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

Sort 方法 Sort 方法を整数値 (0~6)、もしくは以下に示す定数で指定します。コマンドウインドウから実行する場合、省略可能です。省略すると、現在の Sort 方法が表示されます。

定数	値	内容
QUE_SORT_NONE	0	並べ替えなし (ワークキューへの登録順)
QUE_SORT_POS_X	1	X 座標昇順
QUE_SORT_INV_X	2	X 座標降順
QUE_SORT_POS_Y	3	Y 座標昇順
QUE_SORT_INV_Y	4	Y 座標降順
QUE_SORT_POS_USER	5	ユーザーデータ (実数)昇順
QUE_SORT_INV_USER	6	ユーザーデータ (実数)降順

解説

ワークキューに Sort 方法を設定します。WorkQue_Add により、ポイントデータとユーザデータを追加した際、設定された Sort 方法にしたがってワークキューに登録されます。

WorkQue_UserData により、ユーザデータの再設定をした際、設定された Sort 方法にしたがってワークキューの並び順が変わります。

WorkQue_Add により、ワークキューデータ (ポイントデータとユーザーデータ)を追加する前に、WorkQue_Sort を実行する必要があります。

WorkQue_UserData により、ユーザーデータの再設定を行う前に、WorkQue_Sort を実行する必要があります。

ワークキュー毎に、Sort 方法が設定可能です。

参照

WorkQue_Add, WorkQue_UserData

WorkQue_Sort 使用例

```
WorkQue_Sort 1, QUE_SORT_POS_X
```

WorkQue_Sort 関数

指定したワークキューに設定されている Sort 方法を返します。

書式

WorkQue_Sort (ワークキュー番号)

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16)で指定します。

戻り値

ワークキューに設定されている Sort 方法を整数値で返します。

- 0 = 並び替えなし (ワークキューへの登録順)
- 1 = X 座標昇順
- 2 = X 座標降順
- 3 = Y 座標昇順
- 4 = Y 座標降順
- 5 = ユーザーデータ (実数)昇順
- 6 = ユーザーデータ (実数)降順

参照

WorkQue_Add, WorkQue_Sort, WorkQue_UserData

WorkQue_Sort 関数使用例

```
Integer quesort  
  
quesort = WorkQue_Sort(1)
```

WorkQue_UserData

指定したワークキューに登録されているユーザーデータ (実数) の再設定, 表示をします。

書式

WorkQue_UserData ワークキュー番号 [, インデックス] [, ユーザーデータ]

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16) で指定します。

インデックス ワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は 0 です。) コマンドウインドウから実行する場合、省略可能です。

ユーザーデータ 再設定するユーザーデータを実数値で指定します。コマンドウインドウから実行する場合、省略可能です。省略すると、現在のユーザーデータ (実数) が表示されます。

解説

ワークキューに登録されているユーザーデータの再設定, 表示をします。

WorkQue_Sort により、以下の Sort 方法が設定されている場合は、その設定された Sort 方法にしたがってワークキューデータの並び順が変わります。

QUE_SORT_POS_USER : ユーザーデータ (実数) 昇順

QUE_SORT_INV_USER : ユーザーデータ (実数) 降順

参照

WorkQue_UserData 関数

WorkQue_UserData 使用例

```
WorkQue_UserData 1, 1, angle
```


WorkQue_UserData 関数

指定したワークキューに登録されているユーザーデータ (実数) を返します。

書式

WorkQue_UserData (ワークキュー番号 [, インデックス])

パラメーター

ワークキュー番号 ワークキューの番号を整数値 (1~16) で指定します。

インデックス ワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は 0 です。) 省略可能です。

戻り値

実数値を返します。

参照

WorkQue_UserData

WorkQue_UserData 関数使用例

```
' キューから削除
angle = WorkQue_UserData (1) ' デフォルトのインデックス“0”になる
Jump WorkQue_Get (1) :U (angle)
WorkQue_Remove 1
```

Wrist

ポイントの手首姿勢を設定します。

書式

(1) **Wrist** ポイント指定 [, Flip | NoFlip]

(2) **Wrist**

パラメーター

ポイント指定 P 番号, P(式), ポイントラベルのいずれかを指定します。

Flip | NoFlip 手首姿勢を指定します。

結果

2つのパラメーターを両方省略すると、ロボットの現在位置の手首姿勢を表示します。

Flip | NoFlip を省略すると、指定ポイントの手首姿勢を表示します。

参照

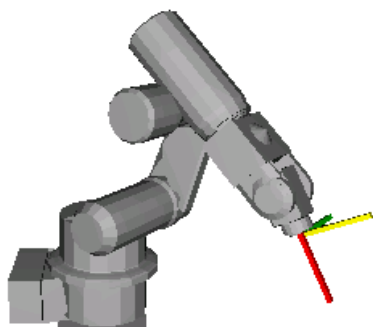
Elbow, Hand, J4Flag, J6Flag, Wrist 関数

Wrist 使用例

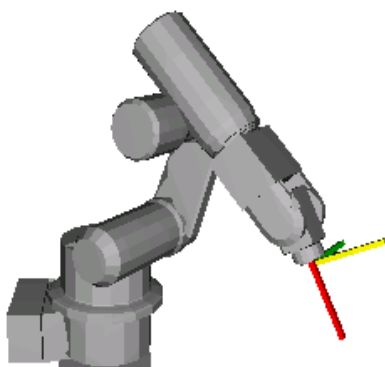
```
Wrist P0, Flip
```

```
Wrist P(my point), NoFlip
```

```
P1 = 320.000, 400.000, 350.000, 140.000, 0.000, 150.000
```



```
Wrist P1, NoFlip  
Go P1
```



```
Wrist P1, Flip  
Go P1
```

Wrist 関数

ポイントの手首姿勢を返します。

書式

Wrist [(ポイント指定)]

パラメーター

ポイント指定 P 番号, P(式), ポイントラベル, ポイント式のいずれかで指定します。省略可能で、ポイント指定を省略すると、現在のロボット位置の手首姿勢を返します。

戻り値

- 1 NoFlip (/NF)
- 2 Flip (/F)

参照

Elbow, Hand, J4Flag, J6Flag, Wrist

Wrist 関数使用例

```
Print Wrist(pick)
Print Wrist(P1)
Print Wrist
Print Wrist(P1 + P2)
```

Write

ファイル、または通信ポートに文字列を書き込みます。行末ターミネーターは付加しません。

書式

Write # ポート番号, 文字列

パラメーター

ポート番号	ファイルまたは通信ポートを示す ID 番号です。 ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。 通信ポート番号は、OpenCom (RS-232C)または OpenNet (TCP/IP)ステートメントで指定した番号です。
文字列	書き込む文字列を指定します。

解説

Write コマンドは Print コマンドとは異なり、行末ターミネーターは付加しません。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

ファイルへの書き込みはバッファリングされます

バッファリングされたデータは Flush ステートメントで書き込むことができます。Close ステートメントでファイルを閉じた場合も書き込まれます。

参照

Print, Read, WriteBin

Write 使用例

```
OpenCom #1
For i = 1 to 10
    Write #1, data$(i)
Next i
CloseCom #1
```

WriteBin

ファイル、または通信ポートへバイナリーデータを書き出します。

書式

WriteBin #ポート番号, 書き込みデータ

WriteBin #ポート番号, 配列変数名(), バイト数

パラメーター

ポート番号

ファイルまたは通信ポートを示す ID 番号です。
ファイル番号は、BOpen などのステートメントで指定した番号です。
通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。

書き込みデータ

書き込むデータを整数または式で指定します。

配列変数名()

書き出すデータバイトを格納している、Byte 型変数、整数変数、または Long 型変数の名前を指定します。指定可能なのは 1 次元の配列変数です。

バイト数

書き出すバイト数を指定します。
配列の最大要素番号以下で、かつ 256Byte 以下である必要があります。
通信ポート (TCP/IP) が対象の場合は、配列の最大要素番号以下で、かつ 1024Byte 以下である必要があります。

注意

使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

参照

ReadBin, Write

WriteBin 使用例

```
Integer i, data(100)

OpenCom #1
For i = 0 To 100
    WriteBin #1, i
Next i
WriteBin #1, data(), 100
CloseCom #1
```

Xor 演算子

2 つの値をビット単位で Xor 演算 (排他的 OR 演算)します。

書式

result = 値 1 Xor 値 2

パラメーター

値 1, 値 2 数値、または変数名を指定します。

result 整数を返します。

結果

ビット単位で Xor 演算した result を返します。

解説

Xor オペレーターはビット単位で Xor 演算を行います。result のビットは、2 つの値をビット単位で Xor した結果です。

値 1 のビット	値 2 のビット	result
0	0	0
0	1	1
1	0	1
1	1	0

参照

And, LShift, Not, Or, Rshift

Xor 演算子使用例

```
>print 2 Xor 6
4
>
```

Xqt

ファンクション名で指定したプログラムを実行し、タスクを生成します。

書式

Xqt [タスク番号,] ファンクション名 [(引数リスト)] [, Normal | NoPause | NoEmgAbort]

パラメーター

タスク番号	実行するタスクのタスク番号を、1~32 の整数で指定します。省略可能です。バックグラウンドタスクの場合、65~80 の整数を指定します。
ファンクション名	実行するファンクション名を指定します。
引数リスト	呼び出されたときにファンクションに渡される引数のリストを指定します。引数が複数ある場合は、カンマで区切ってください。省略可能です。
タスクタイプ	省略可能です。通常は省略します。バックグラウンドタスクの場合、タスクタイプの指定は意味を持ちません。
Normal	通常のタスクを生成します。
NoPause	Pause ステートメントや、Pause 入力信号が発生したときに、および、安全扉開状態で一時停止しないタスクを生成する場合に指定します。
NoEmgAbort	非常停止時、およびエラー発生時に処理を継続するタスクを生成する場合に指定します。

解説

Xqt は、指定されたファンクションを開始し、すぐに戻ります。

通常は、タスク番号パラメーターは必要ありません。タスク番号を省略すると、SPEL+が自動的にファンクションにタスク番号をつけるので、ユーザーがタスク番号の管理をする必要はありません。

注意

タスクタイプ

タスクタイプで、NoPause や NoEmgAbort を指定することにより、コントローラー全体を監視するようなタスクを生成することができます。

ただし、これら特殊なタスクは、SPEL+のタスクの動作や特殊なタスクの制限事項を十分理解した上で使用することを強く推奨します。

特殊なタスクの詳細については、EPSON RC+ ユーザーズガイド「特殊なタスク」に記載されています。

バックグラウンドタスク

バックグラウンドタスクで Xqt を実行した場合、生成されるタスクもバックグラウンドタスクとなります。

バックグラウンドタスクからメインファンクションを実行するには、StartMain 命令を使用してください。

バックグラウンドタスクの詳細は、EPSON RC+ ユーザーズガイド「特殊なタスク」に記載されています。

NoEmgAbort タスクとバックグラウンドタスクで実行できないコマンド

NoEmgAbort タスクとバックグラウンドタスクでは以下のコマンドを実行することはできません。

A Accel	Cnv_Trigger	O OLAcel	VCreateCalibration
AccelR	Cnv_UpStream	P Pass	VCreateObject
AccelS	CollisionDetect	PerformMode	VCreateSequence
AIO_TrackingStart	CP	Pg_LSpeed	VDefArm
AIO_TrackingEnd	CP_Offset	Pg_Scan	VDefGetMotionRange
Arc	Curve	Plane	VDefLocal
Arc3	CVMove	PlaneClr	VDefSetMotionRange
Arch	E ECP	Power	VDefTool
Arm	ECPClr	PTPBoost	VDeleteCalibration
ArmCalib	ECPSet	Pulse	VDeleteObject
ArmCalibCLR	F Find	Q QP	VDeleteSeuence
ArmCalibSET	Fine	QPDcelR	VEditWindow
ArmClr	FineDist	QPDcelS	VGet
ArmSet	Force_Calibrate	R Range	VGoCenter
AutoLJM	Force_ClearTrigger	Reset *1	VLoad
AutoOrientationFlag	Force_Sensor	Restart *2	VLoadModel
AvoidSingularity	Force_SetTrigger	S Sense	VRun
B Base	G Go	SetLatch	VSave
BGo	H Hand_On	SFree	VSaveImage
BMove	Hand_Off	SingularityAngle	VSaveModel
Box	Home	SingularityDist	VSet
BoxClr	HomeClr	SingularitySpeed	VShowModel
Brake	HomeSet	SLock	VStasShow
C Calib	Hordr	SoftCP	VStatsReset
Cnv_AbortTrack	I Inertia	Speed	VStatsResetAll
Cnv_Accel	J JTran	SpeedFactor	VStatsSave
Cnv_AccelLim	Jump	SpeedR	VSD
Cnv_Adjust	Jump3	SpeedS	VStatsShow
Cnv_AdjustClear	Jump3CP	SyncRobots	VTech
Cnv_AdjustGet	JRange	T TC	VTrain
Cnv_AdjustSet	L LatchEnable	TGo	W WaitPos
Cnv_DownStream	LimitTorque	Till	Weight
Cnv_Fine	LimZ	TLSet	WorkQue_Add
Cnv_Mode	LimZMargin	TLClr	WorkQue_Reject
Cnv_OffsetAngle	Local	TMove	WorkQue_Remove
Cnv_QueueAdd	LocalClr	Tool	WorkQue_Sort
Cnv_QueueMove	M MCal	Trap	WorkQue_UserData
Cnv_QueueReject	MCodr	V VCal	X Xqt *3
Cnv_QueueRemove	Motor	VcalPoints	XYLim
Cnv_QueueUserData	Move	VClS	

*1 Reset Error は実行可能

*2 Trap Error の処理タスクからは実行可能

*3 バックグラウンドタスクからは実行可能

ループ文の中で XQT 命令を頻繁に繰り返すような使い方はしないでください

Do...Loop 等ループ文の中で XQT 命令を頻繁に繰り返すような使い方はしないでください。コントローラーがハングアップ状態になる可能性があります。このような使い方をする場合は、Wait 命令(Wait 0.1)を入れてください。

参照

Function/Fend, Halt, Resume, Quit, Startmain, Trap

Xqt 使用例

```
Function main
  Xqt flash          'タスク flash を開始
  Xqt Cycle(5)       'タスク Cycle を開始

  Do
    Wait 3           'タスク flash を 3 秒間実行
    Halt flash       'タスクを一時停止

    Wait 3
    Resume flash     'タスクを再開
  Loop
Fend

Function Cycle(count As Integer)
  Integer i

  For i = 1 To count
    Jump pick
    On vac
    Wait .2
    Jump place
    Off vac
    Wait .2
  Next i
Fend

Function flash
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

XY 関数

指定された座標値をポイントデータとして返します。

書式

XY(x 座標要素, y 座標要素, z 座標要素, u 座標要素 [, v 座標要素, w 座標要素])

パラメーター

x 座標要素	x 座標を実数値で指定します。
y 座標要素	y 座標を実数値で指定します。
z 座標要素	z 座標を実数値で指定します。
u 座標要素	u 座標を実数値で指定します。
v 座標要素	v 座標を実数値で指定します。 垂直 6 軸型ロボット (N シリーズを含む)用パラメーターで、省略可能です。
w 座標要素	w 座標を実数値で指定します。 垂直 6 軸型ロボット (N シリーズを含む)用パラメーターで、省略可能です。

戻り値

指定された座標値をポイントデータとして返します。

解説

付加軸(S 軸, T 軸)を使用していない場合は特別な注意はありません。

Go XY (60,30,-50,45) というような使い方でロボットを指定の座標値へ動作させることができます。

付加軸 (S 軸, T 軸)を使用している場合は注意が必要です。

XY 関数は付加軸を除いたロボットのポイントデータしか返しません。

Go XY (60,30,-50,45) というような使い方をした場合、ロボットは指定の座標値へ動作しますが、付加軸は動きません。付加軸も同時に動作させるようなケースでは Go XY (60,30,-50,45) : ST (10,20) のような指定をしてください。

詳細は、ユーザズガイドの「21. 付加軸」を参照してください。

参照

JA, P# = ポイント指定 , ST 関数

XY 関数使用例

P10 = **XY** (60, 30, -50, 45) + P20

XYLim

許容動作エリアの設定と表示をします。

書式

XYLim X 軸下限位置, X 軸上限位置, Y 軸下限位置, Y 軸上限位置 [, Z 軸下限位置] [, Z 軸上限位置]
XYLim

パラメーター

X 軸下限位置	マニピュレーターが動作できる下限位置の X 座標値 (実数) を、数値または式で指定します。
X 軸上限位置	マニピュレーターが動作できる上限位置の X 座標値 (実数) を、数値または式で指定します。
Y 軸下限位置	マニピュレーターが動作できる下限位置の Y 座標値 (実数) を、数値または式で指定します。
Y 軸上限位置	マニピュレーターが動作できる上限位置の Y 座標値 (実数) を、数値または式で指定します。
Z 軸下限位置	マニピュレーターが動作できる下限位置の Z 座標値 (実数) を、数値または式で指定します。省略可能です。
Z 軸上限位置	マニピュレーターが動作できる上限位置の Z 座標値 (実数) を、数値または式で指定します。省略可能です。

結果

パラメーターを省略すると、現在の XYLim 値を表示します。

解説

XYLim は、許容動作エリアを設定します。多くのロボットシステムでは、ユーザーが関節の動作許容範囲を設定できるようになっていますが、SPEL+ 言語では関節の動作範囲だけでなく、動作許容範囲も設定できるようになっています。これにより、ロボットのアプリケーションに応じて稼動範囲を制限できます。

XYLim 値で設定した動作範囲は、XYLimMode コマンドで設定された監視方法で適用されます。監視方法の詳細は、XYLimMode ステートメントを参照してください。

ロボットパラメーターデータは、コントローラー内のコンパクトフラッシュに格納されています。このため、本コマンドを実行するとコンパクトフラッシュへの書き込みが発生します。頻繁なコンパクトフラッシュへの書き込みは、コンパクトフラッシュの寿命に影響します。本コマンド実行を必要最小限にとどめることを推奨します。

注意

許容動作エリアの設定をオフにする

許容動作エリアの設定が必要でない用途も数多くあります。そのため、この設定は簡単にオフできるようになっています。この設定をオフにするためには、各パラメーター値 (X 軸下限位置, X 軸上限位置, Y 軸下限位置, Y 軸上限位置) を“0”に設定してください。例: XYLim 0, 0, 0, 0.

許容動作制限値のデフォルト

XYLim のデフォルトは、“0, 0, 0, 0”です。(許容動作エリアの設定は、オフになります。)

ヒント

XYLim をポイント&クリックで設定する

EPSON RC+ では、[プロジェクト]メニュー-[ロボットマネージャー]の[XYLim]パネルから XYLim 値を設定できます。

参照

Range, XYLimMode

XYLim 使用例

下記はコマンドウィンドウから XYLim 値を設定し、現在値を表示する簡単な操作例です。

```
> XYLim -200, 300, 0, 500  
  
> XYLim  
-200.000, 300.000, 0.000, 500.000
```

XYLim 関数

設定された XY 平面許容動作エリアを返します。

書式

XYLim(参照データ)

パラメーター

参照データ 戻り値として返す許容エリアを整数値で指定します。
1: 下限値
2: 上限値

戻り値

参照データとして 1 を指定すると、XYLim 設定値で指定した X 軸下限位置をポイントデータの X として、Y 軸下限位置をポイントデータの Y として、Z 軸下限位置をポイントデータの Z として返します。
参照データとして 2 を指定すると、XYLim 設定値で指定した X 軸上限位置をポイントデータの X として、Y 軸上限位置をポイントデータの Y として、Z 軸上限位置をポイントデータの Z として返します。

参照

XYLim

XYLim 関数使用例

```
P1 = XYLim(1)  
P2 = XYLim(2)
```

XYLimClr

設定された XYLim をクリアします。

書式

XYLimClr

参照

XYLim, XYLimDef

XYLimClr 関数使用例

下記は XYLimClr 関数を使用したプログラムです。

```
Function ClearXYLim  
  
    If XYLimDef = True Then  
        XYLimClr  
    EndIf  
End
```

XYLimDef 関数

XYLim を設定したかどうかを返します。

書式

XYLimDef

戻り値

XYLim が設定されている場合は“True”、設定されていない場合は“False”を返します。

参照

XYLim, XYLimClr

XYLimDef 関数使用例

下記は XYLimDef 関数を使用したプログラムです。

```
Function ClearXYLim  
  
    If XYLimDef = True Then  
        XYLimClr  
    EndIf  
End
```

XYLimMode

XYLim の監視方法の設定と表示をします。

書式

- (1) XYLimMode 監視方法
- (2) XYLimMode

パラメーター

監視方法 使用する XYLim の監視方法を表す整数式

定数	値	内容
XYLIM_STANDARD	0	動作コマンドの目標座標に対して XYLim を適用します。(Pulse に対しては影響しません。)
XYLIM_STRICT	1	XYLIM_STANDARD の監視方法に加え、動作軌跡とパルス動作に対しても XYLim を適用します。

結果

パラメーターを省略すると、現在設定されている XYLim の監視方法を表示します。

解説

XYLimMode は、指定されているロボットの XYLim の監視方法を設定します。

XYLIM_STANDARD を指定した場合、XYLim で設定した動作範囲は、動作コマンドの目標座標に対してのみ有効となります。動作の開始地点から目標座標までの動作軌跡に対しては適用されません。したがって、動作中、アームが XYLim で設定した範囲外を通過することもあります。このモードでは、XYLim はパルス動作に対しては適用されません。

XYLIM_STRICT を指定した場合、XYLim で設定した動作範囲は、動作コマンドの目標座標と、動作の開始地点から目標座標までの動作軌跡に対して適用されます。したがって、動作中、アームが XYLim で設定した範囲を超えて動こうとするとエラーになります。このモードでは、XYLim はパルス動作に対しても適用されます。ただし、開始地点が XYLim 範囲外で目標座標が XYLim 範囲内のように XYLim 範囲外から範囲内に動く場合、XYLim 範囲外エラーにならずに動けます。

ロボットと装置との干渉を防ぐために、XYLIM_STRICT でご使用いただくことを推奨します。

EPSON RC+ のコントローラー環境設定により、コントローラー起動時の監視方法のデフォルト値を変更することができます。XYLimMode コマンドで設定された値は、コントローラー再起動まで有効です。コントローラーを再起動した場合、XYLim の監視方法はコントローラー環境設定で指定した監視方法にリセットされます。



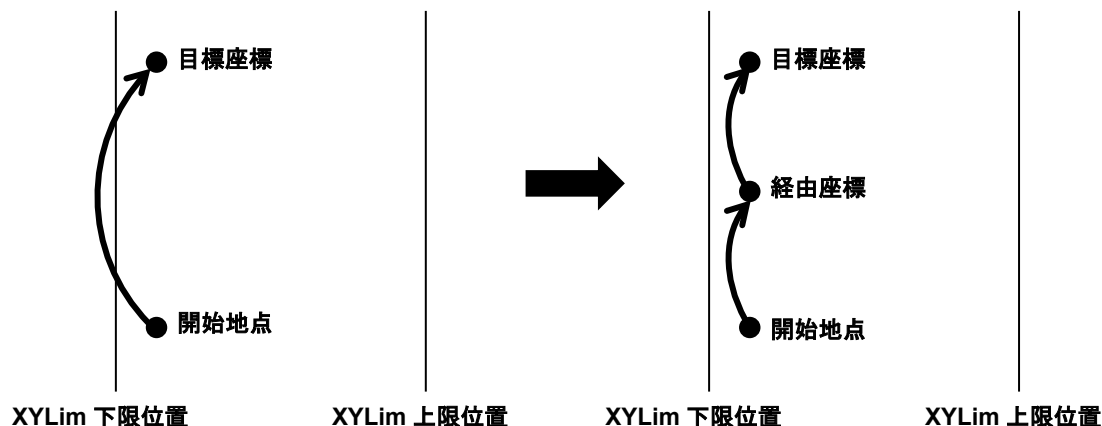
注意

- XYLIM_STANDARD は、アームが XYLim で設定した範囲外を通過することがあります。そのため、XYLim を実際の障害物と余裕をもって設定したり、XYLim の境界付近の動作は低速で軌道を確認して、ロボットと装置とが干渉しないことに注意してください。

起こりやすいエラー

XYLim 境界付近で PTP 動作を実行した場合

Go 命令などの PTP 動作を実行すると、開始地点と目標座標は左下図の矢印のように動きます。そのため、XYLIM_STRICT では、目標座標は範囲内ですが、動作軌跡は範囲外なのでエラーになります。このような場合は、右下図のように開始位置と目標座標の間に経由座標を追加し、動作軌跡が XYLim を超えないようにして回避してください。



過去プロジェクト使った場合

XYLim の監視方法を XYLIM_STRICT にしたコントローラーに、コントローラーFW の Ver 7.5.2.0、7.5.52.0 未満で使っていたプロジェクトを適用すると、動作軌跡が範囲外エラーになる場合があります。その場合、経由座標を追加するなど XYLim を超えないようにプログラムを修正してください。

参照

XYLim

XYLimMode コマンド使用例

下記は XYLimMode を使用する例です。現在位置から P1 までは XYLIM_STANDARD で移動し、P1 から P2 までは XYLIM_STRICT で移動します。

```
Function XYLimMode_sample
  Motor On
  XYLimMode XYLIM_STANDARD
  Go P1                                     ' 目標座標のみに XYLim を適用
  XYLimMode XYLIM_STRICT
  Go P2                                     ' 目標座標と動作軌跡に XYLim を適用
End
```

下記は、XYLimMode をコマンドウィンドウで使用する例です。現在の XYLim の監視方法を表示します。

```
> XYLimMode
1
```

XYLimMode 関数

設定された XYLim の監視方法を取得します。

書式

XYLimMode

戻り値

設定された XYLim の監視方法を返します。

0 = 動作コマンドの目標座標に対して XYLim を適用する。(パルス動作に適用しない)

1 = 動作コマンドの目標座標と動作軌跡に対して XYLim を適用する。(パルス動作にも適用する)

参照

XYLimMode

XYLimMode 関数使用例

下記は XYLimMode 関数の使用例です。変数に XYLim の監視方法を取得し、それを表示するプログラムです。

```
Function XYLimMode_sample
    Integer iVar

    iVar = XYLimMode
    Print iVar

End
```

Appendix A: SPEL+ コマンド 使用条件一覧

コマンドウィンドウ
プログラム
関数

コマンドウィンドウで使えます。
SPEL+プログラム中でステートメントとして使えます。
関数として使えます。

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
A	AbortMotion	○	○	-
	Abs	-	○	○
	Accel	○	○	○
	AccelMax	-	○	○
	AccelR	○	○	○
	AccelS	○	○	○
	Acos	-	○	○
	Agl	-	○	○
	AglToPls	-	○	○
	AIO_In	○	-	○
	AIO_InW	○	-	○
	AIO_Out	○	○	○
	AIO_OutW	○	○	○
	AIO_Set	○	○	○
	AIO_TrackingSet	○	○	-
	AIO_TrackingStart	-	○	-
	AIO_TrackingEnd	-	○	-
	AIO_TrackingOn	○	○	○
	Align	-	○	○
	AlignECP	-	○	○
	And	-	○	-
	AOpen	○	○	-
	Arc	○	○	-
	Arc3	○	○	-
	Arch	○	○	○
	AreaCorrection	○	○	○
	AreaCorrectionClr	○	○	-
	AreaCorrectionDef	○	○	○
	AreaCorrectionInv	○	○	○
	AreaCorrectionOffset	○	○	○
	AreaCorrectionSet	○	○	-
	Arm	○	○	○
	ArmClr	○	○	-
	ArmDef	-	○	○
	ArmSet	○	○	○
	Asc	-	○	○
	ArmCalib	○	○	○
	ArmCalibClr	○	○	-
	ArmCalibDef	-	○	○
	ArmCalibSet	○	○	○
	Asin	-	○	○
	Atan	-	○	○
	Atan2	-	○	○
	ATCLR	○	○	-

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	AtHome	-	○	○
	ATRQ	○	○	○
	AutoLJM	○	○	○
	AvoidSingularity	○	○	○
B	Base	○	○	○
	BClr	-	○	○
	BClr64	-	○	○
	BGo	○	○	-
	BMove	○	○	-
	Boolean	-	○	-
	BOpen	○	○	-
	Box	○	○	○
	BoxClr	○	○	-
	BoxDef	-	○	○
	Brake	○	○(関数のみ)	○
	BSet	-	○	○
	BSet64	-	○	○
	BTst	-	○	○
	BTst64	-	○	○
	Byte	-	○	-
C	Calib	○	○	-
	Call	-	○	-
	CalPls	○	○	○
	ChDir	○	○	-
	ChDisk	○	○	-
	ChDrive	○	○	-
	ChkCom	-	○	○
	ChkNet	-	○	○
	Chr\$	-	○	○
	ClearPoints	○	○	-
	Close	○	○	-
	CloseCom	○	○	-
	CloseDB	○	○	-
	CloseNet	○	○	-
	Cls	○	○	-
	Cnv_AbortTrack	○	○	-
	Cnv_Accel	○	○	○
	Cnv_AccelLim	○	○	○
	Cnv_Adjust	○	○	-
	Cnv_AdjustClear	○	○	-
	Cnv_AdjustGet	○	○	○
	Cnv_AdjustSet	○	○	-
	Cnv_Downstream	○	○	○
	Cnv_Fine	○	○	○
	Cnv_LPulse	-	○	○
	Cnv_Mode	○	○	○
	Cnv_Name\$	-	○	○
	Cnv_Number	-	○	○
	Cnv_OffsetAngle	○	○	○
	Cnv_Point	-	○	○
	Cnv_PosErr	-	○	○

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	Cnv_PosErrOffset	-	-	○
	Cnv_Pulse	-	-	○
	Cnv_QueueAdd	○	○	○
	Cnv_QueueGet	-	-	○
	Cnv_QueueLen	-	-	○
	Cnv_QueueList	○	○	-
	Cnv_QueueMove	○	○	○
	Cnv_QueueReject	○	○	○
	Cnv_QueueRemove	○	○	○
	Cnv_QueueUserData	○	○	○
	Cnv_RobotConveyor	-	-	○
	Cnv_Speed	-	-	○
	Cnv_Trigger	○	○	○
	Cnv_Upstream	○	○	○
	CollisionDetect	○	○	○
	Cont	○	-	○
	Copy	○	○	○
	Cos	-	-	○
	CP	○	○	○
	Ctr	-	-	○
	CTReset	○	○	○
	CtrlDev	-	-	○
	CtrlInfo	-	-	○
	CurDir\$	-	-	○
	CurDisk\$	-	-	○
	CurDrive\$	-	-	○
	CurPos	-	-	○
	Curve	○	○	○
	CVMove	○	○	○
	CP_Offset	○	○	○
	CR	○	○	○
	CS	○	○	○
	CT	○	○	○
	CU	○	○	○
	CV	○	○	○
	CW	○	○	○
	CX	○	○	○
	CY	○	○	○
	CZ	○	○	○
D	Date	○	○	○
	Date\$	-	-	○
	Declare	-	-	○
	DegToRad	-	-	○
	Del	○	○	○
	DeleteDB	○	○	○
	DiffPoint	○	○	○
	DispDev	○	○	○
	Dist	-	-	○
	Do...Loop	-	-	○
	Double	-	-	○
E	ECP	○	○	○

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	ECPClr	○	○	-
	ECPDef	-	○	○
	ECPSet	○	○	○
	ElapsedTime	-	○	○
	Elbow	○	○	○
	Eof	-	○	○
	Era	-	○	○
	EResume	○	○	-
	Erf\$	-	○	○
	Erl	-	○	○
	Err	-	○	○
	Errb	○	○	○
	ErrMsg\$	-	○	○
	Error	○	○	-
	ErrorOn	-	○	○
	Ert	-	○	○
	EStopOn	-	○	○
	Eval	-	○	○
	Exit	-	○	-
	ExportPoints	○	○	-
F	FbusIO_GetBusStatus	-	○	○
	FbusIO_GetDeviceStatus	-	○	○
	FbusIO_SendMsg	○	○	-
	FileDataTime\$	-	○	○
	FileExists	-	○	○
	FileLen	-	○	○
	Find	○	○	-
	FindPos	-	○	○
	Fine	○	○	○
	FineDist	○	○	○
	FineStatus	○	○	○
	Fix	-	○	○
	Flush	○	○	-
	FmtStr	○	○	-
	FmtStr\$	-	○	○
	FolderExists	-	○	○
	For...Next	-	○	-
	Force_Calibrate	○	○	-
	Force_ClearTrigger	○	○	-
	Force_GetForce	-	○	○
	Force_GetForces	○	○	-
	Force_Sensor	○	○	○
	Force_SetTrigger	○	○	-
	FreeFile	-	○	○
	Function...Fend	-	○	-
G	GClose	○	○	-
	GetCurrentUser\$	-	○	○
	GetRobotInsideBox	-	○	○
	GetRobotInsidePlane	-	○	○
	GGet	○	○	-
	Global	-	○	-

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	Go	○	○	-
	Gosub...Return	-	○	-
	Goto	-	○	-
	GSet	○	○	-
	GShow	○	○	-
	GShowDialog	-	○	○
H	Halt	-	○	-
	Hand	○	○	○
	HealthCalcPeriod	○	○	○
	HealthCtrlAlarmOn	○	○	○
	HealthCtrlInfo	○	○	○
	HealthCtrlRateOffset	○	○	-
	HealthCtrlReset	○	○	-
	HealthCtrlWarningEnable	○	○	○
	HealthRateCtrlInfo	○	○	○
	HealthRateRBInfo	○	○	○
	HealthRBAAlarmOn	○	○	○
	HealthRBAnalysis	○	○	○
	HealthRBDistance	○	○	○
	HealthRBInfo	○	○	○
	HealthRBRateOffset	○	○	-
	HealthRBReset	○	○	-
	HealthRBSpeed	○	○	○
	HealthRBStart	○	○	-
	HealthRBStop	○	○	-
	HealthRBTRQ	○	○	○
	HealthRBWarningEnable	○	○	○
	Here	○	○	○
	Hex\$	-	○	○
	Hofs	○	○	○
	HofsJointAccuracy	○	○	-
	Home	○	○	-
	HomeClr	○	○	-
	HomeDef	-	○	○
	HomeSet	○	○	○
	Hordr	○	○	○
	Hour	○	○	○
I	If...Then..Else...EndIf	-	○	-
	ImportPoints	○	○	-
	In	-	○	○
	InBCD	-	○	○
	Inertia	○	○	○
	InPos	-	○	○
	Input	○	○	-
	Input #	○	○	-
	InputBox	○	○	-
	InReal	-	○	○
	InsideBox	-	○	○
	InsidePlane	-	○	○
	InStr	-	○	○
	Int	-	○	○

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	Int32	-	○	-
	Integer	-	○	-
	InW	-	○	○
	IODef	-	○	○
	IOLabel\$	-	○	○
	IONumber	-	○	○
J	J1Angle	○	○	○
	J4Angle	○	○	○
	J1Flag	○	○	○
	J2Flag	○	○	○
	J4Flag	○	○	○
	J6Flag	○	○	○
	JA	-	○	○
	Joint	○	○	-
	JointAccuracy	○	○	○
	JRange	○	○	○
	JS	-	○	○
	JT	-	○	○
	JTran	○	○	-
	Jump	○	○	-
	Jump3	○	○	-
	Jump3CP	○	○	-
	JumpTLZ	○	○	-
L	LatchEnable	○	○	-
	LatchPos	-	○	○
	LatchState	-	○	○
	LCase\$	-	○	○
	Left\$	-	○	○
	Len	-	○	○
	LimitTorque	○	○	○
	LimitTorqueLP	○	○	○
	LimitTorqueStop	○	○	○
	LimitTorqueStopLP	○	○	○
	LimZ	○	○	○
	LimZMargin	○	○	○
	Line Input	○	○	-
	Line Input #	○	○	-
	LJM	-	○	○
	LoadPoints	○	○	-
	Local	○	○	○
	LocalClr	○	○	-
	LocalDef	-	○	○
	Lof	-	○	○
	LogIn	-	○	○
	Long	-	○	-
	LSet\$	-	○	○
	LShift	-	○	○
	LShift64	-	○	○
	LTrim\$	-	○	○
M	Mask	-	○	-
	MCal	○	○	-

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	MCalComplete	-	-	○
	MCordr	○	○	○
	MemIn	-	-	○
	MemInW	-	-	○
	MemOff	○	○	○
	MemOn	○	○	-
	MemOut	○	○	-
	MemOutW	○	○	-
	MemSw	-	-	○
	MHour	-	-	○
	Mid\$	-	-	○
	MkDir	○	○	○
	Mod	-	-	○
	Motor	○	○	○
	Move	○	○	○
	MsgBox	○	○	○
	MyTask	-	-	○
N	Next	-	-	○
	Not	-	-	○
O	Off	○	○	○
	OLAccel	○	○	○
	OLRate	○	○	○
	On	○	○	-
	OnErr	-	-	○
	OpBCD	○	○	○
	OpenCom	○	○	○
	OpenDB	○	○	-
	OpenNet	○	○	○
	Oport	-	-	○
	Or	-	-	○
	Out	○	○	○
	OutReal	○	○	○
	OutW	○	○	○
P	P#	○	○	○
	PAGl	-	-	○
	Pallet	○	○	○
	PalletClr	○	○	○
	ParseStr	○	○	○
	Pass	○	○	-
	Pause	-	-	○
	PauseOn	-	-	○
	PDescription	○	○	○
	PDescription\$	○	○	-
	PDef	-	-	○
	PDel	○	○	○
	PerformMode	○	○	○
	PG_FastStop	○	○	-
	PG_LSpeed	○	○	○
	PG_Scan	○	○	-
	PG_SlowStop	○	○	-
	PLabel	○	○	-

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	PLabel\$	-	○	○
	Plane	○	○	○
	PlaneClr	○	○	-
	PlaneDef	-	○	○
	PList	○	○	-
	PLocal	○	○	○
	Pls	-	○	○
	PNumber	-	○	○
	PosFound	-	○	○
	Power	○	○	○
	PPls	-	○	○
	Preserve	-	○	-
	Print	○	○	-
	Print #	○	○	-
	PTCLR	○	○	-
	PTPBoost	○	○	○
	PTPBoostOK	-	○	○
	PTPTime	-	○	○
	PTran	○	○	-
	PTRQ	○	○	○
	Pulse	○	○	○
Q	QP	○	○	-
	QPDcelR	○	○	○
	QPDcelS	○	○	○
	Quit	-	○	-
R	RadToDeg	-	○	○
	Randmize	○	○	-
	Range	○	○	-
	Read	○	○	-
	ReadBin	○	○	-
	Real	-	○	-
	RealAccel	-	○	○
	RealPls	-	○	○
	RealPos	-	○	○
	RealTorque	-	○	○
	Recover	○	○	○
	RecoverPos	-	○	○
	Redim	○	○	-
	Rename	○	○	-
	RenDir	○	○	-
	Reset	○	○	-
	ResetElapsedTime	○	○	-
	Restart	○	○	-
	Resume	-	○	-
	Return	-	○	-
	Right\$	-	○	○
	RmDir	○	○	-
	Rnd	-	○	○
	Robot	○	○	○
	RobotInfo	-	○	○
	RobotInfo\$	-	○	○

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	RobotModel\$	-	○	○
	RobotName\$	-	○	○
	RobotSerial\$	-	○	○
	RobotType	-	○	○
	ROpen	○	○	-
	ROTK	○	○	○
	RSet\$	-	○	○
	RShift64	-	○	○
	RShift	-	○	○
	RTrim\$	-	○	○
	RunDialog	-	○	-
S	SafetyOn	-	○	○
	SavePoints	○	○	-
	Seek	○	○	-
	Select...Send	-	○	-
	SelectDB	○	○	○
	Sense	○	○	-
	SetCom	○	○	-
	SetIn	○	○	-
	SetInReal	○	○	-
	SetInW	○	○	-
	SetLatch	○	○	-
	SetNet	○	○	-
	SetSw	○	○	-
	SF_GetParam	○	○	○
	SF_GetParam\$	○	○	○
	SF_GetStatus	○	○	○
	SF_LimitSpeedS	○	○	○
	SF_LimitSpeedSEnable	○	○	○
	SF_RealSpeedS	○	○	○
	SF_PeakSpeedS	○	○	○
	SF_PeakSpeedSClear	○	○	-
	SFree	○	○	○
	Sgn	-	○	○
	Short	-	○	-
	Shutdown	○	○	○
	Signal	○	○	-
	SimGet	-	○	-
	SimSet	○	○	-
	Sin	-	○	○
	SingularityAngle	○	○	○
	SingularityDist	○	○	○
	SingularitySpeed	○	○	○
	SLock	○	○	-
	SoftCP	○	○	○
	Space\$	-	○	○
	Speed	○	○	○
	SpeedFactor	○	○	○
	SpeedR	○	○	○
	SpeedS	○	○	○
	SPELCom_Event	○	○	-

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	Sqr	-	○	○
	ST	-	○	○
	StartMain	-	○	-
	Stat	-	○	○
	Str\$	-	○	○
	String	-	○	-
	Sw	-	○	○
	SyncLock	-	○	-
	SyncUnlock	-	○	-
	SyncRobots	○	○	○
	SysConfig	○	-	-
	SysErr	-	○	○
T	Tab\$	-	○	○
	Tan	-	○	○
	TargetOK	-	○	○
	TaskDone	-	○	○
	TaskInfo	-	○	○
	TaskInfo\$	-	○	○
	TaskState	○	○	○
	TaskWait	○	○	-
	TC	○	○	-
	TCLim	○	○	○
	TCPSpeed	-	○	○
	TCSpeed	○	○	○
	TeachOn	-	○	○
	TGo	○	○	-
	Till	○	○	-
	TillOn	-	○	○
	Time	○	○	○
	Time\$	-	○	○
	TLClr	○	○	-
	TLDef	-	○	○
	TLSet	○	○	○
	TMOut	○	○	-
	TMove	○	○	-
	Tmr	-	○	○
	TmReset	○	○	-
	Toff	○	○	-
	Ton	○	○	-
	Tool	○	○	○
	Trap	-	○	-
	Trim\$	-	○	○
	TW	-	○	○
U	UBound	-	○	○
	UByte	-	○	-
	UCase\$	-	○	○
	UInt32	-	○	-
	UOpen	○	○	-
	UpdateDB	○	○	-
	UShort	-	○	-
V	Val	-	○	○

命令	コマンドウィンドウ		プログラム	関数
	RC+	TP3		
	VCal	○	○	○
	VCalPoints	○	○	○
	VClIs	-	-	○
	VCreateCalibration	-	-	○
	VCreateObject	-	-	○
	VCreateSequence	-	-	○
	VDefArm	-	-	○
	VDefGetMotionRange	○	-	○
	VDefLocal	-	-	○
	VDefSetMotionRange	○	-	○
	VDefTool	-	-	○
	VDeleteCalibration	-	-	○
	VDeleteObject	-	-	○
	VDeleteSequence	-	-	○
	VGet	-	-	○
	VGoCenter	-	-	○
	VisCalib	-	-	-
	VisCalInfo	-	-	-
	VisCalLoad	-	-	-
	VisCalSave	-	-	-
	VisTrans	-	-	-
	VLoad	-	-	○
	VLoadModel	-	-	○
	VRun	-	-	○
	VSave	-	-	○
	VSaveImage	-	-	○
	VSaveModel	-	-	○
	VSD	○	○	○
	VSet	-	-	○
	VShowModel	○	-	○
	VStatsReset	-	-	○
	VStatsResetAll	-	-	○
	VStatsSave	-	-	○
	VStatsShow	○	○	○
	VTech	-	-	○
	VTrain	○	-	○
	VxCalib	○	○	○
	VxCalDelete	○	○	○
	VxCalLoad	○	○	○
	VxCalInfo	-	-	○
	VxCalSave	○	○	○
	VxTrans	-	-	○
W	Wait	○	○	○
	WaitNet	○	○	○
	WaitPos	○	○	○
	WaitSig	○	○	○
	Weight	○	○	○
	Where	○	○	-
	WindowStatus	-	-	○
	WorkQue_ Add	○	○	○
	WorkQue_ AutoRemove	○	○	○
	WorkQue_ Get	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3		
	WorkQue_Len	○	○	○	○
	WorkQue_List	○	○	-	-
	WorkQue_Reject	○	○	○	○
	WorkQue_Remove	○	○	○	-
	WorkQue_Sort	○	○	○	○
	WorkQue_UserData	○	○	○	○
	WOpen	○	○	○	-
	Wrist	○	○	○	○
	Write	○	○	○	-
	WriteBin	○	○	○	-
X	Xor	-	-	○	-
	Xqt	-	-	○	-
	XY	-	-	○	○
	XYLim	○	○	○	○
	XYLimClr	○	○	○	-
	XYLimDef	-	-	○	○
	XYLimMode	○	○	○	○

Appendix B: 互換性に関する注意

B-1: EPSON RC+ 6.0 からの互換性に関する注意

概要

すでに、RC620コントローラーで、EPSON RC+ 6.0をご利用の方が、RC700シリーズコントローラーでEPSON RC+ 7.0を使う場合に読んでいただきたい内容をまとめました。

EPSON RC+ 7.0を使用する場合は、ハードウェアの違い、対応マニピュレーターや関節数の違い、ソフトウェアの実行環境の違いなどにより、EPSON RC+ 6.0とは異なっている部分があります。あらかじめその内容をご理解いただき、安全にロボットをご使用してください。

EPSON RC+ 7.0は、できるだけ今までの製品と互換性を保ちながら改善し、先進のソフトウェア技術を取り入れて設計しています。しかしながら、ロボットコントローラーに特化し、より使いやすくするために、これまでのEPSON RC+ 6.0との互換性を全て満たしてはいない部分や、削除された部分があります。

以下の互換性に関する記述は、EPSON RC+ 6.0を元に、EPSON RC+ 7.0と比較をしています。

全般にわたる違い

EPSON RC+ 6.0と、EPSON RC+ 7.0の全般にわたる違いは下記のとおりです。

項目	EPSON RC+ 7.0	EPSON RC+ 6.0
タスク数	最大 32 タスク (バックグラウンドタスクは最大 16 タスク)	最大 32 タスク (バックグラウンドタスクは最大 16 タスク)
タスクの種類	NoPause タスクの指定が可 NoEmgAbort タスクの指定が可 バックグラウンドタスクの指定が可	NoPause タスクの指定が可 NoEmgAbort タスクの指定が可 バックグラウンドタスクの指定が可
TRAP ERROR などの特殊な TRAP	サポートされています	サポートされています
TRAPトラップ番号で起動されるタスク	専用のタスク番号	専用のタスク番号
マルチマニピュレーター	サポートされています	サポートされています
ロボット番号	1~16	1~16
Real 型の有効桁数	6 桁	6 桁
Double 型の有効桁数	14 桁	14 桁
配列要素数	文字列変数以外 ローカル変数 2,000 グローバル変数 100,000 モジュール変数 100,000 バックアップ変数 4,000	文字列変数以外 ローカル変数 2,000 グローバル変数 100,000 モジュール変数 100,000 バックアップ変数 4,000
	文字列変数 ローカル変数 200 グローバル変数 10,000 モジュール変数 10,000 バックアップ変数 400	文字列変数 ローカル変数 200 グローバル変数 10,000 モジュール変数 10,000 バックアップ変数 400

項目	EPSON RC+ 7.0	EPSON RC+ 6.0
デバイス番号	21:PC 22:リモート 24:TP 20:TP3	21:PC 22:リモート 24:TP 28:LCD
コントロールデバイス	リモート I/O PC リモート COM リモート Ethernet TP3	リモート I/O PC
タイマー番号の範囲	0~63	0~63
プログラム容量	8MB	8MB
SyncLock, SyncUnlock 用の シグナル番号の範囲	0~63	0~63
WaitSig, Signal 用の シグナル番号の範囲	0~63	0~63
メモリー I/O 点数	1024	1024
I/O ポートの番号	EPSON RC+6.0 と共通	
イーサネットのポート番号	201~216	201~216
リモート入出力割りあて	標準で機能割りあて済	デフォルト設定なし
RS-232C 通信ポート番号 SEPL+制御部	1~8, 1001~1008	1~8, 1001, 1002
RS-232C 通信ポートの OpenCom 実行	必須	必須
ファイルへの入出力	サポートされています	サポートされています
ファイルアクセスのファイル番号	30~63	30~63
データベースアクセス番号	501~508	501~508
VisionGuide	スマートカメラタイプ フレームグラバタイプ	スマートカメラタイプ フレームグラバタイプ
コンベヤトラッキング	サポートされています	サポートされています
PG ロボット	サポートされています	サポートされています
OCR	サポートされています	サポートされています
セキュリティー	サポートされています	サポートされています
VB Guide 6.0 (RC+ API 7.0)	サポートされています	サポートされています
フィールドバス I/O の扱い	通常の I/O 命令を使用	通常の I/O 命令を使用
フィールドバス マスター	応答性が保証されません	応答性が保証されません
フィールドバス スレーブ	応答性が保証されています	応答性が保証されています
GUI Builder	サポートされています	サポートされています
エラー番号	EPSON RC+6.0 と共通	

命令の互換性一覧

- + 機能拡張、機能変更されているが上位互換が保たれている命令
- 変更がない命令
- ! 機能変更または、文法変更があり注意が必要な命令
- !! 大幅な変更があり注意が必要な命令
- × 削除された命令

	命令	互換性	備考
A	Abs 関数	–	
	Accel	–	
	Accel 関数	–	
	AccelMax 関数	–	
	AccelR	–	
	AccelR 関数	–	
	AccelS	–	
	AccelS 関数	–	
	Acos 関数	–	
	AglToPls 関数	–	
	Agl 関数	–	
	AlignECP 関数	–	
	Align 関数	–	
	And	–	
	Arc	–	
	Arc3	–	
	Arch	–	
	Arch 関数	–	
	Arm	–	
	ArmClr	–	
	ArmDef 関数	–	
	ArmSet	–	
	ArmSet 関数	–	
	Arm 関数	–	
	Asc 関数	–	
	Asin 関数	–	
	Atan2 関数	–	
	Atan 関数	–	
	ATCLR	–	
	ATRQ	–	
	ATRQ 関数	–	
B	Base	–	
	Base 関数	–	
	BClr 関数	–	
	BGo	+	動作モード指定を追加
	BMove	–	
	Boolean	–	
	Box	+	リモート出力論理設定の指定を追加
	Box 関数	–	

命令		互換性	備考
	BoxClr 関数	—	
	BoxDef 関数	—	
	Brake	—	
	Brake 関数	—	
	BSet 関数	—	
	BTst 関数	—	
	Byte	—	
C	Call	—	
	ChkCom 関数	—	
	ChkNet 関数	—	
	Chr\$関数	—	
	ClearPoints	—	
	CloseCom	—	
	CloseNet	—	
	Cls	—	
	Cos 関数	—	
	CP	—	
	CP 関数	—	
	CTReset	—	
	CtrlDev 関数	—	
	CtrlInfo 関数	—	
	Ctr 関数	—	
	CurPos 関数	—	
	Curve	—	
	CVMove	—	
	CX~CW	—	
	CX~CW 関数	—	
D	Date	—	
	Date\$関数	—	
	DegToRad 関数	—	
	DispDev	—	
	DispDev 関数	—	
	Dist 関数	—	
	Do...Loop	—	
	Double	—	
E	ECP	—	
	ECPClr	—	
	EcpDef 関数	—	
	ECPSet	—	
	ECPSet 関数	—	
	ECP 関数	—	
	Elbow	—	
	Elbow 関数	—	
	Era 関数	—	
	EResume	—	
	Erf\$関数	—	

命令		互換性	備考
	Erl 関数	—	
	ErrMsg\$関数	—	
	Error	—	
	ErrorOn 関数	—	
	Err 関数	—	
	Ert 関数	—	
	EStopOn 関数	—	
	Exit	—	
	Find	—	
	FindPos 関数	—	
	Fine	—	
	Fine 関数	—	
	Fix 関数	—	
	FmtStr\$	—	
	For...Next	—	
	Function...Fend	—	
G	Global	—	
	Go	+	動作モード指定を追加
	Gosub...Return	—	
	Goto	—	
H	Halt	—	
	Hand	—	
	Hand 関数	—	
	Here	—	
	Here 関数	—	
	Hex\$関数	—	
	Home	—	
	HomeClr	—	
	HomeDef 関数	—	
	HomeSet	—	
	HomeSet 関数	—	
	HOrdr	—	
	HOrdr 関数	—	
	Hour	—	
	Hour 関数	—	
I	If...EndIf	—	
	In 関数	—	
	InBCD 関数	—	
	Inertia	—	
	Inertia 関数	—	
	InPos 関数	—	
	Input	—	
	Input#	—	
	InsideBox 関数	—	
	InsidePlane 関数	—	
	InStr 関数	—	

	命令	互換性	備考
	Integer	—	
	Int 関数	—	
	InW 関数	—	
	IOLabel\$関数	—	
	IONumber 関数	—	
J	J1Flag	—	
	J1Flag 関数	—	
	J2Flag	—	
	J2Flag 関数	—	
	J4Flag	—	
	J4Flag 関数	—	
	J6Flag	—	
	J6Flag 関数	—	
	JA 関数	—	
	Joint	—	
	JRange	—	
	JRange 関数	—	
	JS 関数	—	
	JTran	—	
	JT 関数	—	
	Jump	+	動作モード指定を追加
	Jump3	—	
	Jump3CP	—	
L	LCase\$関数	—	
	Left\$関数	—	
	Len 関数	—	
	LimZ	—	
	LimZ 関数	—	
	Line Input	—	
	Line Input#	—	
	LJM 関数	—	
	LoadPoints	—	
	Local	—	
	LocalClr	—	
	LocalDef 関数	—	
	Local 関数	—	
	Lof 関数	—	
	Long	—	
	LSet\$関数	—	
	LShift 関数	—	
	LTrim\$関数	—	
M	Mask	—	
	MemInW 関数	—	
	MemIn 関数	—	
	MemOff	—	
	MemOn	—	

	命令	互換性	備考
	MemOut	—	
	MemOutW	—	
	MemSw 関数	—	
	Mid\$関数	—	
	Mod	—	
	Motor	—	
	Motor 関数	—	
	Move	—	
	MyTask 関数	—	
N	Not	—	
O	Off	—	
	OLAccel	—	
	OLAccel 関数	—	
	OLRate	—	
	OLRate 関数	—	
	On	—	
	OnErr	—	
	OpBCD	—	
	OpenCom	—	
	OpenNet	—	
	Oport 関数	—	
	Or	—	
	Out	—	
	OutW	—	
	OutW 関数	—	
	Out 関数	—	
P	P#	—	
	PAGl 関数	—	
	Pallet	+	座標値指定の追加
	Pallet 関数	—	
	ParseStr	—	
	ParseStr 関数	—	
	Pass	—	
	Pause	—	
	PauseOn 関数	—	
	PDef 関数	—	
	PDel	—	
	PLabel	—	
	PLabel\$関数	—	
	Plane	—	
	Plane 関数	—	
	PlaneClr	—	
	PlaneDef 関数	—	
	PList	—	
	PLocal	—	
	PLocal 関数	—	

命令		互換性	備考
	Pls 関数	—	
	PNumber 関数	—	
	PosFound 関数	—	
	Power	—	
	Power 関数	—	
	PPls 関数	—	
	Print	—	
	Print#	—	
	PTCLR	—	
	PTPBoost	—	
	PTPBoostOK 関数	—	
	PTPBoost 関数	—	
	PTPTime 関数	—	
	PTran	—	
	PTRQ	—	
	PTRQ 関数	—	
	Pulse	—	
	Pulse 関数	—	
Q	QP	—	
	Quit	—	
R	RadToDeg 関数	—	
	Randmize	—	
	Range	—	
	Read	—	
	ReadBin	—	
	Real	—	
	RealPls 関数	—	
	RealPos 関数	—	
	RealTorque	—	
	Redim	—	
	Reset	—	
	Resume	—	
	Return	—	
	RobotInfo 関数	—	
	RobotInfo\$関数	—	
	RobotModel\$関数	—	
	RobotName\$関数	—	
	RobotSerial\$関数	—	
	RobotType 関数	—	
	RSet\$関数	—	
	RShift 関数	—	
	RTrim\$関数	—	
S	SafetyOn 関数	—	
	SavePoints	—	
	Select...Send	—	
	Sense	—	

命令		互換性	備考
	SetCom	—	
	SetInW	—	
	SetIn	—	
	SetNet	—	
	SetSw	—	
	SFree	—	
	SFree 関数	—	
	Sgn 関数	—	
	Signal	—	
	Sin 関数	—	
	SLock	—	
	SoftCP	—	
	SoftCP 関数	—	
	Space\$関数	—	
	Speed	—	
	SpeedR	—	
	SpeedR 関数	—	
	SpeedS	—	
	SpeedS 関数	—	
	Speed 関数	—	
	SPELCom_Event	—	
	Sqr 関数	—	
	Stat 関数	—	
	Str\$関数	—	
	String	—	
	Sw 関数	—	
	SyncLock	—	
	SyncUnlock	—	
	SysConfig	—	
	SysErr 関数	—	
T	Tab\$関数	—	
	Tan 関数	—	
	TargetOK 関数	—	
	TaskDone 関数	—	
	TaskInfo 関数	—	
	TaskInfo\$関数	—	
	TaskState	—	
	TaskState 関数	—	
	TaskWait	—	
	TC	—	
	TCLim	—	
	TCLim 関数	—	
	TCSpeed	—	
	TCSpeed 関数	—	
	TGo	+	動作モード指定を追加
	TillOn 関数	—	

命令		互換性	備考
	Time	—	
	Time 関数	—	
	Time\$関数	—	
	TLClr	—	
	TlDef 関数	—	
	TLSet	—	
	TLSet 関数	—	
	TMOut	—	
	TMove	—	
	TmReset	—	
	Tmr 関数	—	
	Toff	—	
	Ton	—	
	Tool	—	
	Tool 関数	—	
	Trap	—	
	Trim\$関数	—	
	Tw 関数	—	
U	UBound 関数	—	
	UCase\$関数	—	
V	Val 関数	—	
W	Wait	—	
	WaitNet	—	
	WaitPos	—	
	WaitSig	—	
	Weight	—	
	Weight 関数	—	
	Where	—	
	Wrist	—	
	Wrist 関数	—	
	Write	—	
	WriteBin	—	
X	Xor	—	
	Xqt	—	
	XY 関数	—	
	XYLim	—	
	XYLim 関数	—	
	XYLimClr	—	
	XYLimDef	—	
	XYLimDef 関数	—	

B-2: EPSON RC+ 5.0 からの互換性に関する注意

概要

すでに、RC180コントローラーで、EPSON RC+ 5.0をご利用の方が、RC700シリーズ、RC90シリーズコントローラーでEPSON RC+ 7.0を使う場合に読んでいただきたい内容をまとめました。

EPSON RC+ 7.0を使用する場合は、ハードウェアの違い、対応マニピュレーターや関節数の違い、ソフトウェアの実行環境の違いなどにより、EPSON RC+ 5.0とは異なっている部分があります。あらかじめその内容をご理解いただき、安全にロボットをご使用してください。

EPSON RC+ 7.0は、できるだけ今までの製品と互換性を保ちながら改善し、先進のソフトウェア技術を取り入れて設計しています。しかしながら、ロボットコントローラーに特化し、より使いやすくするために、これまでのEPSON RC+ 5.0との互換性を全て満たしてはいない部分や、削除された部分があります。

以下の互換性に関する記述は、EPSON RC+ 5.0を元に、EPSON RC+ 7.0と比較をしています。

全般にわたる違い

EPSON RC+ 5.0と、EPSON RC+ 7.0の全般にわたる違いは下記のとおりです。

項目	EPSON RC+ 7.0	EPSON RC+ 5.0
タスク数	最大 32 タスク (バックグラウンドタスクは最大 16 タスク)	最大 16 タスク
タスクの種類	NoPause タスクの指定が可 NoEmgAbort タスクの指定が可 バックグラウンドタスクの指定が可	NoPause タスクの指定が可 NoEmgAbort タスクの指定が可
TRAP ERROR などの特殊な TRAP	サポートされています	サポートされません
TRAPトラップ番号で起動されるタスク	専用のタスク番号	専用のタスク番号
マルチマニピュレーター	サポートされています	サポートされません
ロボット番号	1~16	1
Real 型の有効桁数	6 桁	6 桁
Double 型の有効桁数	14 桁	14 桁
配列要素数	文字列変数以外 ローカル変数 2,000 グローバル変数 100,000 モジュール変数 100,000 バックアップ変数 4,000	文字列変数以外 ローカル変数 1,000 グローバル変数 10,000 モジュール変数 10,000 バックアップ変数 1,000
	文字列変数 ローカル変数 200 グローバル変数 10,000 モジュール変数 10,000 バックアップ変数 400	文字列変数 ローカル変数 100 グローバル変数 1,000 モジュール変数 1,000 バックアップ変数 100
デバイス番号	21:PC 22:リモート 24:TP 20:TP3	21:PC 22:リモート 23:OP 24:TP

項目	EPSON RC+ 7.0	EPSON RC+ 5.0
コントロールデバイス	リモート I/O PC リモート COM リモート Ethernet TP3	リモート I/O PC OP1 リモート Ethernet
タイマー番号の範囲	0~63	0~15
プログラム容量	8MB	4MB
SyncLock, SyncUnlock 用の シグナル番号の範囲	0~63	0~15
WaitSig, Signal 用の シグナル番号の範囲	0~63	0~5
メモリー I/O 点数	1024	256
I/O ポートの番号	EPSON RC+5.0 と共通	
イーサネットのポート番号	201~216	201~208
リモート入出力割りあて	標準で機能割りあて済	標準で機能割りあて済
RS-232C 通信ポート番号 SEPL+制御部	1~8, 1001~1008	1~8
RS-232C 通信ポートの OpenCom 実行	必須	必須
ファイルへの入出力	サポートされています	サポートされません
ファイルアクセスのファイル番号	30~63	サポートされません
データベースアクセス番号	501~508	サポートされません
VisionGuide	スマートカメラタイプ フレームグラバタイプ	スマートカメラタイプ
コンベヤートラッキング	サポートされています	サポートされません
PG ロボット	サポートされています	サポートされません
OCR	サポートされています	サポートされません
セキュリティ	サポートされています	サポートされません
VBGuide 5.0 (RC+ API 7.0)	サポートされています	Lite 版がサポートされています
フィールドバス I/O の扱い	通常の I/O 命令を使用	通常の I/O 命令を使用
フィールドバス マスター	応答性が保証されません	サポートされません
フィールドバス スレーブ	応答性が保証されています	応答性が保証されています
GUI Builder	サポートされています	サポートされません
エラー番号	EPSON RC+5.0 と共通	

命令の互換性一覧

- + 機能拡張、機能変更されているが上位互換が保たれている命令
- 変更がない命令
- ! 機能変更または、文法変更があり注意が必要な命令
- !! 大幅な変更があり注意が必要な命令
- × 削除された命令

命令		互換性	備考
A	Abs 関数	—	
	Accel	—	
	Accel 関数	—	
	AccelMax 関数	—	
	AccelR	—	
	AccelR 関数	—	
	AccelS	—	
	AccelS 関数	—	
	Acos 関数	—	
	AglToPls 関数	—	
	Agl 関数	—	
	AlignECP 関数	—	
	Align 関数	—	
	And	—	
	Arc	—	
	Arc3	—	
	Arch	—	
	Arch 関数	—	
	Arm	—	
	ArmClr	—	
	ArmDef 関数	—	
	ArmSet	—	
	ArmSet 関数	—	
	Arm 関数	—	
	Asc 関数	—	
	Asin 関数	—	
	Atan2 関数	—	
	Atan 関数	—	
	ATCLR	—	
	ATRQ	—	
	ATRQ 関数	—	
B	Base	—	
	Base 関数	—	
	BClr 関数	—	
	BGo	+	動作モード指定を追加
	BMove	—	
	Boolean	—	
	Box	+	ロボット番号の指定を追加
	Box 関数	+	ロボット番号の指定を追加

命令		互換性	備考
	BoxClr 関数	+	ロボット番号の指定を追加
	BoxDef 関数	+	ロボット番号の指定を追加
	Brake	—	
	Brake 関数	—	
	BSet 関数	—	
	BTst 関数	—	
	Byte	—	
C	Call	+	外部ファンクション呼び出し
	ChkCom 関数	—	
	ChkNet 関数	—	
	Chr\$関数	—	
	ClearPoints	—	
	CloseCom	—	
	CloseNet	—	
	Cls	—	
	Cos 関数	—	
	CP	—	
	CP 関数	—	
	CTReset	—	
	CtrlDev 関数	!	コントローラーデバイス変更
	CtrlInfo 関数	—	
	Ctr 関数	—	
	CurPos 関数	—	
	Curve	—	
	CVMove	—	
	CX~CW	+	CR, CS, CT 追加
	CX~CW 関数	+	CR, CS, CT 追加
D	Date	!	表示のみ
	Date\$関数	—	
	DegToRad 関数	—	
	DispDev	—	
	DispDev 関数	—	
	Dist 関数	—	
	Do...Loop	—	
	Double	—	
E	ECP	—	
	ECPClr	—	
	EcpDef 関数	—	
	ECPSet	—	
	ECPSet 関数	—	
	ECP 関数	—	
	ElapsedTime 関数	—	
	Elbow	—	
	Elbow 関数	—	
	Era 関数	—	
	EResume	—	

命令		互換性	備考
	Erf\$関数	—	
	Erl 関数	—	
	ErrMsg\$関数	—	
	Error	—	
	ErrorOn 関数	—	
	Err 関数	—	
	Ert 関数	—	
	EStopOn 関数	—	
	Exit	—	
	Find	—	
	FindPos 関数	—	
	Fine	—	
	Fine 関数	—	
	Fix 関数	—	
	FmtStr\$	—	
	For...Next	—	
	Function...Fend	—	
G	Global	—	
	Go	+	動作モード指定を追加
	Gosub...Return	—	
	Goto	—	
H	Halt	—	
	Hand	—	
	Hand 関数	—	
	Here	—	
	Here 関数	—	
	Hex\$関数	—	
	Home	—	
	HomeClr	—	
	HomeDef 関数	—	
	HomeSet	—	
	HomeSet 関数	—	
	HOrdr	—	
	HOrdr 関数	—	
	Hour	—	
	Hour 関数	—	
I	If...EndIf	—	
	In 関数	—	
	InBCD 関数	—	
	Inertia	—	
	Inertia 関数	—	
	InPos 関数	—	
	Input	—	
	Input#	+	デバイス番号追加
	InsideBox 関数	!	ロボット番号、All の指定を追加、Wait 待ちできない
	InsidePlane 関数	!	ロボット番号、All の指定を追加、Wait 待ちできない

	命令	互換性	備考
	InStr 関数	—	
	Integer	—	
	Int 関数	—	
	InW 関数	—	
	IOLabel\$関数	—	
	IONumber 関数	—	
J	J1Flag	—	
	J1Flag 関数	—	
	J2Flag	—	
	J2Flag 関数	—	
	J4Flag	—	
	J4Flag 関数	—	
	J6Flag	—	
	J6Flag 関数	—	
	JA 関数	—	
	Joint	—	
	JRange	—	
	JRange 関数	—	
	JS 関数	—	
	JTran	—	
	JT 関数	—	
	Jump	+	動作モード指定を追加
	Jump3	+	
	Jump3CP	+	
L	LCase\$関数	—	
	Left\$関数	—	
	Len 関数	—	
	LimZ	—	
	LimZ 関数	—	
	Line Input	—	
	Line Input#	+	デバイス番号追加
	LJM 関数	—	
	LoadPoints	—	
	Local	—	
	LocalClr	—	
	LocalDef 関数	—	
	Local 関数	—	
	Lof 関数	—	
	Long	—	
	LSet\$関数	—	
	LShift 関数	—	
	LTrim\$関数	—	
M	Mask	—	
	MemInW 関数	—	
	MemIn 関数	—	
	MemOff	—	

	命令	互換性	備考
	MemOn	—	
	MemOut	—	
	MemOutW	—	
	MemSw 関数	—	
	Mid\$関数	—	
	Mod	—	
	Motor	—	
	Motor 関数	—	
	Move	—	
	MyTask 関数	—	
N	Not	—	
O	Off	—	
	OLAccel	—	
	OLAccel 関数	—	
	OLRate	—	
	OLRate 関数	—	
	On	—	
	OnErr	—	
	OpBCD	—	
	OpenCom	—	
	OpenNet	—	
	Oport 関数	—	
	Or	—	
	Out	—	
	OutW	—	
	OutW 関数	—	
	Out 関数	—	
P	P#	—	
	PAgl 関数	—	
	Pallet	—	座標値指定の追加
	Pallet 関数	—	
	ParseStr	—	
	ParseStr 関数	—	
	Pass	+	
	Pause	—	
	PauseOn 関数	—	
	PDef 関数	—	
	PDel	—	
	PLabel	—	
	PLabel\$関数	—	
	Plane	+	ロボット番号の指定を追加
	Plane 関数	+	ロボット番号の指定を追加
	PlaneClr	+	ロボット番号の指定を追加
	PlaneDef 関数	+	ロボット番号の指定を追加
	PList	!	表示形式変更
	PLocal	—	

	命令	互換性	備考
	PLocal 関数	—	
	Pls 関数	—	
	PNumber 関数	—	
	PosFound 関数	—	
	Power	—	
	Power 関数	—	
	PPls 関数	—	
	Print	—	
	Print#	+	デバイス番号変更
	PTCLR	—	
	PTPBoost	—	
	PTPBoostOK 関数	—	
	PTPBoost 関数	—	
	PTPTIME 関数	—	
	PTran	—	
	PTRQ	—	
	PTRQ 関数	—	
	Pulse	—	
	Pulse 関数	—	
Q	QP	—	
	Quit	—	
R	RadToDeg 関数	—	
	Randmize	—	
	Range	—	
	Read	—	
	ReadBin	—	
	Real	—	
	RealPls 関数	—	
	RealPos 関数	—	
	RealTorque	—	
	Redim	—	
	Reset	—	
	ResetElapsedTime	—	
	Resume	—	
	Return	—	
	RobotInfo 関数	+	情報の追加
	RobotInfo\$関数	+	デフォルトポイントファイル名の表示追加
	RobotModel\$関数	—	
	RobotName\$関数	—	
	RobotSerial\$関数	—	
	RobotType 関数	—	
	RSet\$関数	—	
	RShift 関数	—	
	RTrim\$関数	—	
S	SafetyOn 関数	—	
	SavePoints	—	

命令		互換性	備考
	Select...Send	—	
	Sense	—	
	SetCom	—	
	SetInW	—	
	SetIn	—	
	SetNet	—	
	SetSw	—	
	SFree	—	
	SFree 関数	—	
	Sgn 関数	—	
	Signal	—	
	Sin 関数	—	
	SLock	—	
	SoftCP	—	
	SoftCP 関数	—	
	Space\$関数	—	
	Speed	—	
	SpeedR	—	
	SpeedR 関数	—	
	SpeedS	—	
	SpeedS 関数	—	
	Speed 関数	—	
	SPELCom_Event	—	
	Sqr 関数	—	
	Stat 関数	+	情報の追加
	Str\$関数	—	
	String	—	
	Sw 関数	—	
	SyncLock	—	
	SyncUnlock	—	
	SysConfig	+	情報の追加
	SysErr 関数	+	ワーニング取得機能追加
T	Tab\$関数	—	
	Tan 関数	—	
	TargetOK 関数	—	
	TaskDone 関数	—	
	TaskInfo 関数	—	
	TaskInfo\$関数	—	
	TaskState	+	バックグラウンドタスクの表示追加
	TaskState 関数	—	
	TaskWait	—	
	TC	—	
	TCLim	—	
	TCLim 関数	—	
	TCSpeed	—	
	TCSpeed 関数	—	

命令		互換性	備考
	TGo	+	動作モード指定を追加
	TillOn 関数	—	
	Time	!	表示のみ
	Time 関数	—	
	Time\$関数	—	
	TLClr	—	
	TlDef 関数	—	
	TLSet	—	
	TLSet 関数	—	
	TMOut	—	
	TMove	—	
	TmReset	—	
	Tmr 関数	—	
	Toff	—	
	Ton	—	
	Tool	—	
	Tool 関数	—	
	Trap	!	コントローラー状態をトリガーとする Trap を追加
	Trim\$関数	—	
	Tw 関数	—	
U	UBound 関数	—	
	UCase\$関数	—	
V	Val 関数	—	
W	Wait	!	グローバル変数などが待ち条件に追加
	WaitNet	—	
	WaitPos	—	
	WaitSig	—	
	Weight	+	S,T の指定追加
	Weight 関数	+	S,T の指定追加
	Where	—	
	Wrist	—	
	Wrist 関数	—	
	Write	—	
	WriteBin	—	
X	Xor	—	
	Xqt	—	
	XY 関数	—	
	XYLim	—	
	XYLim 関数	—	
	XYLimClr	—	
	XYLimDef	—	
	XYLimDef 関数	—	

EPSON RC+ Ver.4.*からの移行コマンド一覧 (EPSON RC+5.0 未サポート)

AOpen	Cnv_QueueUserData	Hofs
BOpen	Cnv_QueueUserData 関数	Hofs 関数
Calib	Cnv_RobotConveyor 関数	ImportPoints
CalPls	Cnv_Speed 関数	InputBox
ChDir	Cnv_Trigger	LogIn 関数
ChDrive	Cnv_Upstream 関数	MCalComplete 関数
Close	Cont	MCal
Cnv_AbortTrack	Copy	MCordr
Cnv_Downstream	CurDir\$関数	MCordr 関数
Cnv_Fine	CurDrive\$関数	MKDir
Cnv_Fine 関数	Declare	MsgBox
Cnv_Name\$ 関数	Del	Recover 関数
Cnv_Number 関数	Eof 関数	Rename
Cnv_Point 関数	Eval 関数	RenDir
Cnv_PosErr 関数	FbusIO_GetBusStatus 関数	Restart
Cnv_Pulse 関数	FbusIO_GetDeviceStatus 関数	Rmdir
Cnv_QueueAdd	FbusIO_SendMsg	Robot
Cnv_QueueGet 関数	FileDateTime\$関数	Robot 関数
Cnv_QueueLen 関数	FileExists 関数	ROpen
Cnv_QueueList	FileLen 関数	RunDialog
Cnv_QueueMove	FolderExists 関数	Seek
Cnv_QueueReject	FreeFile 関数	Shutdown
Cnv_QueueReject 関数	GetCurrentUser\$	UOpen
Cnv_QueueRemove		WOpen

B-3: EPSON RC+ Ver.4.* からの互換性に関する注意

概要

すでに、RC520および、RC420コントローラーで、EPSON RC+ Ver.4.*をご利用の方が、RC700シリーズコントローラーでEPSON RC+ 7.0を使う場合に読んでいただきたい内容をまとめました。

EPSON RC+ 7.0を使用する場合は、ハードウェアの違い、対応マニピュレーターや関節数の違い、ソフトウェアの実行環境の違いなどにより、EPSON RC+ Ver.4.*とは異なっている部分があります。あらかじめその内容をご理解いただき、安全にロボットをご使用してください。

EPSON RC+7.0は、できるだけ今までの製品と互換性を保ちながら改善し、先進のソフトウェア技術を取り入れて設計しています。しかしながら、ロボットコントローラーに特化し、より使いやすくするために、これまでのEPSON RC+ Ver.4.*との互換性を全て満たしてはいない部分や、削除された部分があります。

以下の互換性に関する記述は、EPSON RC+ Ver.4.*を元に、EPSON RC+ 7.0と比較をしています。

全般にわたる違い

EPSON RC+ Ver.4.*と、EPSON RC+ 7.0の全般にわたる違いは下記のとおりです。

項目	EPSON RC+ 7.0	EPSON RC+ Ver.4.*
タスク数	最大 32 タスク (バックグラウンドタスクは最大 16 タスク)	最大 32 タスク
タスクの種類	NoPause タスクの指定が可 NoEmgAbort タスクの指定が可 バックグラウンドタスクの指定が可	NoPause タスクの指定が可能
TRAP ERROR などの特殊な TRAP	サポートされています	サポートされています
TRAPトラップ番号で起動されるタスク	専用のタスク番号	1 から 32 のみ使用のタスク番号
マルチマニピュレーター	サポートされています	サポートされています
ロボット番号	1~16	1~16
Real 型の有効桁数	6 桁	7 桁
Double 型の有効桁数	14 桁	15 桁
配列要素数	文字列変数以外 ローカル変数 2000 グローバル変数 1,000,000 モジュール変数 1,000,000 バックアップ変数 4000 文字列変数 ローカル変数 200 グローバル変数 10,000 モジュール変数 10,000 バックアップ変数 400	メモリーの許す限り使用可能
行番号	サポートされません	サポートされています

項目	EPSON RC+ 7.0	EPSON RC+ Ver.4.*
デバイス番号	21:PC 22:リモート 24:TP 20:TP3	1:コントローラー 2:リモート 3:OP
コントロールデバイス	リモート I/O PC リモート COM リモート Ethernet TP3	リモート I/O PC OP500RC
タイマー番号の範囲	0~63	0~63
プログラム容量	8MB	4MB
SyncLock, SyncUnlock 用の シグナル番号の範囲	0~63	1~32
WaitSig, Signal 用の シグナル番号の範囲	0~63	0~127
メモリー I/O 点数	1024	512
I/O ポートの番号	RC+4.0 とは異なる	
イーサネットのポート番号	201~216	128~147
リモート入出力割りあて	標準で機能割りあて済	デフォルト設定なし
RS-232C 通信ポート番号 SPEL+制御部	1~8, 1001~1008	1~16
RS-232C 通信ポートの OpenCom 実行	必須	任意
ファイルへの入出力	サポートされています	サポートされています
ファイルアクセスのファイル番号	30~63	30~63
データベースアクセス番号	501~508	サポートされません
VisionGuide	スマートカメラタイプ フレームグラバタイプ	フレームグラバタイプ
コンベヤトラッキング	サポートされています	サポートされています
PG ロボット	サポートされています	サポートされています
OCR	サポートされています	サポートされています
セキュリティ	サポートされています	サポートされています
VBGuide (RC+ API 7.0)	サポートされています	サポートされています
フィールドバス I/O の扱い	通常の I/O 命令を使用	特殊な命令を使用
フィールドバス マスター	応答性が保証されません	応答性が保証されません
フィールドバス スレーブ	応答性が保証されています	応答性が保証されません
GUI Builder	サポートされています	サポートされません
プロジェクト内のグループ	サポートされません	サポートされています
エラー番号	EPSON RC+ Ver.4.*とは 異なるエラー番号	

命令の互換性一覧

- + 機能拡張、機能変更されているが上位互換が保たれている命令
- 変更がない命令
- ! 機能変更または、文法変更があり注意が必要な命令
- !! 大幅な変更があり注意が必要な命令
- × 削除された命令

	命令	互換性	備考
A	Abs 関数	–	
	Accel	+	ロボットによっては 100 以上を設定可能
	Accel 関数	–	
	AccelR	–	
	AccelR 関数	–	
	AccelS	–	
	AccelS 関数	–	
	Acos 関数	+	引数の範囲チェック追加
	Agl 関数	–	
	AglToPls 関数	–	
	And	–	
	AOpen	–	
	Arc	–	
	Arc3	–	
	Arch	–	
	Arch 関数	–	
	Arm	–	
	Arm 関数	–	
	ArmClr	–	
	ArmSet	–	
	ArmSet 関数	–	
	Asc 関数	–	
	Asin 関数	+	引数の範囲チェック追加
	Atan 関数	–	
	Atan2 関数	–	
	ATCLR	–	
	ATRQ	–	
	ATRQ 関数	–	
B	Base	–	
	BClr 関数	+	引数の範囲チェック追加
	Beep	×	
	BGo	+	動作モード指定を追加
	BMove	–	
	Boolean	–	
	BOpen	–	
	Brake	–	
	BSet 関数	+	引数の範囲チェック追加
	BTst 関数	+	引数の範囲チェック追加
	Byte	–	
C	Calib	–	

	命令	互換性	備考
	Call	—	
	CalPls	—	
	CalPls 関数	—	
	Chain	×	
	ChDir	—	
	ChDrive	—	
	ChkCom 関数	—	
	ChkNet 関数	—	
	Chr\$関数	—	
	Clear	!	ClearPoints に名称変更
	Close	—	
	CloseCom	—	
	CloseNet	+	All の指定を追加
	ClrScr	!	Cls に名称変更・引数にデバイス ID の指定が可能
	Cnv_**	—	
	Cont	!	設定により実行可能
	Copy	—	
	Cos 関数	—	
	CP	—	
	CP 関数	—	
	Ctr 関数	—	
	CTReset	—	
	CtrlDev	×	
	CtrlDev 関数	!	デバイス ID 変更
	CtrlInfo 関数	!!	取得内容変更
	CurDir\$関数	—	
	CurDrive\$関数	—	
	CurPos 関数	—	
	Curve	—	
	CVMove	—	
	CX~CW	+	CR, CS, CT 追加
	CX~CW 関数	+	CR, CS, CT 追加
D	Date	!	表示のみ
	Date\$関数	—	
	Declare	!	処理にかかる時間が長くなる
	DegToRad 関数	—	
	Del	—	
	Dir	—	
	Dist 関数	—	
	Do...Loop	—	
	Double	!	有効桁数 14 桁
E	EClr	×	
	ECP	—	
	ECP 関数	—	
	ECPClr	—	
	ECPSet	—	
	ECPSet 関数	—	
	Elbow	—	

	命令	互換性	備考
	Elbow 関数	—	
	ENetIO ****	×	
	Eof 関数	—	
	EPrint	×	
	Era 関数	—	
	Erase	×	
	EResume	—	
	Erf\$関数	+	タスク番号省略可能
	Erl 関数	+	タスク番号省略可能
	Err 関数	—	
	ErrHist	×	
	ErrMsg\$関数	!	言語 ID を引数に持つ
	Error	+	引数にタスク番号指定可能
	Ert 関数	—	
	EStopOn 関数	+	Wait 待ち可能
	Eval 関数	!	エラー発生時の出力結果に違いがある
	Exit	—	
F	FbusIO ****	×	通常の I/O 命令が使用可能
	FileDateTime\$関数	—	
	FileExists 関数	—	
	FileLen 関数	—	
	Find	—	
	FindPos 関数	—	
	Fine	—	
	Fine 関数	—	
	Fix 関数	—	
	FmtStr\$	!!	機能が大幅に限定
	FoldrExist 関数	—	
	For...Next	—	
	FreeFile 関数	—	
	Function...Fend	—	
G	GetCurrentUser\$関数	—	
	Global	—	
	Go	+	動作モード指定を追加
	Gosub...Return	—	
	Goto	—	
H	Halt	—	
	Hand	—	
	Hand 関数	—	
	Here	—	
	Here 関数	—	
	Hex\$関数	—	
	Hofs	—	
	Hofs 関数	—	
	Home	—	
	HomeSet	—	
	HomeSet 関数	—	
	HOrdr	—	

	命令	互換性	備考
	HOrdr 関数	—	
	Hour	—	
	Hour 関数	—	
	HTest	×	
	HTest 関数	×	
I	If...EndIf	—	
	ImportPoints	!	拡張子 “.pnt” から “.pts” へ変更
	In 関数	—	
	In(\$n)	×	MemIn に置き換え
	InBCD 関数	—	
	Inertia	—	
	Inertia 関数	—	
	InPos 関数	—	
	Input	—	
	Input#	+	各種デバイスからの Input も可能
	InputBox	—	
	InStr 関数	—	
	Int 関数	—	
	Integer	—	
	InW 関数	—	
	InW(\$n)	×	MemInW に置き換え
	IONumber 関数	—	
J	J4Flag	—	
	J4Flag 関数	—	
	J6Flag	—	
	J6Flag 関数	—	
	JA 関数	—	
	JRange	—	
	JRange 関数	—	
	JS 関数	!	True/False を返す
	JT 関数	—	
	JTran	—	
	Jump	+	動作モード指定を追加
	Jump3	—	
	Jump3CP	—	
K	Kill	×	Del に置き換え
L	LCase\$関数	—	
	Left\$関数	—	
	Len 関数	—	
	LimZ	—	
	LimZ 関数	—	
	Line Input	—	
	Line Input#	+	各種デバイスからの Input も可能
	LoadPoints	!	拡張子 “.pnt” から “.pts” へ変更
	Local	!	ローカル番号 0 は、エラー
	Local 関数	!	ローカル番号 0 は、エラー
	LocalClr	—	
	Lof 関数	—	

	命令	互換性	備考
	LogIn	!	コマンドから関数に変更
	Long	—	
	LPrint	×	
	LSet\$関数	—	
	LShift 関数	+	引数の範囲チェック追加
	LTrim\$関数	—	
M	Mask	—	
	MCal	—	
	MCalComplete 関数	—	
	MCofs	×	
	MCofs 関数	×	
	MCordr	—	
	MCordr 関数	—	
	Mcorg	×	
	MemIn 関数	—	
	MemInW 関数	—	
	MemOff	—	
	MemOn	—	
	MemOut	—	
	MemOutW	—	
	MemSw 関数	—	
	Mid\$関数	—	
	MKDir	—	
	Mod	—	
	Motor	—	
	Motor 関数	—	
	Move	—	
	MsgBox	—	
	MyTask 関数	—	
N	Not	—	
O	Off	—	
	Off\$	×	MemOff に置き換え
	OLRate	—	
	OLRate 関数	—	
	On	—	
	On\$	×	MemOn に置き換え
	OnErr	—	
	OP *	×	
	OpBCD	—	
	OpenCom	!	OpenCom は必須
	OpenNet	—	
	Oport 関数	—	
	Or	—	
	Out	—	
	Out 関数	—	
	Out\$	×	MemOut に置き換え
	OutW	—	
	OutW 関数	—	

命令		互換性	備考
	OutW\$	×	MemOutW に置き換え
P	PAgl 関数	—	
	Pallet	—	
	Pallet 関数	—	
	ParseStr	—	
	ParseStr 関数	—	
	Pass	+	連続ポイントの指定可能
	Pause	—	
	PauseOn 関数	—	
	PDef 関数	—	
	PDel	+	引数のチェック追加
	PLabel\$関数	—	
	PLabel	—	
	PList	!!	表示形式変更 引数のチェック追加 Plist*の機能は削除
	PLocal	—	
	PLocal 関数	—	
	Pls 関数	—	
	PNumber 関数	—	
	P#	—	
	POrient	×	
	POrient 関数	×	
	PosFound 関数	!	True/False を返す
	Power	—	
	Power 関数	—	
	PPls 関数	—	
	Print	!	ポイントを出力する場合は、全てのフラグを出力する Double 型, Real 型の出力桁数を有効桁数に合わせる
	Print#	!	Print と同じ 各種デバイスへの Print も可能
	PTCLR	—	
	PTPBoost	—	
	PTPBoost 関数	—	
	PTPBoostOK 関数	!	True/False を返す
	PTPTIME 関数	—	
	PTran	—	
	PTRQ	—	
	PTRQ 関数	—	
	Pulse	—	
	Pulse 関数	—	
Q	QP	—	
	Quit	—	
R	RadToDeg 関数	—	
	Randmize	+	シード値を指定可能
	Range	—	
	Read	—	
	ReadBin	+	複数バイトを配列変数に読み込むことも可能
	Real	!	有効桁数 6 桁

	命令	互換性	備考
	Recover	!	設定により実行可能
	Redim	!	要素数に上限あり 参照渡しされた配列には実行できない
	Rename	—	
	RenDir	—	
	Reset	—	
	Resume	—	
	Restart	—	
	Reset	+	Reset Error が追加
	Return	—	
	Right\$関数	—	
	RmDir	—	
	Rnd 関数	—	
	Robot	—	
	Robot 関数	—	
	RobotModel\$関数	—	
	RobotType 関数	+	RS シリーズ追加
	ROpen	—	
	RSet\$関数	—	
	RShift 関数	+	引数チェック追加
	RTrim\$関数	—	
	RunDialog	—	
S	SafetyOn 関数	+	Wait 待ち可能
	SavePoints	!	拡張子 “.pnt” から “.pts” へ変更
	Seek	—	
	Select...Send	—	
	Sense	—	
	SetCom	!	転送速度 “56000” は、指定できない OpenCom されているポートに対しては実行できない
	SetNet	—	
	SFree	—	
	SFree 関数	—	
	Sgn 関数	—	
	Shutdown	—	
	Signal	—	
	Sin 関数	—	
	SLock	—	
	Space\$関数	—	
	Speed	—	
	Speed 関数	+	引数省略可能
	SpeedR	—	
	SpeedR 関数	—	
	SpeedS	—	
	SpeedS 関数	—	
	SPELCom_Event	—	
	SPELCom_Return	×	
	Sqr 関数	—	
	Stat 関数	!	一部情報が取得できない
	Str\$関数	—	

	命令	互換性	備考
	String	—	
	Sw 関数	—	
	Sw(\$)関数	×	MemSw に置き換え
	SyncLock	!	SyncLock の実行後、再度 SyncLock を行うとエラー タスク終了でロックを解除
	SyncUnlock	—	
T	Tab\$関数	—	
	Tan 関数	—	
	TargetOK 関数	!	True/False を返す
	TaskDone 関数	—	
	TaskState 関数	!	6 指定タスクは、Wait ステートメント実行中は返らない
	TaskWait	—	
	TGo	+	動作モード指定を追加
	TillOn 関数	—	
	Time	!	表示のみ
	Time 関数	—	
	Time\$関数	—	
	TLClr	—	
	TLSet	—	
	TLSet 関数	—	
	TMOut	—	
	TMove	—	
	Tmr 関数	—	
	TmReset	—	
	Tool	—	
	Tool 関数	—	
	Trap	!!	Trap Goto 互換性あり Trap Gosub 廃止 Trap Call に置き換え Trap Call は Trap Xqt に名称変更 Trap Finish を追加
	Trim\$関数	—	
	Tw 関数	!	True/False を返す
	Type	—	
U	UBound 関数	—	
	UCase\$関数	—	
	UOpen	—	
V	Val 関数	—	
	Ver	×	SysConfig に置き換え
	Verinit	×	
W	Wait	+	グローバル変数などが待ち条件に追加
	WaitNet	—	
	WaitPos	—	
	WaitSig	—	
	Weight	+	S,T の指定追加
	Weight 関数	+	S,T の指定追加
	Where	!	座標値は常に 6 軸表示
	While..Wend	×	Do...Loop に置き換え
	WOpen	—	
	Wrist	—	

命令		互換性	備考
	Wrist 関数	—	
	Write	—	
	WriteBin	+	複数バイトを配列変数から書き出すことも可能
X	Xor	—	
	Xqt	+	NoEmgAbort 指定を追加
	XY 関数	—	
	XYLim	—	
	XYLim 関数	—	
Z	ZeroFlg 関数	×	

Appendix C: EPSON RC+7.0 のコマンド

C-1: EPSON RC+ 4.0 以降で追加されたコマンド一覧

AbortMotion	ChDisk	EStopOn 関数
AccelMax 関数	ChkCom 関数	Exit
AglToPls 関数	ChkNet 関数	ExportPoints
AIO_Out	CloseCom	
AIO_Out 関数	CloseDB	FindPos 関数
AIO_OutW	CloseNet	Find
AIO_OutW 関数	Cls	FineDist
AIO_Set	CP	FineDist 関数
AIO_Set 関数	CP 関数	FineStatus 関数 Fix 関数
AIO_TrackingSet	CP_Offset	Flush
AIO_TrackingStart	CP_Offset 関数	Fmtstr
AIO_TrackingEnd	CR	
AIO_TrackingOn 関数	CR 関数	GetRobotInsideBox 関数
AIO_In 関数	CS	GetRobotInsidePlane 関数
AIO_InW 関数	CS 関数	
Align 関数	CT	Hand_On
AlignECP 関数	CT 関数	Hand_On 関数
AreaCorrection 関数	CtrlDev 関数	Hand_Off
AreaCorrectionClr	Curve	Hand_Off 関数
AreaCorrectionDef 関数	CVMove	Hand_TW 関数
AreaCorrectionInv 関数	Cnv_Accel	Hand_Def 関数
AreaCorrectionOffset 関数	Cnv_Accel 関数	Hand_Type 関数
AreaCorrectionSet	Cnv_AccelLim	Hand_Label\$ 関数
ArmCalib	Cnv_AccelLim 関数	Hand_Number 関数
ArmCalib 関数 ArmCalibSet	Cnv_Adjust	HealthCalcPeriod
ArmCalibSet 関数	Cnv_AdjustClear	HealthCalcPeriod 関数
ArmCalibClr	Cnv_AdjustGet	HealthCtrlAlarmOn 関数
ArmCalibDef 関数	Cnv_AdjustSet	HealthCtrlInfo
ArmDef 関数	Cnv_DownStream	HealthCtrlInfo 関数
ATCLR	Cnv_Mode	HealthCtrlRateOffset
AtHome 関数	Cnv_Mode 関数	HealthCtrlReset
ATRQ	Cnv_OffsetAngle	HealthCtrlWarningEnable
ATRQ 関数	Cnv_OffsetAngle 関数	HealthCtrlWarningEnable 関数
AutoLJM	Cnv_PosErrOffset	HealthRateCtrlInfo 関数
AutoLJM 関数	Cnv_Upstream	HealthRateRInfo 関数
AvoidSingularity	CollisionDetect	HealthRBAlarmOn 関数
AvoidSingularity 関数	CollisionDetect 関数	HealthRBAnalysis
		HealthRBAnalysis 関数
BCLr 関数	DegToRad 関数	HealthRBDistance
BCLr64 関数	DeleteDB	HealthRBDistance 関数
Box	DiffPoint 関数	HealthRBInfo
Box 関数	DispDev	HealthRBInfo 関数
BoxClr 関数	DispDev 関数	HealthRBRateOffset
BoxDef 関数	Dist 関数	HealthRBReset
Brake 関数		HealthRBSpeed
BSet 関数	EcpDef 関数	HealthRBSpeed 関数
BSet64 関数	ElapsedTime 関数	HealthRBStart
BTst 関数	EResume	HealthRBStop
BTst64 関数	Errb 関数	HealthRBTRQ
	ErrorOn 関数	HealthRBTRQ 関数
	Error	HealthRBWarningEnable
		HealthRBWarningEnable 関数

Here	P#	RobotType 関数
Here 関数	PalletClr	ROTOk 関数
Hex\$関数	PauseOn 関数	RShift64関数
HofsJointAccuracy	PDef 関数	
HomeClr	PDcl	SafetyOn関数
HomeDef 関数	PDdescription	SelectDB
	PDdescription 関数	SetCom
InReal 関数	PerformMode	SetInW
InsideBox 関数	PerformMode 関数	SetIn
InsidePlane 関数	PG_FastStop	SetNet
InStr 関数	PG_LSpeed	SetSw
IODef 関数	PG_LSpeed 関数	SF_GetParam関数
IOLabel\$関数	PG_Scan	SF_GetParam\$関数
IONumber 関数	PG_SlowStop	SF_GetStatus関数
	PLabel	SF_LimitSpeedS
J1Angle	PLabel\$関数	SF_LimitSpeedS関数
J1Angle 関数	PlaneClr	SF_LimitSpeedEnable
J4Angle	PlaneDef	SF_LimitSpeedEnable関数
JA 関数	Plane	SF_PeakSpeedS
Joint	Plane 関数	SF_PeakSpeedS関数
JointAccuracy	PList	SF_PeakSpeedSClear
JointAccuracy 関数	PLocal	SF_RealSpeedS
JumpTLZ	PLocal 関数	SF_RealSpeedS関数
JTran	PNumber 関数	Shutdown関数
	PosFound 関数	SimGet
LatchEnable	PTCLR	SimSet
LatchState 関数	PTPBoostOK 関数	SingularityAngle
LatchPos 関数	PTPTIME 関数	SingularityAngle関数
LimZMargin	PTran	SingularityDist
LimZMargin 関数	PTRQ	SingularityDist関数
LimitTorque	PTRQ 関数	SingularitySpeed
LimitTorque 関数		SingularitySpeed関数
LimitTorqueLP	QPDECEL	SoftCP
LimitTorqueLP 関数	QPDECEL 関数	SoftCP関数
LimitTorqueStop	QPDECELS	SpeedFactor
LimitTorqueStop 関数	QPDECELS 関数	SpeedFactor関数
LimitTorqueStopLP		StartMain
LimitTorqueStopLP 関数	RadToDeg 関数	SyncRobots
LJM 関数	Randomize	SyncRobots関数
LocalDef 関数	ReadBin	SysErr関数
LShift64 関数	Read	
	RealAccel 関数	Tab\$関数
MemInW 関数	RealPls 関数	TargetOK関数
MemOutW	RealPos 関数	TaskDone関数
MHour 関数	RealTorque 関数	TaskInfo関数
	RecoverPos 関数	TaskInfo\$関数
OLAccel	Recover	TaskState
OLAccel 関数	Redim	TaskState関数
OpenCom	ResetElapsedTime	TaskWait
OpenCom 関数	Rnd 関数	TC
OpenDB	RobotInfo 関数	TCLim
OpenNet	RobotInfo\$関数	TCLim関数
OpenNet 関数	RobotModel\$関数	TCSpeed
OutReal	RobotName\$関数	TCSpeed 関数
OutReal 関数	RobotSerial\$関数	TeachOn関数

TillOn関数	VxCalib	WorkQue_Reject
TlDef 関数	VxCalDelete	WorkQue_Reject 関数
Toff	VxCalLoad	WorkQue_Remove
Ton	VxCalInfo 関数	WorkQue_Sort
	VxCalSave	WorkQue_Sort 関数
UBound 関数		WorkQue_UserData
UpdateDB	WaitNet	WorkQue_UserData 関数
	WaitPos	WriteBin
VDefArm	Where	Write
VDefLocal	WindosStatus 関数	
VDefSetMotionRange	WorkQue_Add	XYLimClr
VDefGetMotionRange	WorkQue_AutoRemove	XYLimDef
VDefTool	WorkQue_AutoRemove 関数	XY関数
VGoCenter	WorkQue_Get 関数	
VSD	WorkQue_Len 関数	
VSD 関数	WorkQue_List	

C-2: EPSON RC+ 7.0 のバージョンごとに追加されたコマンド一覧

EPSON RC+ 6.0, 5.0, 4.0 共通

EPSON RC+7.0 のバージョン	追加されたコマンド	
Ver.7.5.4	Cnv_AccelLim Cnv_AccelLim関数 SF_GetParam関数 SF_GetParam\$関数 SF_GetStatus関数 SF_LimitSpeedS SF_LimitSpeedS関数	SF_LimitSpeedEnable SF_LimitSpeedEnable関数 SF_PeakSpeedS SF_PeakSpeedS関数 SF_PeakSpeedSClear SF_RealSpeedS SF_RealSpeedS関数
Ver.7.5.3	AreaCorrection関数 AreaCorrectionClr AreaCorrectionDef関数 AreaCorrectionInv関数	AreaCorrectionOffset関数 AreaCorrectionSet Cnv_PosErrOffset
Ver.7.5.2	XYLimMode XYLimMode関数	
Ver.7.5.1	ArmCalib ArmCalib関数 ArmCalibSet ArmCalibSet関数 ArmCalibClr ArmCalibDef関数 JointAccuracy JointAccuracy関数 HofsJointAccuracy Hand_On Hand_On関数 Hand_Off Hand_Off関数	Hand_TW 関数 Hand_Def 関数 Hand_Type 関数 Hand_Label\$ 関数 Hand_Number 関数 Cnv_Adjust Cnv_AdjustClear Cnv_AdjustGet Cnv_AdjustSet DiffPoint関数 ROTOK関数
Ver.7.4.3	AIO_TrackingSet AIO_TrackingStart	AIO_TrackingEnd AIO_TrackingOn関数
Ver.7.4.1	AutoOrientationFlag AutoOrientationFlag 関数	
Ver.7.3.4	SimGet SimSet	
Ver.7.3.3	HealthCtrlWarningEnable HealthCtrlWarningEnable 関数	HealthRBWarningEnable HealthRBWarningEnable 関数
Ver.7.3.2	PDescription PDescription 関数	
Ver.7.3.1	AIO_Out AIO_Out 関数 AIO_OutW AIO_OutW 関数 AIO_Set	AIO_Set 関数 AIO_In 関数 AIO_InW 関数 HealthCalcPeriod HealthCalcPeriod 関数
Ver.7.3.0	VDefTool VDefArm VDefLocal	VGoCenter VDefSetMotionRange VDefGetMotionRange

EPSON RC+7.0 のバージョン	追加されたコマンド	
Ver.7.2.0	CP_Offset CP_Offset 関数 HealthCtrlAlarmOn 関数 HealthCtrlInfo HealthCtrlInfo 関数 HealthCtrlRateOffset HealthCtrlReset HealthRateCtrlInfo 関数 HealthRateRBInfo 関数 HealthRBAAlarmOn 関数 HealthRBAAnalysis HealthRBAAnalysis 関数 HealthRBDistance HealthRBDistance 関数 HealthRBInfo HealthRBInfo 関数 HealthRBRateOffset	HealthRBReset HealthRBSpeed HealthRBSpeed 関数 HealthRBStart HealthRBStop HealthRBTRQ HealthRBTRQ 関数 J4Angle JumpTLZ LimitTorqueLP LimitTorqueLP 関数 LimitTorqueStop LimitTorqueStop 関数 LimitTorqueStopLP LimitTorqueStopLP 関数 VSD VSD 関数
Ver.7.1.4	CollisionDetect 関数	
Ver.7.1.3	CollisionDetect MHour 関数	
Ver.7.1.2	SingularityDist SingularityDist 関数	ExportPoints
Ver.7.1.0	BClr64 関数 BSet64 関数 BTst64 関数 FineDist FineDist 関数 FineStatus 関数 Fmtstr IODef 関数 LShift64 関数 RealAccel 関数 RShift64 関数 WorkQue_Add WorkQue_AutoRemove	WorkQue_AutoRemove 関数 WorkQue_Get 関数 WorkQue_Len 関数 WorkQue_List WorkQue_Reject WorkQue_Reject 関数 WorkQue_Remove WorkQue_Sort WorkQue_Sort 関数 WorkQue_UserData WorkQue_UserData 関数
Ver.7.0.3	PerformMode	PerformMode 関数

NOTE



EPSON RC+7.0 Ver.7.0.0 の場合は、
EPSON RC+ 6.0, 5.0, 4.0 では、追加されたコマンドが異なります。

EPSON RC+ 6.0, 5.0 に対して追加されたコマンド

EPSON RC+7.0 のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0
Ver.7.0.0	AutoLJM AutoLJM 関数 AvoidSingularity AvoidSingularity 関数 Cnv_Accel Cnv_Accel 関数 Cnv_DownStream Cnv_Mode Cnv_Mode 関数 Cnv_Upstream DeleteDB ElapsedTime 関数 Errb 関数 LimZMargin LimZMargin 関数 LimitTorque LimitTorque 関数 PalletClr ResetElapsedTime	AbortMotion AutoLJM AutoLJM 関数 AvoidSingularity AvoidSingularity 関数 ChDisk CloseDB Cnv_Accel Cnv_Accel 関数 Cnv_DownStream Cnv_Mode Cnv_Mode 関数 Cnv_Upstream CR CR 関数 CS CS 関数 CT CT 関数 DeleteDB Errb 関数 Flush GetRobotInsideBox 関数 GetRobotInsidePlane 関数 J1Angle J1Angle 関数 LimZMargin LimZMargin 関数 LimitTorque LimitTorque 関数 OpenDB PalletClr PG_FastStop PG_LSpeed PG_LSpeed 関数 PG_Scan PG_SlowStop QPDECELR QPDECELR 関数 QPDECELS QPDECELS 関数 RecoverPos 関数 Recover SelectDB Shutdown 関数

EPSON RC+7.0 のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0
Ver.7.0.0	SingularityAngle SingularityAngle 関数 SingularitySpeed SingularitySpeed 関数 SpeedFactor SpeedFactor 関数 UpdateDB	SingularityAngle SingularityAngle 関数 SingularitySpeed SingularitySpeed 関数 SpeedFactor SpeedFactor 関数 StartMain SyncRobots SyncRobots 関数 TeachOn 関数 UpdateDB WindosStatus 関数

EPSON RC+ 4.0 に対して追加されたコマンド

EPSON RC+7.0 のバージョン	EPSON RC+ 4.0	
Ver.7.0.0	AbortMotion AccelMax 関数 AglToPls 関数 Align 関数 AlignECP 関数 ArmDef 関数 ATCLR AtHome 関数 ATRQ ATRQ 関数 AutoLJM AutoLJM 関数 AvoidSingularity AvoidSingularity 関数 BClr 関数 Box Box 関数 BoxClr 関数 BoxDef 関数 Brake 関数 BSet 関数 BTst 関数 ChDisk ChkCom 関数 ChkNet 関数 CloseCom CloseDB CloseNet Cls CP CP 関数	CR CR 関数 CS CS 関数 CT CT 関数 CtrlDev 関数 Curve CVMove Cnv_Accel Cnv_Accel 関数 Cnv_DownStream Cnv_Mode Cnv_Mode 関数 Cnv_OffsetAngle Cnv_OffsetAngle 関数 Cnv_Upstream DegToRad 関数 DeleteDB DispDev DispDev 関数 Dist 関数 EcpDef 関数 EResume Errb 関数 ErrorOn 関数 Error EStopOn 関数 Exit

EPSON RC+7.0 のバージョン	EPSON RC+ 4.0	
Ver.7.0.0	FindPos 関数 Find FineStatus 関数 Fix 関数 Flush GetRobotInsideBox 関数 GetRobotInsidePlane 関数 Here Here 関数 Hex\$関数 HomeClr HomeDef 関数 InReal 関数 InsideBox 関数 InsidePlane 関数 InStr 関数 IOLabel\$関数 IONumber 関数 J1Angle J1Angle 関数 JA 関数 Joint JTran LatchEnable LatchState 関数 LatchPos 関数 LimZMargin LimZMargin 関数 LimitTorque LimitTorque 関数 LJM 関数 LocalDef 関数 MemInW 関数 MemOutW OLAccel OLAccel 関数 OpenCom OpenCom 関数 OpenDB OpenNet OpenNet 関数 OutReal OutReal 関数	P# PalletClr PauseOn 関数 PDef 関数 PDel PG_FastStop PG_LSpeed PG_LSpeed 関数 PG_Scan PG_SlowStop PLabel PLabel\$関数 PlaneClr PlaneDef Plane Plane 関数 PList PLocal PLocal 関数 PNumber 関数 PosFound 関数 PTCLR PTPBoostOK 関数 PTPTIME 関数 PTran PTRQ PTRQ 関数 QPDECELR QPDECELR 関数 QPDECELS QPDECELS 関数 RadToDeg 関数 Randomize ReadBin Read RealPls 関数 RealPos 関数 RealTorque 関数 RecoverPos 関数 Recover

EPSON RC+7.0 のバージョン	EPSON RC+ 4.0	
Ver.7.0.0	Redim RobotInfo 関数 RobotInfo\$関数 RobotModel\$関数 RobotName\$関数 RobotSerial\$関数 RobotType 関数 SafetyOn 関数 SelectDB SetCom SetInW SetIn SetNet SetSw Shutdown 関数 SingularityAngle SingularityAngle 関数 SingularitySpeed SingularitySpeed 関数 SoftCP SoftCP 関数 SpeedFactor SpeedFactor 関数 StartMain SyncRobots SyncRobots 関数 SysErr 関数 Tab\$関数 TargetOK 関数 TaskDone 関数 TaskInfo 関数 TaskInfo\$関数 TaskState TaskState 関数 TaskWait TC TCLim TCLim 関数 TCSpeed TCSpeed 関数 TeachOn 関数 TillOn 関数 TIDef 関数 Toff Ton UBound 関数 UpdateDB	VxCalib VxCalDelete VxCalLoad VxCalInfo 関数 VxCalSave VxTrans 関数 WaitNet WaitPos Where WindosStatus 関数 WriteBin Write XYLimClr XYLimDef XY 関数

C-3: EPSON RC+ 7.0 のバージョンごとに削除されたコマンド一覧

EPSON RC+ 6.0, 5.0, 4.0 に対して削除されたコマンド

EPSON RC+7.0 のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0	EPSON RC+ 4.0
Ver.7.1.2	SetLCD	SetLCD	SetLCD
Ver.7.0.0	Dir Type	-	Dir Type

Appendix D: 定数

SPEL プログラム中で使用可能な、定数があります。プロジェクトビルドでは、定数値が使用されます。

定数名	値	用法
TRUE	-1	論理式
FALSE	0	論理式
High	1	
Low	0	
Off	0	
On	1	
Above	1	
Below	2	
NoFlip	1	
Flip	2	
Righty	1	
Lefty	2	
J1	1	
J2	2	
J3	4	
J4	8	
J5	16	
J6	32	
J7	64	
CR	CHR\$(13)	
CRLF	CHR\$(13) + CHR\$(10)	
LF	CHR\$(10)	
MB_OK	0	MsgBox フラグ
MB_OKCANCEL	1	MsgBox フラグ
MB_ABORTRETRYIGNORE	2	MsgBox フラグ
MB_YESNOCANCEL	3	MsgBox フラグ
MB_YESNO	4	MsgBox フラグ
MB_RETRYCANCEL	5	MsgBox フラグ
MB_ICONSTOP	16	MsgBox フラグ
MB_ICONQUESTION	32	MsgBox フラグ
MB_ICONEXCLAMATION	48	MsgBox フラグ
MB_ICONINFORMATION	64	MsgBox フラグ
MB_DEFBUTTON1	0	MsgBox フラグ
MB_DEFBUTTON2	256	MsgBox フラグ
IDOK	1	MsgBox 戻り値
IDCANCEL	2	MsgBox 戻り値
IDABORT	3	MsgBox 戻り値
IDRETRY	4	MsgBox 戻り値
IDIGNORE	5	MsgBox 戻り値
IDYES	6	MsgBox 戻り値
IDNO	7	MsgBox 戻り値
BACKCOLORMODE_VISUALSTYLE	0	GUI Builder 用
BACKCOLORMODE_USER	1	GUI Builder 用
BORDERSTYLE_NONE	0	GUI Builder 用
BORDERSTYLE_FIXEDSINGLE	1	GUI Builder 用
BORDERSTYLE_FIXED3D	2	GUI Builder 用
CNV_QUELEN_ALL	0	Cnv_QueLen

定数名	値	用法
CNV_QUELEN_UPSTREAM	1	Cnv_QueLen
CNV_QUELEN_PICKUPAREA	2	Cnv_QueLen
CNV_QUELEN_DOWNSTREAM	3	Cnv_QueLen
DEVID_SELF	21	CLS
DEVID_TP	24	CLS
DEVID_TP3	30	CLS
DIALOGRESULT_NOE	0	GUI Builder 用
DIALOGRESULT_OK	1	GUI Builder 用
DIALOGRESULT_CANCEL	2	GUI Builder 用
DLG_IOMON	102	RunDialog
DLG_ROBOTMNG	100	RunDialog
DLG_VGUIDE	110	RunDialog
DOCK_NONE	0	GUI Builder 用
DOCK_TOP	1	GUI Builder 用
DOCK_BOTTOM	2	GUI Builder 用
DOCK_LEFT	3	GUI Builder 用
DOCK_RIGHT	4	GUI Builder 用
DOCK_FILL	5	GUI Builder 用
DROPDOWNSTYLE_SIMPLE	0	GUI Builder 用
DROPDOWNSTYLE_DROPDOWN	1	GUI Builder 用
DROPDOWNSTYLE_DROPDOWNLIST	2	GUI Builder 用
ERROR_DOINGMOTION	2999	GUI Builder 用
ERROR_NOMOTION	2998	GUI Builder 用
EVENTTASKTYPE_NORMAL	0	GUI Builder 用
EVENTTASKTYPE_NOPAUSE	1	GUI Builder 用
EVENTTASKTYPE_NOEMGABORT	2	GUI Builder 用
FORCE_LESS	0	Force_SetTrigger
FORCE_GREATER	1	Force_SetTrigger
FORCE_XFORCE	2	Force_SetTrigger
FORCE_YFORCE	3	Force_SetTrigger
FORCE_ZFORCE	4	Force_SetTrigger
FORCE_XTORQUE	5	Force_SetTrigger
FORCE_YTORQUE	6	Force_SetTrigger
FORCE_ZTORQUE	7	Force_SetTrigger
FORMBORDERSTYLE_NONE	0	GUI Builder 用
FORMBORDERSTYLE_FIXEDSINGLE	1	GUI Builder 用
FORMBORDERSTYLE_FIXED3D	2	GUI Builder 用
FORMBORDERSTYLE_FIXEDDIALOG	3	GUI Builder 用
FORMBORDERSTYLE_SIZABLE	4	GUI Builder 用
IMAGEALIGN_TOLEFT	1	GUI Builder 用
IMAGEALIGN_TOPCENTER	2	GUI Builder 用
IMAGEALIGN_TOPRIGHT	3	GUI Builder 用
IMAGEALIGN_MIDDLELEFT	4	GUI Builder 用
IMAGEALIGN_MIDDLECENTER	5	GUI Builder 用
IMAGEALIGN_MIDDLERIGHT	6	GUI Builder 用
IMAGEALIGN_BOTTOMLEFT	7	GUI Builder 用
IMAGEALIGN_BOTTOMCENTER	8	GUI Builder 用
IMAGEALIGN_BOTTOMRIGHT	9	GUI Builder 用
IOTYPE_INPUT	0	IOLabel 関数
IOTYPE_OUTPUT	1	IOLabel 関数
IOTYPE_MEMORY	2	IOLabel 関数
IOSIZE_BIT	1	IOLabel 関数

定数名	値	用法
IOSIZE_BYTE	8	IOLabel 関数
IOSIZE_WORD	16	IOLabel 関数
LANGID_ENGLISH	0	ErrMsg\$
LANGID_JAPANESE	1	ErrMsg\$
LANGID_GERMAN	2	ErrMsg\$
LANGID_FRENCH	3	ErrMsg\$
LANGID_SIMPLIFIED_CHINESE	4	ErrMsg\$
LANGID_TRADITIONAL_CHINESE	5	ErrMsg\$
MODE_STANDARD	1	PerformMode
MODE_HIGH_SPEED	2	PerformMode
MODE_LOW_OSCILLATION	3	PerformMode
ORIENT_HORIZONTAL	0	GUI Builder 用
ORIENT_VERTICAL	1	GUI Builder 用
PROGRESSBAR_STYLE_BLOCKS	0	GUI Builder 用
PROGRESSBAR_STYLE_CONT	1	GUI Builder 用
PROGRESSBAR_STYLE_MARQUEE	2	GUI Builder 用
SCROLLBARS_NONE	0	GUI Builder 用
SCROLLBARS_HORIZ	1	GUI Builder 用
SCROLLBARS_VERT	2	GUI Builder 用
SCROLLBARS_BOTH	3	GUI Builder 用
SETLATCH_PORT_CU_0	24	SetLatch
SETLATCH_PORT_CU_1	25	SetLatch
SETLATCH_PORT_DU1_0	56	SetLatch
SETLATCH_PORT_DU1_1	57	SetLatch
SETLATCH_PORT_DU2_0	280	SetLatch
SETLATCH_PORT_DU2_1	281	SetLatch
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	SetLatch
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	SetLatch
SHUTDOWN_ALL	0	Shutdown
SHUTDOWN_RESTART	1	Shutdown
SHUTDOWN_EPSONRC	2	Shutdown
SING_NONE	0	AvoidSingularity
SING_THRU	1	AvoidSingularity
SING_THRUROT	2	AvoidSingularity
SING_VSD	3	AvoidSingularity
SING_AUTO	4	AvoidSingularity
SIZEMODE_NORMAL	0	GUI Builder 用
SIZEMODE_STRETCHIMAGE	1	GUI Builder 用
SIZEMODE_AUTOSIZE	2	GUI Builder 用
SIZEMODE_CENTERIMAGE	3	GUI Builder 用
SIZEMODE_ZOOM	4	GUI Builder 用
STARTPOSITION_MANUAL	0	GUI Builder 用
STARTPOSITION_CENTERSCREEN	1	GUI Builder 用
STARTPOSITION_CENTERPARENT	2	GUI Builder 用
TEXTALIGN_LEFT	1	GUI Builder 用
TEXTALIGN_CENTER	2	GUI Builder 用
TEXTALIGN_RIGHT	3	GUI Builder 用
TEXTALIGN_TOPLEFT	1	GUI Builder 用
TEXTALIGN_TOPCENTER	2	GUI Builder 用
TEXTALIGN_TOPRIGHT	3	GUI Builder 用
TEXTALIGN_MIDDLELEFT	4	GUI Builder 用
TEXTALIGN_MIDDLECENTER	5	GUI Builder 用

定数名	値	用法
TEXTALIGN_MIDDLERIGHT	6	GUI Builder 用
TEXTALIGN_BOTTOMLEFT	7	GUI Builder 用
TEXTALIGN_BOTTOMCENTER	8	GUI Builder 用
TEXTALIGN_BOTTOMRIGHT	9	GUI Builder 用
TICKSTYLE_NONE	0	GUI Builder 用
TICKSTYLE_TOPLEFT	1	GUI Builder 用
TICKSTYLE_BOTTOMRIGHT	2	GUI Builder 用
TICKSTYLE_BOTH	3	GUI Builder 用
VISION_SORT_NONE	0	Vision Guide 用
VISION_SORT_PIXELX	1	Vision Guide 用
VISION_SORT_PIXELY	2	Vision Guide 用
VISION_SORT_PIXELXY	3	Vision Guide 用
VISION_SORT_CAMERAX	4	Vision Guide 用
VISION_SORT_CAMERAY	5	Vision Guide 用
VISION_SORT_CAMERAXY	6	Vision Guide 用
VISION_SORT_ROBOTX	7	Vision Guide 用
VISION_SORT_ROBOTY	8	Vision Guide 用
VISION_SORT_ROBOTXY	9	Vision Guide 用
VISION_SIZETOFind_ANY	0	Vision Guide 用
VISION_SIZETOFind_LARGEST	1	Vision Guide 用
VISION_SIZETOFind_SMALLEST	2	Vision Guide 用
VISION_BACKCOLOR_NONE	0	Vision Guide 用
VISION_BACKCOLOR_BLACK	1	Vision Guide 用
VISION_BACKCOLOR_WHITE	2	Vision Guide 用
VISION_CAMORIENT_STANDALONE	1	Vision Guide 用
VISION_CAMORIENT_FIXEDDOWN	2	Vision Guide 用
VISION_CAMORIENT_FIXEDUP	3	Vision Guide 用
VISION_CAMORIENT_MOBILEJ2	4	Vision Guide 用
VISION_CAMORIENT_MOBILEJ4	5	Vision Guide 用
VISION_CAMORIENT_MOBILEJ5	6	Vision Guide 用
VISION_CAMORIENT_MOBILEJ6	7	Vision Guide 用
VISION_FOUNDCOLOR_LIGHTGREEN	1	Vision Guide 用
VISION_FOUNDCOLOR_DARKGREEN	2	Vision Guide 用
VISION_GRAPHICS_ALL	1	Vision Guide 用
VISION_GRAPHICS_POSONLY	2	Vision Guide 用
VISION_GRAPHICS_NONE	3	Vision Guide 用
VISION_OPERATION_OPEN	1	Vision Guide 用
VISION_OPERATION_CLOSE	2	Vision Guide 用
VISION_OPERATION_ERODE	3	Vision Guide 用
VISION_OPERATION_DILATE	4	Vision Guide 用
VISION_OPERATION_SMOOTH	5	Vision Guide 用
VISION_OPERATION_SHARPEN1	6	Vision Guide 用
VISION_OPERATION_SHARPEN2	7	Vision Guide 用
VISION_OPERATION_HORIZEDGE	8	Vision Guide 用
VISION_OPERATION_VERTEDGE	9	Vision Guide 用
VISION_OPERATION_EDGEDETECT1	10	Vision Guide 用
VISION_OPERATION_EDGEDETECT2	11	Vision Guide 用
VISION_OPERATION_LAPLACE1	12	Vision Guide 用
VISION_OPERATION_LAPLACE2	13	Vision Guide 用
VISION_OPERATION_THIN	14	Vision Guide 用

定数名	値	用法
VISION_OPERATION_THICKEN	15	Vision Guide 用
VISION_OPERATION_BINARIZE	16	Vision Guide 用
VISION_OPERATION_ROTATE	17	Vision Guide 用
VISION_OPERATION_FLIPHORIZ	18	Vision Guide 用
VISION_OPERATION_FLIPVERT	19	Vision Guide 用
VISION_OPERATION_FLIPBOTH	20	Vision Guide 用
VISION_OPERATION_COLORFILTER	21	Vision Guide 用
VISION_OPERATION_SUBTRACTABS	22	Vision Guide 用
VISION_OPERATION_ZOOM	23	Vision Guide 用
VISION_ACQUIRE_NONE	0	Vision Guide 用
VISION_ACQUIRE_STATIONARY	1	Vision Guide 用
VISION_ACQUIRE_STROBED	2	Vision Guide 用
VISION_TRIGGERMODE_LEADINGEDGE	1	Vision Guide 用
VISION_TRIGGERMODE_TRAILINGEDGE	2	Vision Guide 用
VISION_THRESHCOLOR_BLACK	1	Vision Guide 用
VISION_THRESHCOLOR_WHITE	2	Vision Guide 用
VISION_OBJTYPE_CORRELATIO	1	Vision Guide 用
VISION_OBJTYPE_BLOB	2	Vision Guide 用
VISION_OBJTYPE_EDGE	3	Vision Guide 用
VISION_OBJTYPE_POLAR	4	Vision Guide 用
VISION_OBJTYPE_LINE	5	Vision Guide 用
VISION_OBJTYPE_POINT	6	Vision Guide 用
VISION_OBJTYPE_FRAME	7	Vision Guide 用
VISION_OBJTYPE_IMAGEOP	8	Vision Guide 用
VISION_OBJTYPE_OCR	9	Vision Guide 用
VISION_OBJTYPE_CODEREADER	10	Vision Guide 用
VISION_OBJTYPE_GEOMETRIC	11	Vision Guide 用
VISION_DETAILLEVEL_MEDIUM	1	Vision Guide 用
VISION_DETAILLEVEL_HIGH	2	Vision Guide 用
VISION_DETAILLEVEL_VERYHIGH	3	Vision Guide 用
VISION_IMAGESOURCE_CAMERA	1	Vision Guide 用
VISION_IMAGESOURCE_FILE	2	Vision Guide 用
VISION_CODETYPE_AUTO	0	Vision Guide 用
VISION_CODETYPE_EAN13	2	Vision Guide 用
VISION_CODETYPE_CODE39	3	Vision Guide 用
VISION_CODETYPE_INTERLEAVED25	4	Vision Guide 用
VISION_CODETYPE_CODE128	5	Vision Guide 用
VISION_CODETYPE_CODABAR	6	Vision Guide 用
VISION_CODETYPE_PDF417	8	Vision Guide 用
VISION_CODETYPE_QR	10	Vision Guide 用
VISION_CODETYPE_EAN8	13	Vision Guide 用
VISION_CODETYPE_UPCA	18	Vision Guide 用
VISION_CODETYPE_UPCE	19	Vision Guide 用
VISION_CODETYPE_UPC	20	Vision Guide 用
VISION_EDGETYPE_SINGLE	1	Vision Guide 用
VISION_EDGETYPE_PAIR	2	Vision Guide 用
VISION_IMAGECOLOR_ALL	1	Vision Guide 用
VISION_IMAGECOLOR_RED	2	Vision Guide 用
VISION_IMAGECOLOR_GREEN	3	Vision Guide 用
VISION_IMAGECOLOR_BLUE	4	Vision Guide 用

定数名	値	用法
VISION_IMAGECOLOR_GRAYSCALE	5	Vision Guide 用
VISION_POINTTYPE_POINT	0	Vision Guide 用
VISION_POINTTYPE_ENDPOINT	1	Vision Guide 用
VISION_POINTTYPE_MIDPOINT	2	Vision Guide 用
VISION_POINTTYPE_PERPTOLINE	3	Vision Guide 用
VISION_POINTTYPE_STARTPOINT	4	Vision Guide 用
VISION_POINTTYPE_PERPTOSTARTPOINT	5	Vision Guide 用
VISION_POINTTYPE_PERPTOMIDPOINT	6	Vision Guide 用
VISION_POINTTYPE_PERPTOENDPOINT	7	Vision Guide 用
VISION_REFTYPE_TAUGHTPOINTS	1	Vision Guide 用
VISION_REFTYPE_UPWARDCAMERA	2	Vision Guide 用
VISION_IMAGESIZE_320X240	1	Vision Guide 用
VISION_IMAGESIZE_640X480	2	Vision Guide 用
VISION_IMAGESIZE_800X600	3	Vision Guide 用
VISION_IMAGESIZE_1024X768	4	Vision Guide 用
VISION_IMAGESIZE_1280X1024	5	Vision Guide 用
VISION_IMAGESIZE_1600X1200	6	Vision Guide 用
VISION_IMAGESIZE_2048X1536	7	Vision Guide 用
VISION_IMAGESIZE_2560X1920	8	Vision Guide 用
VISION_WINTYPE_RECTANGLE	1	Vision Guide 用
VISION_WINTYPE_ROTATEDRECT	2	Vision Guide 用
VISION_WINTYPE_CIRCLE	3	Vision Guide 用
VISION_ORIENT_BOTH	1	Vision Guide 用
VISION_ORIENT_HORIZ	2	Vision Guide 用
VISION_ORIENT_VERT	3	Vision Guide 用
VISION_DIRECTION_INSIDEOUT	1	Vision Guide 用
VISION_DIRECTION_OUTSIDEIN	2	Vision Guide 用
VISION_POLARITY_DARK	1	Vision Guide 用
VISION_POLARITY_LIGHT	2	Vision Guide 用
VISION_PASSTYPE_SOMEFOUND	1	Vision Guide 用
VISION_PASSTYPE_ALLFOUND	2	Vision Guide 用
VISION_PASSTYPE_SOMENOTFOUND	3	Vision Guide 用
VISION_PASSTYPE_ALLNOTFOUND	4	Vision Guide 用
WIN_IOMON	-1	GUI Builder 用
WIN_TASKMGR	-2	GUI Builder 用
WIN_FORCEMON	-3	GUI Builder 用
WIN_SIMULATOR	-4	GUI Builder 用
WINDOWSTATE_NORMAL	0	WindowsStatus
WINDOWSTATE_MINIMIZED	1	WindowsStatus
WINDOWSTATE_MAXIMIZED	2	WindowsStatus
WithMove	0	Recover
WithoutMove	1	Recover
DRYRUNOFF	1	SF_GetParam 関数
SLS_1_HAND_EN	2	SF_GetParam 関数
SLS_1_SPEED	3	SF_GetParam 関数
SLS_1_ELBOW_EN	4	SF_GetParam 関数
SLS_1_JOINT_EN	5	SF_GetParam 関数
SLS_1_JOINTSPEED	6	SF_GetParam 関数
SLS_1_WRIST_EN	7	SF_GetParam 関数
SLS_1_SHOULDER_EN	8	SF_GetParam 関数

定数名	値	用法
SLS_2_HAND_EN	9	SF_GetParam 関数
SLS_2_SPEED	10	SF_GetParam 関数
SLS_2_ELBOW_EN	11	SF_GetParam 関数
SLS_2_JOINT_EN	12	SF_GetParam 関数
SLS_2_JOINTSPEED	13	SF_GetParam 関数
SLS_2_WRIST_EN	14	SF_GetParam 関数
SLS_2_SHOULDER_EN	15	SF_GetParam 関数
SLS_3_HAND_EN	16	SF_GetParam 関数
SLS_3_SPEED	17	SF_GetParam 関数
SLS_3_ELBOW_EN	18	SF_GetParam 関数
SLS_3_JOINT_EN	19	SF_GetParam 関数
SLS_3_JOINTSPEED	20	SF_GetParam 関数
SLS_3_WRIST_EN	21	SF_GetParam 関数
SLS_3_SHOULDER_EN	22	SF_GetParam 関数
SLS_T2_HAND_EN	23	SF_GetParam 関数
SLS_T2_SPEED	24	SF_GetParam 関数
SLS_T2_ELBOW_EN	25	SF_GetParam 関数
SLS_T2_JOINT_EN	26	SF_GetParam 関数
SLS_T2_JOINTSPEED	27	SF_GetParam 関数
SLS_T2_WRIST_EN	28	SF_GetParam 関数
SLS_T2_SHOULDER_EN	29	SF_GetParam 関数
SLS_T_SPEED	30	SF_GetParam 関数
SLS_T_JOINT_EN	31	SF_GetParam 関数
SLS_T_JOINTSPEED	32	SF_GetParam 関数
SLS_HAND_OFS_X	33	SF_GetParam 関数
SLS_HAND_OFS_Y	34	SF_GetParam 関数
SLS_HAND_OFS_Z	35	SF_GetParam 関数
SLS_1_DELAY	36	SF_GetParam 関数
SLS_2_DELAY	37	SF_GetParam 関数
SLS_3_DELAY	38	SF_GetParam 関数
SLS_JOINT_POS_EN	39	SF_GetParam 関数
SLS_JOINT_POS_ANGLE	40	SF_GetParam 関数
SLP_A_XU_EN	41	SF_GetParam 関数
SLP_A_XU_POS	42	SF_GetParam 関数
SLP_A_XL_EN	43	SF_GetParam 関数
SLP_A_XL_POS	44	SF_GetParam 関数
SLP_A_YU_EN	45	SF_GetParam 関数
SLP_A_YU_POS	46	SF_GetParam 関数
SLP_A_YL_EN	47	SF_GetParam 関数
SLP_A_YL_POS	48	SF_GetParam 関数
SLP_A_ZU_EN	49	SF_GetParam 関数
SLP_A_ZU_POS	50	SF_GetParam 関数
SLP_A_ZL_EN	51	SF_GetParam 関数
SLP_A_ZL_POS	52	SF_GetParam 関数
SLP_B_XU_EN	53	SF_GetParam 関数
SLP_B_XU_POS	54	SF_GetParam 関数
SLP_B_XL_EN	55	SF_GetParam 関数
SLP_B_XL_POS	56	SF_GetParam 関数
SLP_B_YU_EN	57	SF_GetParam 関数
SLP_B_YU_POS	58	SF_GetParam 関数

定数名	値	用法
SLP_B_YL_EN	59	SF_GetParam 関数
SLP_B_YL_POS	60	SF_GetParam 関数
SLP_B_ZU_EN	61	SF_GetParam 関数
SLP_B_ZU_POS	62	SF_GetParam 関数
SLP_B_ZL_EN	63	SF_GetParam 関数
SLP_B_ZL_POS	64	SF_GetParam 関数
SLP_C_XU_EN	65	SF_GetParam 関数
SLP_C_XU_POS	66	SF_GetParam 関数
SLP_C_XL_EN	67	SF_GetParam 関数
SLP_C_XL_POS	68	SF_GetParam 関数
SLP_C_YU_EN	69	SF_GetParam 関数
SLP_C_YU_POS	70	SF_GetParam 関数
SLP_C_YL_EN	71	SF_GetParam 関数
SLP_C_YL_POS	72	SF_GetParam 関数
SLP_C_ZU_EN	73	SF_GetParam 関数
SLP_C_ZU_POS	74	SF_GetParam 関数
SLP_C_ZL_EN	75	SF_GetParam 関数
SLP_C_ZL_POS	76	SF_GetParam 関数
SLP_J2_MON_RAD	77	SF_GetParam 関数
SLP_J3_MON_RAD	78	SF_GetParam 関数
SLP_J5_MON_RAD	79	SF_GetParam 関数
SLP_J6_MON_RAD	80	SF_GetParam 関数
SLP_J1_RANGE_MAX	81	SF_GetParam 関数
SLP_J1_RANGE_MIN	82	SF_GetParam 関数
SLP_J2_RANGE_MAX	83	SF_GetParam 関数
SLP_J2_RANGE_MIN	84	SF_GetParam 関数
SLP_J3_RANGE_MAX	85	SF_GetParam 関数
SLP_J3_RANGE_MIN	86	SF_GetParam 関数
SLP_J4_RANGE_MAX	87	SF_GetParam 関数
SLP_J4_RANGE_MIN	88	SF_GetParam 関数
SLP_J5_RANGE_MAX	89	SF_GetParam 関数
SLP_J5_RANGE_MIN	90	SF_GetParam 関数
SLP_J6_RANGE_MAX	91	SF_GetParam 関数
SLP_J6_RANGE_MIN	92	SF_GetParam 関数
SIN_1_SLS_1_EN	93	SF_GetParam 関数
SIN_1_SLS_2_EN	94	SF_GetParam 関数
SIN_1_SLS_3_EN	95	SF_GetParam 関数
SIN_1_SLP_A_EN	96	SF_GetParam 関数
SIN_1_SLP_B_EN	97	SF_GetParam 関数
SIN_1_SLP_C_EN	98	SF_GetParam 関数
SIN_1_SG_EN	99	SF_GetParam 関数
SIN_1_ESTOP_EN	100	SF_GetParam 関数
SIN_2_SLS_1_EN	101	SF_GetParam 関数
SIN_2_SLS_2_EN	102	SF_GetParam 関数
SIN_2_SLS_3_EN	103	SF_GetParam 関数
SIN_2_SLP_A_EN	104	SF_GetParam 関数
SIN_2_SLP_B_EN	105	SF_GetParam 関数
SIN_2_SLP_C_EN	106	SF_GetParam 関数
SIN_2_SG_EN	107	SF_GetParam 関数
SIN_2_ESTOP_EN	108	SF_GetParam 関数

定数名	値	用法
SIN_3_SLS_1_EN	109	SF_GetParam 関数
SIN_3_SLS_2_EN	110	SF_GetParam 関数
SIN_3_SLS_3_EN	111	SF_GetParam 関数
SIN_3_SLP_A_EN	112	SF_GetParam 関数
SIN_3_SLP_B_EN	113	SF_GetParam 関数
SIN_3_SLP_C_EN	114	SF_GetParam 関数
SIN_3_SG_EN	115	SF_GetParam 関数
SIN_3_ESTOP_EN	116	SF_GetParam 関数
SIN_4_SLS_1_EN	117	SF_GetParam 関数
SIN_4_SLS_2_EN	118	SF_GetParam 関数
SIN_4_SLS_3_EN	119	SF_GetParam 関数
SIN_4_SLP_A_EN	120	SF_GetParam 関数
SIN_4_SLP_B_EN	121	SF_GetParam 関数
SIN_4_SLP_C_EN	122	SF_GetParam 関数
SIN_4_SG_EN	123	SF_GetParam 関数
SIN_4_ESTOP_EN	124	SF_GetParam 関数
SIN_5_SLS_1_EN	125	SF_GetParam 関数
SIN_5_SLS_2_EN	126	SF_GetParam 関数
SIN_5_SLS_3_EN	127	SF_GetParam 関数
SIN_5_SLP_A_EN	128	SF_GetParam 関数
SIN_5_SLP_B_EN	129	SF_GetParam 関数
SIN_5_SLP_C_EN	130	SF_GetParam 関数
SIN_5_SG_EN	131	SF_GetParam 関数
SIN_5_ESTOP_EN	132	SF_GetParam 関数
SOUT_1_STO	133	SF_GetParam 関数
SOUT_1_SLS_1	134	SF_GetParam 関数
SOUT_1_SLS_2	135	SF_GetParam 関数
SOUT_1_SLS_3	136	SF_GetParam 関数
SOUT_1_SLS_T2	137	SF_GetParam 関数
SOUT_1_SLS_T	138	SF_GetParam 関数
SOUT_1_SLP_A	139	SF_GetParam 関数
SOUT_1_SLP_B	140	SF_GetParam 関数
SOUT_1_SLP_C	141	SF_GetParam 関数
SOUT_1_EP_RC	142	SF_GetParam 関数
SOUT_1_EP_TP	143	SF_GetParam 関数
SOUT_1_EN_SW	144	SF_GetParam 関数
SOUT_2_STO	145	SF_GetParam 関数
SOUT_2_SLS_1	146	SF_GetParam 関数
SOUT_2_SLS_2	147	SF_GetParam 関数
SOUT_2_SLS_3	148	SF_GetParam 関数
SOUT_2_SLS_T2	149	SF_GetParam 関数
SOUT_2_SLS_T	150	SF_GetParam 関数
SOUT_2_SLP_A	151	SF_GetParam 関数
SOUT_2_SLP_B	152	SF_GetParam 関数
SOUT_2_SLP_C	153	SF_GetParam 関数
SOUT_2_EP_RC	154	SF_GetParam 関数
SOUT_2_EP_TP	155	SF_GetParam 関数
SOUT_2_EN_SW	156	SF_GetParam 関数
SOUT_3_STO	157	SF_GetParam 関数
SOUT_3_SLS_1	158	SF_GetParam 関数

定数名	値	用法
SOUT_3_SLS_2	159	SF_GetParam 関数
SOUT_3_SLS_3	160	SF_GetParam 関数
SOUT_3_SLS_T2	161	SF_GetParam 関数
SOUT_3_SLS_T	162	SF_GetParam 関数
SOUT_3_SLP_A	163	SF_GetParam 関数
SOUT_3_SLP_B	164	SF_GetParam 関数
SOUT_3_SLP_C	165	SF_GetParam 関数
SOUT_3_EP_RC	166	SF_GetParam 関数
SOUT_3_EP_TP	167	SF_GetParam 関数
SOUT_3_EN_SW	168	SF_GetParam 関数
POS_ROT_U	169	SF_GetParam 関数
POS_ROT_V	170	SF_GetParam 関数
POS_ROT_W	171	SF_GetParam 関数
POS_OFS_X	172	SF_GetParam 関数
POS_OFS_Y	173	SF_GetParam 関数
POS_OFS_Z	174	SF_GetParam 関数
SF_TOOLVERSION	1	SF_GetParam\$関数
SF_CHECKSUM	2	SF_GetParam\$関数
SF_LAST_MODIFIED	3	SF_GetParam\$関数
SF_ROBOT_MODEL_NAME	4	SF_GetParam\$関数
SF_ROBOT_CHECKSUM	5	SF_GetParam\$関数
SF_HOFS	6	SF_GetParam\$関数
SF_HOFS_LAST_MODIFIED	7	SF_GetParam\$関数
SLS_1	1	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_2	2	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_3	3	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_T	9	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_T2	10	SF_LimitSpeedS, SF_LimitSpeedSEnable