OMRON

eV+3

キーワードリファレンスマニュアル

SBCE-487C

おことわり

- 本マニュアルの内容の一部または全部を無断で複写、複製、転載することを禁じます。
- 本マニュアルの内容に関しては、改良のため予告なしに仕様などを変更することがあります。
  あらかじめご了承ください。
- 本マニュアルの内容に関しては、万全を期しておりますが、万一ご不審な点や誤りなど、お気づきのことがありましたら、お手数ですが、弊社支店または営業所までご連絡ください。
  その際、巻末記載の Man.No.（マニュアル No.）も併せてお知らせください。

商標

- Microsoft、Windows 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。
- ODVA、CIP、DeviceNet、EtherNet/IP は ODVA の商標です。
- SD、SDHC ロゴは、SD-3C, LLC の商標です。

その他、本文中に掲載している会社名および製品名は、それぞれ各社の商標または登録商標です。

著作権

スクリーンショットはマイクロソフトの許可を得て使用しています。

表記についての留意事項

本マニュアルでは、「Automation Control Environment（オートメーション コントロール エンバイロメント）」をすべて「ACE」と略して記述しています。

ご承諾事項

「当社商品」について特別の合意がない場合には、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。

- **定義**

  本ご承諾事項中の用語の定義は次のとおりです。

  - 「当社商品」：「当社」の FA システム機器、汎用制御機器、センシング機器、電子・機構部品
  - 「カタログ等」：「当社商品」に関する、ベスト制御機器オムロン、電子・機構部品総合カタログ、その他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含みます。
  - 「利用条件等」：「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取り扱い方法、利用上の注意、禁止事項その他
  - 「お客様用途」：「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み、または利用を含みます。
  - 「適合性等」：「お客様用途」での「当社商品」の (a) 適合性、(b) 動作、(c) 第三者の知的財産の非侵害、(d) 法令の遵守および (e) 各種規格の遵守

- **記載事項のご注意**

  「カタログ等」の記載内容については次の点をご理解ください。

  - 定格値および性能値は、単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。
  - 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。
  - 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。
  - 「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

- **ご利用にあたってのご注意**

  ご採用およびご利用に際しては次の点をご理解ください。

  - 定格・性能ほか「利用条件等」を遵守しご利用ください。
  - お客様ご自身にて「適合性等」をご確認いただき、「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」を一切保証いたしかねます。
  - 「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で、必ず事前に確認してください。
  - 「当社商品」をご使用の際には、(i)定格および性能に対し余裕のある「当社商品」のご利用、冗長設計などの安全設計、(ii)「当社商品」が故障しても、「お客様用途」の危険を最小にする安全設計、(iii)利用者に危険を知らせるための、安全対策のシステム全体としての構築、(iv)「当社商品」および「お客様用途」の定期的な保守、の各事項を実施してください。
  - 「当社」はDDoS攻撃（分散型DoS攻撃）、コンピュータウイルスその他の技術的な有害プログラム、不正アクセスにより、「当社商品」、インストールされたソフトウェア、またはすべてのコンピュータ機器、コンピュータプログラム、ネットワーク、データベースが感染したとしても、そのことにより直接または間接的に生じた損失、損害その他の費用について一切責任を負わないものとします。
    お客様ご自身にて、（ⅰ）アンチウイルス保護、（ⅱ）データ入出力、（ⅲ）紛失デー

タの復元、（ⅳ）「当社商品」またはインストールされたソフトウェアに対するコンピュータウイルス感染防止、（ⅴ）「当社商品」に対する不正アクセス防止についての十分な措置を講じてください。

- 「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。ただし、次に掲げる用途であっても「当社」の意図した特別な商品用途の場合や特別の合意がある場合は除きます。
  - (a) 高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、娯楽設備、医用機器、安全装置、その他生命・身体に危険が及びうる用途）
  - (b) 高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24 時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
  - (c) 厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
  - (d) 「カタログ等」に記載のない条件や環境での用途
- 上記の (a) から (d) に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないで下さい。自動車搭載用商品については当社営業担当者にご相談ください。

- **保証条件**

  「当社商品」の保証条件は次のとおりです。

  - 保証期間 ご購入後 1 年間といたします。
    （ただし「カタログ等」に別途記載がある場合を除きます。）
  - 保証内容 故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
    - (a) 当社保守サービス拠点における故障した「当社商品」の無償修理
      （ただし、電子・機構部品については、修理対応は行いません。）
    - (b) 故障した「当社商品」と同数の代替品の無償提供
  - 保証対象外 故障の原因が次のいずれかに該当する場合は、保証いたしません。
    - (a) 「当社商品」本来の使い方以外のご利用
    - (b) 「利用条件等」から外れたご利用
    - (c) 本ご承諾事項「ご利用にあたってのご注意」に反するご利用
    - (d) 「当社」以外による改造、修理による場合
    - (e) 「当社」以外の者によるソフトウェアプログラムによる場合
    - (f) 「当社」からの出荷時の科学・技術の水準では予見できなかった原因
    - (g) 上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

- **責任の制限**

  本ご承諾事項に記載の保証が、「当社商品」に関する保証のすべてです。

  「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

- **輸出管理**

  「当社商品」または技術資料を、輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。お客様が法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。

労働安全衛生規則第 36 条第31 号および32 号において、産業用ロボットの教示等を行う作業や検査・修理・調整等を行う作業は、労働安全衛生法で定める「危険又は有害な業務」に該当します。労働安全衛生法第59 条により、事業者は労働者に対し「安全又は衛生のための特別の教育」を行う義務があります。

お客様の責任において、リスクアセスメントの結果から適切な安全対策を実施してください。
ロボットセーフティガイドおよび弊社ロボットシステム商品関連資料の記載事項を全て遵守しても、産業用ロボットが引き起こす人身事故または機器の損傷を回避する保証はできません。

目次

第1章: はじめに 19

1.1 対象となる読者 19
1.2 関連マニュアル 20
1.3 警告のレベル 20
1.4 特記事項 20
1.5 キーワードの構文 21
1.6 キーワードパラメータ 21
パラメータのデータタイプの指定 22
数値パラメータ 22

第2章: キーワードクイックリファレンス 23

2.1 関数キーワード概要 23
2.2 モニタコマンドキーワード概要 30
2.3 他のキーワード概要 34
2.4 プログラム命令キーワード概要 34
2.5 システムパラメータキーワード参照 42
2.6 システムスウィッチキーワード概要 42

第3章: キーワードの詳細 45

3.1 関数キーワード 45
ABS 45
ACCEL 46
ACOS 47
ALIGN 49
ASC 50
ASIN 51
ATAN2 53
BASE 54
BCD 55
BELT 56
BITS 58
BMASK 60
BSTATUS 61

CAS 63
$CHR 64
COM 65
CONFIG 66
COS 70
CUBRT 72
$DBLB 73
DBLB 74
DCB 76
$DECODE 77
$DEFAULT 80
DEFINED 82
DEST 83
DEVICE 85
DISTANCE 87
DURATION 88
DX 90
DY 91
DZ 92
ENCLATCH 93
$ENCODE 94
$ERROR 96
ERROR 97
FALSE 102
$FLTB 103
FLTB 104
FRACT 106
FRAME 107
FREE 108
GETC 109
GET.EVENT 111
HERE 112
HOUR.METER 113
$ID 114
ID 115
IDENTICAL 125
INRANGE 126
$INTB 128
INT 129
INTB 130

INVERSE 132
IOSTAT 133
LAST 136
LATCH 138
LATCHED 139
LEN 141
$LNGB 142
LNGB 143
MAX 145
$MID 146
MIN 147
NETWORK 148
NORMAL 150
NOT 151
NULL 152
OFF 153
ON 154
OUTSIDE 155
PARAMETER 156
#PDEST 157
#PHERE 158
PI 159
#PLATCH 160
POS 161
#PPOINT 162
PRIORITY 164
RANDOM 165
RX 166
RY 167
RZ 168
SCALE 169
SELECT 170
#SET.POINT 172
SHIFT 173
SIG.INS 174
SIGN 176
SIG 177
SIN 179
SOLVE.FLAGS 180
SPEED 182

SQRT 184
SQR 185
STATE 186
STATUS 194
STRDIF 196
SWITCH 197
TAN 198
TAS 199
TASK 202
$TIME 205
$TIME4 207
TIME 209
TIMER 211
TOOL 214
TPS 215
TRANS 216
$TRANSB 218
TRANSB 219
TRUE 220
$TRUNCATE 221
$UNPACK 222
VAL 224
VLOCATION 225
VPARAMETER 228
VRESULT 230
VSTATE 232
WINDOW 233
3.2 モニタコマンドキーワード 236
ABORT 236
BASE 238
BITS 240
CALIBRATE 242
CD 245
COMMANDS 246
COPY 248
CYCLE.END 249
DEFAULT 251
DELETE 255
DELETEL 257
DELETEM 259

DELETEP 261
DELETER 263
DELETES 265
DIRECTORY 267
DISABLE 269
DO 271
ENABLE 273
ESTOP 275
EXECUTE 276
FCOPY 279
FDELETE 281
FDIRECTORY 283
FLIST 287
FREE 288
FRENAME 289
FSET 290
HERE 291
ID 293
IO 296
JOG 298
KILL 302
LIST 303
LISTL 305
LISTP 307
LISTR 308
LISTS 310
LOAD 312
MDIRECTORY 315
MODULE 317
NET 319
PANIC 322
PARAMETER 323
PING 325
PRIME 326
PROCEED 328
RENAME 330
RESET 331
RESET.LOCK 332
RETRY 333
SELECT 335

SIGNAL 337
SPEED 339
SRV.RESET 341
SSTEP 342
STACK 344
STATUS 346
STORE 350
STOREL 352
STOREM 354
STOREP 356
STORER 358
STORES 360
SWITCH 362
TESTP 364
TOOL 365
WAIT.START 367
WHERE 368
XSTEP 369
ZERO 372
3.3 その他のキーワード 374
.END 374
IPS 375
MMPS 376
3.4 プログラムコマンドキーワード 377
ABORT 377
ABOVE 379
ACCEL 381
ALIGN 384
ALTER 385
ALTOFF 386
ALTON 387
ANY 389
APPRO 390
APPROS 391
ATTACH 392
AUTO 396
BASE 399
BELOW 401
BITS 403
BRAKE 405

BREAK 406
BY 407
CALIBRATE 408
CALL 412
CALLS 415
CASE 417
CLEAR.EVENT 420
CLEAR.LATCHES 421
COARSE 422
CPOFF 424
CPON 426
CYCLE.END 428
DECOMPOSE 430
DEF.DIO 432
DEFBELT 434
DEPART 436
DEPARTS 438
DETACH 440
DISABLE 442
DO 444
DOS 446
DRIVE 448
DURATION 450
ELSE 452
ENABLE 453
END 455
ESTOP 457
EXECUTE 458
EXIT 462
EXTERNAL 463
FCLOSE 465
FCMND 466
FCOPY 470
FDELETE 472
FEMPTY 474
FINE 476
FLIP 478
FOPEN 481
FOPENA 484
FOPEND 487

FOPENR 490
FOPENW 493
FOR 496
FSEEK 498
FSET 500
GLOBAL 502
GOTO 504
HALT 505
HERE 506
IF ... GOTO 508
IF ...THEN 509
IGNORE 511
JMOVE 512
JOG 513
KEYMODE 517
KILL 519
LEFTY 520
LOCAL 522
LOCK 524
MC 526
MCS 528
MOVE 530
MOVEC 531
MOVES 538
MULTIPLE 539
NEXT 540
NOFLIP 541
NONULL 542
NOOVERLAP 544
NULL 546
OVERLAP 548
PACK 550
PANIC 552
PARAMETER 553
PAUSE 555
PDNT.CLEAR 556
PDNT.NOTIFY 557
PDNT.WRITE 558
PENDANT 560
PROCEED 563

.PROGRAM 565
PROMPT 568
REACT 570
REACTE 573
REACTI 575
READ 577
READY 581
RELEASE 583
RESET 584
RETRY 585
RETURN 587
RETURNE 588
RIGHTY 589
RUNSIG 591
SELECT 593
SET.EVENT 595
SET 596
SETBELT 598
SETDEVICE 600
SIGNAL 602
SINGLE 604
SOLVE.ANGLES 605
SOLVE.TRANS 612
SPEED 614
STOP 617
SWITCH 618
TIMER 620
TOOL 621
TYPE 622
UNTIL 625
VALUE 626
VPARAMETER 627
VRUN 629
VWAITI 630
WAIT 631
WAIT.EVENT 632
WHILE 635
WINDOW 637
WRITE 639
3.5 システムパラメータキーワード 641

BELT.MODE 641
DEVIATION 643
JOG.TIME 644
NOT.CALIBRATED 645
VTIMEOUT 647
3.6 システムスイッチキーワード 648
AUTO.POWER.OFF 648
CP 650
DECEL.100 651
DELAY.IN.TOL 652
DRY.RUN 654
MESSAGES 656
OBSTACLE 657
POWER 659
ROBOT 661
SCALE.ACCEL 663
SCALE.ACCEL.ROT 665
UPPER 666

改訂履歴

改訂コード	日付	改訂内容
A	2020年7月	初版
B	2020年10月	Viperロボットサポート項目の追記
C	2021年01月	iX3およびiX4ロボットサポート項目の追記

第1章: はじめに

eV+ は、独自のプログラミング言語とコマンドセットを使用し、オペレーティングシステムに対して情報を送受信します。本マニュアルで詳しく説明されるキーワードは、Sysmac Studio でプログラムを作成したり、Sysmac Studio インタフェースのモニタウィンドウからコマンドを発行する場合に使用されます。

> **追加情報:** キーワードの構文、パラメータ、データの種類の詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。
>
> **重要:**ホストシステムとして NJ シリーズロボット統合CPU ユニット を含まないシステムでキーワードを発行した場合、エラー"Command not supported"が返されます。

以下に説明するように、機能ごとに 6 種類のキーワードがあります。

表1-1. eV+ キーワードの種類

キーワードの種類	使用法
関数 キーワード	eV+ オペレーティングシステムから値を返すために使用します。
モニタコマンドキーワード	モニタウィンドウで個々の演算を実行したり、モニタコマンドプログラムを作成したりするために使用します。
プログラム コマンドキーワード	V+ プログラムでコマンド演算するために使用します。
システム パラメータキーワード	V+ プログラムまたはモニタウィンドウでシステムパラメータを操作するために使用します。
システム スイッチキーワード	V+ プログラムまたはモニタウィンドウでシステムスイッチを操作するために使用します。
その他のキーワード	SPEED プログラムのコマンドキーワードを使用するときの単位指定に使用します。 詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

1.1 対象となる読者

このマニュアルの対象読者は、次の職種であって、一般的なプログラミング技法およびロボット制御方法に対する知識も有している方です。

- FA システムの導入担当者
- FA システムの設計担当者
- FA システムの設置/保守担当者
- FA システムおよび施設の管理担当者

1.2 関連マニュアル

参考資料として次の関連マニュアルを使用します。

表1-2. 関連マニュアル

マニュアル	説明
『eV+ 3 ユーザーズマニュアル (SBCE-486)』	eV+ を使用するために必要な情報が記載されています。
『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』	ロボット統合CPU ユニットを使用したロボット統合システムを設定するための、Sysmac Studio の操作手順や機能を説明します。
ロボットユーザーズマニュアル	使用するロボットのユーザーズマニュアルです。
『ティーチングペンダントT20 ユーザーズマニュアル (SBCE-392)』	オプションのT20手動制御ペンダントの使用方法について記載されています。
『NJ シリーズ ロボット統合 CPU ユニット ユーザーズマニュアル (SBCE-436)』	CPUユニットの設定、動作、およびオムロン製ロボットの制御に関するプログラミングコンセプトについて記載されています。
『ロボットセーフティガイド (SBCE-381)』	オムロン製産業用ロボットの安全に関する情報が記載されています。

1.3 警告のレベル

この文書を通じて、以下の警告レベルが使用されています。



危険: 正しい取扱いをしなければ、この危険のために、死亡に至ったり重傷を負ったりする差し迫った危険性があります。また、同様に重大な物的損害をもたらす恐れがあります。



警告: 正しい取扱いをしなければ、この危険のために、軽傷・中程度の障害を負ったり、万一の場合には重傷や死亡、あるいは重大な物的損害をもたらす恐れがあります。



注意: 正しい取扱いをしなければ、この危険のために、軽傷・中程度の障害を負ったり、あるいは物的損害をもたらす恐れがあります 。

1.4 特記事項

この文書では特記事項を示すために次の表記を使用しています。

重要:製品を確実に安全利用するための情報です。

注: 製品をより効果的に利用するための情報です。

追加情報: 役に立つヒント、推奨事項、およびベストプラクティスです。

1.5 キーワードの構文

本マニュアルに記載されているキーワードの構文は、以下のガイドラインに従っています。

- キーワードは大文字、パラメータは小文字で表されます。
- 丸括弧、角括弧、コンマは、示されたように正確に使用する必要があります。
- 太字は、必須の構文、太字ではない文字はオプションの構文を表します。
- 連続するパラメータの間は、コンマで区切る必要があります。
- システムは空白文字を評価しないため、余分な空白、または空白がなくても構文の問題にはなりません。

このマニュアルで使用するキーワード構文の書き方について、以下の例で説明します。

KEYWORD req_param1 = req_param2 SECONDARY.KEYWORD opt_param1, opt_param2

表1-3. キーワード構文の詳細

項目	詳細
KEYWORD	必須であり、示されたように正確に入力する必要があります。
req_param1	必須であり、値、変数、または式で置き換える必要があります。
=	代入演算子。必要な場合はキーワードおよびまたはパラメータに続ける必要があります。
req_param2	イコール記号がある場合必須であり、値、変数、または式で置き換える必要があります。
SECONDARY.KEYWORD	このキーワードがある場合は、必須であり、示されたように正確に入力する必要があります
opt_param1, opt_param2	オプションです。使用する場合は、値、変数、または式で置き換える必要があります。

1.6 キーワードパラメータ

本項では、パラメータと、キーワードの構文でどのようにパラメータを使用するかを説明します。

> **注:** キーワードパラメータとシステムパラメータキーワードを混同しないように注意してください。詳細については システムパラメータキーワード 641 ページを参照してください。

パラメータは、キーワードに対する引数です。キーワードの中には、複数の必須およびオプションのパラメータを持つキーワードもあります。BREAK や FALSE などのように、一部のキーワードはパラメータを使用しません。

オプションのパラメータが指定されていない場合、システムはデフォルトのパラメータを使用します。以下に示すように、キーワードの構文の中央にあるオプションパラメータを指定しない場合は、必要なだけのすべてのコンマを使用する必要があります。

```
CALL program_name(1, , n+3)
```

オプションのパラメータが最後の位置にある場合は、不要であれば削除できます。**CALL program**(arg_list)という構文構造で示される CALL キーワードについて考えてみましょう。引数を指定せずに、このキーワードで新しいサブルーチンプログラムを実行する場合、構文は、以下のようになります。

```
CALL program_name
```

パラメータのデータタイプの指定

定数または変数のデータタイプはキーワードで必要とされるデータタイプと同じでなければなりません。文字列、数値パラメータには、定数または有効な変数名を使用できます。以下のルールに従って、キーワードの構文で使用するパラメータのデータタイプを指定します。

- 文字列変数の先頭には$記号を付けて、引用符で文字列定数を囲む必要があります。
- プレシジョンポイント変数の先頭には#記号を付ける必要があります。
- ベルト変数の先頭には % 記号を付ける必要があります。
- 実数の定数と整数の定数はそのまま使用できます。

追加情報: キーワードを変数名やプログラム名に使用することはできません。

数値パラメータ

複数のタイプの数値パラメータが、キーワード構文に含まれる場合があります。パラメータの各タイプでは、一般的に、数値定数、変数名、または数式で値を指定します。eV+ が扱う数値にはいくつかの制限があります。次のルールによって、さまざまな状況でどのように値が解釈されるかが決定されます。

- 距離は、ロボットが移動する位置を定義するために使用されます。値に対して単位を明示的に入力することはありませんが、距離の測定単位はミリメートルです。距離として入力する値は、プラスまたはマイナスで指定できます。ロボット速度を inch/s で指定する特殊な場合については、 IPS キーワードを参照してください。
- ジョイント数は、1 からロボットのジョイント数までの整数です。ロボットのジョイントのナンバリングについては、ロボットユーザーズマニュアルを参照してください。
- 信号番号はデジタル信号の識別に使用されます。必ず整数値を使用してください。マイナスの信号番号は OFF 状態を示します。特定のロボットの信号番号については、適切なロボットユーザーズマニュアルおよび*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
- 整数パラメータには実数値（整数部と小数部を含む値）を使用できます。整数が必要な場合、値は四捨五入され、結果の整数部が使用されます。
- スカラー変数として示されるパラメータには、特別な指定がない限り実数値（整数部と小数部を含む値）を使用できます。スカラー変数の値の範囲は、$-9.22*10^{18}$ ～$9.22*10^{18}$（-9.22E18～9.22E18 として表示）です。倍精度値として宣言される数値の範囲は、 $-1.8*10^{-307}$ ～$1.8*10^{-307}$ です。

このマニュアルで使用するすべての数値は、特に断りのない限り、10 進数です。2 進数は ^B、8 進数は^、16 進数は^Hとして表されます。

第2章: キーワードクイックリファレンス

本書で説明するすべてのキーワードについては、クイックリファレンスとして以下の項を参照してください。

2.1 関数キーワード概要

すべての関数キーワードの概要を次の表に示しています。

キーワード	詳細	参照
ABS	指定された引数の絶対値（大きさ）を返す実数値関数です。	45 ページ
ACCEL	ロボットの設定プロファイルで定義されている加速、減速、または最大許容制限率（%）の現在の設定値を返す実数値関数です。	46 ページ
ACOS	引数のアークコサインを返す実数値関数です。	47 ページ
ALIGN	入力位置をワールド座標系の最も近い軸に合わせる変換関数です。	49 ページ
ASC	文字列内の ASCII 文字の値を返す実数値関数です。	50 ページ
ASIN	引数のアークサインを返す実数値関数です。	51 ページ
ATAN2	value_1/value_2 の値に等しいタンジェント値を持つ角度を返す実数値関数です。	53 ページ
BASE	最新のBASE プログラムのコマンドまたはモニタコマンドで設定された移動・回転を表す変換値を返す変換関数です。	54 ページ
BCD	実数値を 2 進化 10 進数（BCD）形式に変換する実数値関数です。	55 ページ
BELT	コンベア追跡機能で追跡されるコンベアベルトに関する情報を返す実数値関数です。	56 ページ
BITS	複数のデジタル信号を読み取り、バイナリビットパターンに対応する値を返す実数値関数です。	58 ページ
BMASK	個々のビットを指定してビットマスクを作成する実数値関数です。	60 ページ
BSTATUS	コンベア追跡システムの状態に関する情報を返す実数値関数です。	61 ページ
CAS	実数値変数をテスト値と比較し、一致していれば新しい値を実数値変数に設定する処理を、不可分に実行する実数値関数です。	63 ページ
$CHR	指定した ASCII 値に該当する 1 文字を返す文字関数です。	64 ページ
COM	値に対して 2 の補数演算を実行します。	65 ページ
CONFIG	ロボットの幾何学的設定または動作サーボ制御機能の状態に関する情報	66 ページ

キーワード	詳細	参照
	を示す値を返す実数値関数です。	
COS	指定された角度のコサイン値を返す実数値関数です。	70 ページ
CUBRT	値の立方根を返す実数値関数です。	72 ページ
$DBLB	実数値をIEEE 754規格の倍精度浮動小数点形式の2進数で表現した8 バイトの文字列を返す文字列関数です。	73 ページ
DBLB	8 バイトの文字列をIEEE 倍精度浮動小数点数として解釈した値を返す実数値関数です。	74 ページ
DCB	BCD を等しい整数値に変換する実数値関数です。	76 ページ
$DECODE	指定されたブレーク文字で決定される文字列の一部を抽出する文字列関数です。	77 ページ
$DEFAULT	現在または最初のシステムのデフォルトデバイス、ユニット、ディスクファイルにアクセスするためのディレクトリパスを含む文字列を返す文字列関数です。	80 ページ
DEFINED	変数が定義されているかを判定する実数値関数です。	82 ページ
DEST	現在のロボットモーションに対して、計画された目的地を示す変換値を返す変換関数です。	83 ページ
DEVICE	指定されたデバイスから取得した実数値を返す実数値関数です。値は、デバイスやパラメータに応じて、データ、ステータス情報になります。	85 ページ
DISTANCE	2 つのロケーション値で定義されるポイント間の距離を決定する実数値関数です。	87 ページ
DURATION	モーションDURATION 仕様の 1 つに対する現在の設定を返す実数値関数です。	88 ページ
DX	指定された変換値の X 軸の成分を返す実数値関数です。	90 ページ
DY	指定された変換値の Y 軸の成分を返す実数値関数です。	91 ページ
DZ	指定された変換値の Z 軸の成分を返す実数値関数です。	92 ページ
ENCLATCH	システムに含まれているエンコーダの、最新のラッチが発生した時点でのエンコーダ値を返す実数値関数です。	93 ページ
$ENCODE	出力仕様から作成された文字列を返す文字列関数です。	94 ページ
$ERROR	指定のエラーコードに関連するエラーメッセージを返す文字列関数です。	96 ページ
ERROR	プログラムの実行を停止させた、または REACTE 命令を発生させた、最近のシステムメッセージ番号を返す実数値関数です。	97 ページ

キーワード	詳細	参照
FALSE	eV+ で使用される、論理的な偽を表すための値を返す実数値関数です。	102 ページ
$FLTB	実数値を単精度IEEE浮動小数点形式の2 進数で表現した4 バイトの文字列を返す文字列関数です。	103 ページ
FLTB	4 バイトの文字列をIEEE 単精度浮動小数点数として解釈した値を返す実数値関数です。	104 ページ
FRACT	引数の小数部を返す実数値関数です。	106 ページ
FRAME	4 つの位置で定義される変換値を返す変換関数です。	107 ページ
FREE	記憶領域の未使用の空きメモリ容量を返す実数値関数です。	108 ページ
GETC	デバイス、または指定の論理ユニットの入力レコードから入力した次の文字（バイト）を返す実数値関数です。	109 ページ
GET.EVENT	指定のタスクに対して設定されるイベントを返す実数値関数です。	111 ページ
HERE	ロボットのツールポイントの現在のロケーションを示す変換値を返す変換関数です。	112 ページ
HOUR.METER	ロボットアワーメータの現在の値を返す実数値関数です。	113 ページ
$ID	システム ID 文字列を返す文字列関数です。	114 ページ
ID	現在のシステムの設定を指定する値を返す実数値関数です。	115 ページ
IDENTICAL	2 つのロケーションの値が同一かどうかを判定する実数値関数です。	125 ページ
INRANGE	ロボットが指定されたロケーションに到達できるかどうかを示す値を返し、ロケーションに到達できない場合は追加の情報を提供する、実数値関数です。	126 ページ
$INTB	16 ビットの整数の 2 進数表現を格納した 2 バイトの文字列を返す文字列関数です。	128 ページ
INT	値の整数部を返す実数値関数です。	129 ページ

キーワード	詳細	参照
INTB	2 バイトの文字列を符号付き16 ビット2 進整数として解釈した値を返す実数値関数です。	130 ページ
INVERSE	所定の変換値の逆関数となる変換値を返す変換関数です。	132 ページ
IOSTAT	論理ユニットに関連付けられたデバイスの最新の入出力命令に関するステータス情報を返す実数値関数です。	133 ページ
LAST	配列（次元）で使用されているインデックスの最大値を返す実数値関数です。	136 ページ
LATCH	最新の外部トリガーが発生したときのロボットのロケーションを示す変換値を返す変換関数です。	138 ページ
LATCHED	位置ラッチの状態、トリガーした入力の状態を返す実数値関数です。	139 ページ
LEN	指定された文字列の文字数を返す実数値関数です。	141 ページ
$LNGB	32 ビットの整数の2 進数表現を格納した4 バイトの文字列を返す文字列関数です。	142 ページ
LNGB	4 バイトの文字列を符号付き32 ビット2 進整数として解釈した値を返す実数値関数です。	143 ページ
MAX	値のリストにある最大値を返す実数値関数です。	145 ページ
$MID	指定文字列のサブストリングを返す文字列関数です。	146 ページ
MIN	値のリストにある最小値を返す実数値関数です。	147 ページ
NETWORK	ネットワークの状態とロボットコントローラの IP アドレス情報を返す実数値関数です。	148 ページ
NORMAL	変換値の切り捨て誤差を補正する変換関数です。	150 ページ
NOT	値の論理否定を実行する演算子です。	151 ページ
NULL	null 変換値（すべての成分が 0 の値）を返す変換関数です。	152 ページ
OFF	eV+ で使用される論理偽の結果を示す値を返す実数値関数です。	153 ペー

キーワード	詳細	参照
		ジ
ON	eV+ で使用される論理真の結果を示す値を返す実数値関数です。	154 ページ
OUTSIDE	指定範囲以外かを判断するために、値をテストする実数値関数です。	155 ページ
PARAMETER	指定したシステムパラメータの現在の設定を返す実数値関数です。	156 ページ
#PDEST	現在のロボットモーションに対して、計画された目的地を示すプレシジョンポイント値を返す精度ポイント関数です。	157 ページ
#PHERE	選択されているロボットの現在のロケーションを示すプレシジョンポイント値を返す精度ポイント関数です。	158 ページ
PI	円周率（3.141593）の値を返す実数値関数です。	159 ページ
#PLATCH	システムに含まれているエンコーダの、最新のラッチが発生した時点でのエンコーダ値を返すプレシジョンポイント関数です。	160 ページ
POS	文字列内のサブストリングの開始文字位置を返す実数値関数です。	161 ページ
#PPOINT	指定の構成要素から成るプレシジョンポイント値を返すプレシジョンポイント関数です。	162 ページ
PRIORITY	プログラムに対する現在の反応ロックアウトの優先度を返す実数値関数です。	164 ページ
RANDOM	疑似乱数を返す実数値関数です。	165 ページ
RX	X 軸に対する回転を示す変換を返す変換関数です。	166 ページ
RY	Y 軸に対する回転を示す変換を返す変換関数です。	167 ページ
RZ	Z 軸に対する回転を示す変換を返す変換関数です。	168 ページ
SCALE	変換パラメータの位置を倍率に従って拡大縮小した変換値を返す変換関数です。	169 ページ
SELECT	現在選択中のタスクに指定されたデバイスに関する情報を返す実数値関数です。.	170 ページ

キーワード	詳細	参照
#SET.POINT	最後の軌道評価サイクルで軌道ジェネレーターによって算出されたジョイント角の指令位置を返すプレシジョンポイント関数です。	172 ページ
SHIFT	指定のシフト量で変換パラメータの位置をシフトした変換値を返す変換関数です。	173 ページ
SIG.INS	デジタル I/O 信号がシステムに装備されているか、またはソフトウェア信号がシステムで使用できるかを示す値を返す実数値関数です。	174 ページ
SIGN	パラメータ「value」の符号を付けて、値「1」を返す実数値関数です。	176 ページ
SIG	指定されたデジタル信号の状態の論理積（AND）を返す実数値関数です。	177 ページ
SIN	指定した角度の三角関数のサイン値を返す実数値関数です。	179 ページ
SOLVE.FLAGS	ジョイント位置の配列で指定されたロボット設定を示すビットフラグを返す実数値関数です。	180 ページ
SPEED	システムのモーションの速度係数の一つを返す実数値関数。	182 ページ
SQRT	パラメータで指定した値の平方根を返す実数値関数です。	184 ページ
SQR	パラメータで指定した値の二乗値を返す実数値関数です。	185 ページ
STATE	ロボットシステムの状態に関する情報を提供する値を返す実数値関数です。	186 ページ
STATUS	アプリケーションプログラムのステータス情報を返す実数値関数です。	194 ページ
STRDIF	ソート用に 2 つの文字列をバイト単位で比較する実数値関数です。	196 ページ
SWITCH	システムスイッチの設定に関する情報を返す実数値関数です。	197 ページ
TAN	指定した角度の三角関数のタンジェント値を返す実数値関数です。	198 ページ
TAS	実数値変数の現在の値を返し、新しい値を割り当てる実数値関数です。他のプログラムタスクが同時に変数を修正できないようにするために、2つのアクションは不可分的に実行されます。	199 ページ

キーワード	詳細	参照
TASK	プログラム実行タスクに関する情報を返す実数値関数です。	202 ページ
$TIME	現在のシステムの日付と時刻、または指定した日付と時刻のいずれかを含む文字列値を返す文字列関数です。	205 ページ
$TIME4	現在のシステムの 4 桁の日付と時刻、または指定した 4 桁の日付と時刻のいずれかを含む文字列値を返す文字列関数です。	207 ページ
TIME	所定の文字列パラメータで指定された日付または時刻のいずれかを示す整数値を返す実数値関数です。	209 ページ
TIMER	指定されたシステムタイマーの現在のタイマー値を返す実数値関数です。	211 ページ
TOOL	最新の TOOL 命令で指定された変換値を返す変換関数です。	214 ページ
TPS	1秒あたりのシステムクロックのチック数を返す実数値関数です。	215 ページ
TRANS	指定の X、Y、Z の位置変位と y、p、r の回転方向から計算された変換値を返す変換関数です。	216 ページ
$TRANSB	2 進数に表現された変換値を含む 48 バイトの文字列を返す文字列関数です。	218 ページ
TRANSB	48 バイトまたは 96 バイトの文字列で示される変換値を返す変換関数です。	219 ページ
TRUE	eV+ で使用される、論理的な真を表すための値を返す実数値関数です。	220 ページ
$TRUNCATE	ASCII の NULL 文字が見つかるまで入力文字列のすべての文字を返す文字列関数です。	221 ページ
$UNPACK	128 文字の文字列変数配列からサブストリングを返す文字列関数です。	222 ページ
VAL	入力文字列の文字で示される実数値を返す実数値関数です。	224 ページ
VLOCATION	指定されたビジョンシーケンスの実行によるデカルト変換結果を返す変換関数です。x、y、z、yaw、pitch、roll で示される変換結果が戻り値となります。	225 ページ
VPARAMETER	ビジョンツールパラメータの現在の値を返す変換関数です。	228 ページ

キーワード	詳細	参照
VRESULT	ビジョンツールの指定された結果、または指定されたツールの状態を返す実数値関数です。	230 ページ
VSTATE	シーケンスの実行状態を返す実数値関数です。	232 ページ
WINDOW	ベルト相対変換値で記述された位置の、作動しているコンベアベルトの作業範囲の事前定義された境界に対する相対的な位置を示す値を返す、実数値関数です。	233 ページ

2.2 モニタコマンドキーワード概要

すべてのモニタコマンドキーワードの概要を次の表に示します。

キーワード	詳細	参照
ABORT	実行可能プログラムの実行を終了させるモニタコマンドです。	236 ページ
BASE	ロボットに対してワールド基準フレームを平行移動、回転するモニタコマンドです。	238 ページ
CALIBRATE	ロボットのポジショニングシステムを初期化するモニタコマンドです。	242 ページ
CD	ディスクアクセス用のデフォルトパスを表示、または変更するモニタコマンドです。	245 ページ
COMMANDS	モニタコマンドプログラム の処理を開始するモニタコマンドです。	246 ページ
COPY	既存のプログラムのコピーとして新しいプログラムを作成するモニタコマンドです。	248 ページ
CYCLE.END	次回、STOP プログラム操作、または同等の操作が実行されたとき、指定された実行可能なプログラムを終了します。プログラムの実行が完了するまで、コマンドプログラムの処理を一時停止します。	249 ページ
DEFAULT	アクセスされる eV+ ディスク論理デバイスと物理デバイス間のデフォルトの関係を定義するモニタコマンドです。また、現在のデフォルト値を表示します。	251 ページ
DELETE	システムメモリから指定されたプログラムを削除するモニタコマンドです。	255 ページ
DELETEL	システムメモリから名前付きロケーション変数を削除するモニタコマンドです。	257 ページ

キーワード	詳細	参照
DELETEM	システムメモリから指定したプログラムモジュールを削除するモニタコマンドです。	259 ページ
DELETEP	システムメモリから指定したプログラムを削除するモニタコマンドです。	261 ページ
DELETER	システムメモリから名前付き実数値変数を削除するモニタコマンドです。	263 ページ
DELETES	システムメモリから指定した文字列変数を削除するモニタコマンドです。	265 ページ
DIRECTORY	システムメモリ内の一部またはすべてのプログラムの名前を表示するモニタコマンドです。	267 ページ
DISABLE	1 つ以上のシステムスイッチを OFF するモニタコマンドです。	269 ページ
DO	実行可能なプログラム、または指定されたタスクやプログラムコンテキストの次のステップであるかのようにキーワードを実行するモニタコマンドです。	271 ページ
ENABLE	1 つ以上のシステムスイッチを ON するモニタコマンドです。	273 ページ
ESTOP	非常停止信号を受信した場合と同じ方法でロボットを停止するモニタコマンドです。	275 ページ
EXECUTE	制御プログラムの実行を開始するモニタコマンドです。	276 ページ
FCOPY	既存のディスクファイルの情報を新しいディスクファイルにコピーするモニタコマンドです。	279 ページ
FDELETE	ファイル指定に一致する 1 つ以上のディスクファイルを削除するモニタコマンドです。	281 ページ
FDIRECTORY	ディスクのファイル情報とストレージのスペース残量を表示、ディスクのサブディレクトリを作成、削除するモニタコマンドです。	283 ページ
FLIST	指定したディスクファイルの内容をモニタウィンドウにリスト表示するモニタコマンドです。	287 ページ
FREE	現在使用されていない使用可能なシステムメモリの割合（%）を表示するモニタコマンドです。	288 ページ
FRENAME	ディスクファイル名を変更をするモニタコマンドです。	289 ページ
FSET	ネットワークデバイスの属性を設定または修正するモニタコマンドです。	290 ページ
HERE	変換、またはプレシジョンポイント変数の値を現在のロボットの	291 ページ

キーワード	詳細	参照
	ロケーションと等しい値に定義するモニタコマンドです。	
ID	システムのコンポーネントに関する識別情報を表示するモニタコマンドです。	293 ページ
IO	外部デジタル入出力信号、または内部ソフトウェア信号の現在の状態を表示するモニタコマンドです。	296 ページ
JOG	ロボットの指定されたジョイントをジョグしたり、指定されたデカルト座標系の座標軸方向に沿ってロボットツールを移動します。JOG が実行されるたびに、ロボットは最大 300 ms 動作します。	298 ページ
KILL	プログラム実行スタックをクリアし、接続されている I/O デバイスを切り離すモニタコマンドです。	302 ページ
LIST	式の値を表示するモニタコマンドです。	303 ページ
LISTL	リストされたロケーションの値を表示するモニタコマンドです。	305 ページ
LISTP	リストされたユーザープログラムのすべてのステップを表示するモニタコマンドです。	307 ページ
LISTR	指定の実数式の値を表示するモニタコマンドです。	308 ページ
LISTS	指定の文字列の値を表示するモニタコマンドです。	310 ページ
LOAD	指定のディスクファイルの内容をシステムメモリにロードするモニタコマンドです。	312 ページ
MDIRECTORY	システムメモリにあるすべてのプログラムモジュール、または指定のプログラムモジュールにあるプログラムの名前を表示するモニタコマンドです。	315 ページ
MODULE	新しいプログラムモジュールを作成する、または既存のモジュールの内容を修正するモニタコマンドです。	317 ページ
NET	ネットワーク状態の情報を表示するモニタコマンドです。また、eV+ システムで現在定義されているリモートマウントの詳細情報も表示します。	319 ページ
PANIC	外部 E-stop ボタン押下をシミュレートし、すべてのロボットをただちに停止するが、ロボットのハイパワーを OFF にしないモニタコマンドです。	322 ページ
PARAMETER	システムパラメータの値を設定、または表示するモニタコマンドです。	323 ページ
PING	ノードに対してネットワーク接続をテストするモニタコマンドで	325 ページ

キーワード	詳細	参照
	す。	
PRIME	実行するプログラムの準備をしますが、実行は開始しないモニタコマンドです。	326 ページ
PROCEED	アプリケーションプログラムの実行を再開するモニタコマンドです。	328 ページ
RENAME	メモリにあるユーザープログラムの名前を指定された新しい名前に変更するモニタコマンドです。	330 ページ
RESET	すべてのデジタル出力信号をOFF するモニタコマンドです。	331 ページ
RESET.LOCK	アプリケーションプログラムからロボットをデタッチするモニタコマンドです。	332 ページ
RETRY	最後の割り込みステートメントの実行とプログラムの連続実行を繰り返すモニタコマンドです。	333 ページ
SELECT	以降のモニタウィンドウでの操作のために、ロボットを選択するモニタコマンドです。	335 ページ
SIGNAL	デジタル出力信号、内部ソフトウェア信号またはホスト信号をオンまたはオフにするモニタコマンドです。	337 ページ
SPEED	モニタ速度を指定するモニタコマンドです。	339 ページ
SRV.RESET	eV+ を再起動またはリセットするモニタコマンドです。	341 ページ
SSTEP	シングルステップ、または制御プログラムのサブルーチン全体を実行するモニタコマンドです。	342 ページ
STACK	サブルーチンコールと自動変数に使用するためにプログラムタスク用に予約するシステムメモリの量を指定するモニタコマンドです。	344 ページ
STATUS	システムと実行するプログラムのステータス情報を返すモニタコマンドです。	346 ページ
STORE	ディスクファイルのプログラムと変数を保存するモニタコマンドです。	350 ページ
STOREL	ディスクファイルのロケーション変数を保存するモニタコマンドです。	352 ページ
STOREM	指定されたプログラムモジュールをディスクファイルに保存するモニタコマンドです。	354 ページ
STOREP	プログラムファイルをディスクファイルに保存するモニタコマンドです。	356 ページ

キーワード	詳細	参照
STORER	実数値変数をディスクファイルに保存するモニタコマンドです。	358 ページ
STORES	文字列変数をディスクファイルに保存するモニタコマンドです。	360 ページ
SWITCH	モニタウィンドウにシステムスイッチの設定値を表示するモニタコマンドです。	362 ページ
TESTP	システムメモリに名前付きプログラムが存在するかをテストするモニタコマンドです。	364 ページ
TOOL	ロボットのツール取付フランジに対するツール先端のロケーションと方向を示すために使用される内部変換を設定するモニタコマンドです。	365 ページ
WAIT.START	条件が満たされるまで モニタコマンドプログラム を待機状態にするモニタコマンドです。	367 ページ
WHERE	ロボットの現在のロケーションとハンドオープニングを表示するモニタコマンドです。	368 ページ
XSTEP	プログラムをシングルステップで実行するモニタコマンドです。	369 ページ
ZERO	eV+ システムを初期化して、システムメモリにあるすべてのプログラムとデータを削除するモニタコマンドです。	372 ページ

2.3 他のキーワード概要

関数、モニタコマンド、プログラム命令、システムパラメータ、およびシステムスイッチとして分類されないすべてのキーワードの概要を次の表に示します。

キーワード	詳細	参照
.END	eV+ プログラムの終了を示すためのキーワードです。	374 ページ
IPS	SPEED プログラムコマンドの単位を inch/s として指定します。	375 ページ
MMPS	SPEED プログラムコマンドの単位を mm/s として指定します。	376 ページ

2.4 プログラム命令キーワード概要

すべてのプログラム命令キーワードの概要を次の表に示します。

キーワード	詳細	参照
ABORT	実行可能なプログラムタスクの実行を終了します。	377 ページ
ABOVE	次の動作に関するロボットの設定変更を要求して、肘が肩から手首のラインより上になるようにします。	379 ページ

キーワード	詳細	参照
ACCEL	ロボット動作の加減速度を設定し、オプションとして定義された加速プロファイルを指定します。	381 ページ
ALIGN	ロボットツール Z 軸を最も近いワールド軸に合わせます。	384 ページ
ALTER	次の軌道計算の間にロボットのパスに適用されるリアルタイムパス修正の大きさを指定します。	385 ページ
ALTOFF	リアルタイム経路修正モード（変更モード）を終了します。	386 ページ
ALTON	リアルタイムパス修正モード（変更モード）を有効にし、変更座標情報を解釈する方法を指定します。	387 ページ
ANY	CASE 構造の代替命令グループの先頭を知らせます。	389 ページ
APPRO	関節補間動作で指定した位置を基準とした位置へ向かうロボット動作を開始します。	390 ページ
APPROS	直線動作で指定した位置を基準とした位置へ向かうロボット動作を開始します。	391 ページ
ATTACH	デバイスをアプリケーションプログラムで使用できるようにします。	392 ページ
AUTO	プログラムを入力するときにプログラムスタックに自動的に作成されるテンポラリ変数を宣言します。	396 ページ
BASE	ロボットを基準にしてワールド基準フレームを平行移動、回転させます。	399 ページ
BELOW	次の動作に関するロボットの設定変更を要求して、肘が肩から手首のラインより下になるようにします。	401 ページ
BITS	値に基づくデジタル信号のグループを設定またはクリアします。	403 ページ
BRAKE	現在のロボットモーションを中止(アボート）します。	405 ページ
BREAK	現在の動作が完了するまで実行プログラムを中断します。	406 ページ
BY	SCALE および SHIFT 関数の構文を完了させます。	407 ページ
CALIBRATE	ロボットの位置決めシステムをロボットの現在位置を使用して初期化します。	408 ページ
CALL	現在実行しているプログラムを中断し、新しいサブルーチンプログラムを使用して実行を継続します。	412 ページ
CALLS	現在実行しているプログラムを中断し、文字列値で指定される新しいサブルーチンプログラムを使用して実行を継続します。	415 ページ
CASE	該当する値を定義することで CASE 構造の処理を開始します。	417 ページ

キーワード	詳細	参照
CLEAR.EVENT	指定されたタスクに関連するイベントをクリアします。	420 ページ
CLEAR.LATCHES	選択したデバイスのラッチバッファを空にします。	421 ページ
COARSE	ロボットに対して低精度ヌル許容誤差を可能にします。	422 ページ
CPOFF	次の動作（またはその後のすべての動作）の完了時およびヌル位置エラー時にロボットを停止するよう、eV+ システムに指示します。	424 ページ
CPON	eV+ システムに対して、連続経路の一部として次の動作（またはその後の動作すべて）を実行するように命令します。	426 ページ
CYCLE.END	指定されたタスク内で実行中のプログラムが STOP プログラムコマンド（または同等のもの）を次に実行するときに、そのプログラムを終了します。 指定されたタスクにおいて実行されているプログラムが実行を完了するまで実行可能なプログラムの処理を中断します。	428 ページ
DECOMPOSE	ロケーション値の個々の成分の実数値を抽出します。	430 ページ
DEF.DIO	各種キーワードで使用するためにバーチャルデジタル I/O に標準 eV+ 信号番号を割り当てます。	432 ページ
DEFBELT	コンベアトラッキングロボットを使用するためのベルト変数を定義します。	434 ページ
DEPART	現在のロケーションから離れるロボットモーションをジョイント補間モーションを用いて開始します。	436 ページ
DEPARTS	現在のロケーションから離れるロボットモーションを直線モーションを用いて開始します。	438 ページ
DETACH	指定したデバイスをアプリケーションプログラムの制御から解放します。	440 ページ
DISABLE	1 つ以上のシステムスイッチを OFF にします。	442 ページ
DO	DO プログラム構造体を作成します。	444 ページ
DOS	文字列式で定義されるキーワードを実行します。	446 ページ
DRIVE	文字列式で定義されるキーワードを実行します。	446 ページ
DURATION	以降のロボットモーションの最大実行時間を設定します。	450 ページ
ELSE	IF ... THEN 制御構造のステートメントの代替グループを区切ります。	452 ページ
ENABLE	1 つ以上のシステムスイッチを ON にします。	453 ページ

キーワード	詳細	参照
END	制御構造体の末尾を示します。	455 ページ
ESTOP	非常停止信号を受信したときと同様にロボットを停止します。	457 ページ
EXECUTE	制御プログラムの実行を開始します。	458 ページ
EXIT	制御構造体の n 番目にネストされたループに続くステートメントへ分岐します。	462 ページ
EXTERNAL	eV+ と NJ シリーズロボット統合CPU ユニット の間で共有される変数を宣言します。	463 ページ
FCLOSE	指定した論理ユニットにおいて現在開いているファイルを閉じます。	465 ページ
FCMND	論理ユニットが指定した入出力デバイスに対してそのデバイス特有のコマンドを作成します。	466 ページ
FCOPY	既存のディスクファイルの情報を新しいディスクファイルにコピーします。	470 ページ
FDELETE	指定したディスクファイルを削除します。	472 ページ
FEMPTY	ディスクに対して使用する内部バッファを、必要に応じてそのバッファをファイルに書き込むことで、空にします。	474 ページ
FINE	ロボットに対して高精度ヌル許容誤差を有効にします。	476 ページ
FLIP	次の動作に関するロボットの設定変更を要求して、ロボットの手首のピッチ角がマイナスの値になるようにします。	478 ページ
FOPEN	新しい TCP 接続を作成して開きます。	481 ページ
FOPENA	読み書き追加アクセスのためにファイルを開きます。指定したファイルがすでに存在しない場合、ファイルが作成されます。	484 ページ
FOPEND	読み取りのためにディスクディレクトリを開きます。	487 ページ
FOPENR	読み取りアクセスのためにファイルを開きます。	490 ページ
FOPENW	読み書きアクセスのためにファイルを開きます。ファイルが既に存在していると、エラーが発生します。	493 ページ
FOR	プログラムループを指定した回数実行します。	496 ページ
FSEEK	ファイルをランダムアクセスのために開く位置にして、指定レコードで読み取り操作を開始します。	498 ページ
FSET	ネットワークデバイスの属性を設定または変更します。	500 ページ
GLOBAL	変数をグローバルとして宣言し、その変数のタイプを指定します。	502 ページ

キーワード	詳細	参照
GOTO	指定のラベルで特定されるプログラムステップに無条件分岐を実行します。	504 ページ
HALT	実行プログラムを停止して再開できないようにします。	505 ページ
HERE	現在のロボットのロケーションに相当する変換またはプレシジョンポイント変数の値を設定します。	506 ページ
IF ... GOTO	論理式の値が TRUE （非 0）の場合に分岐して指定されたステップに飛びます。	508 ページ
IF ...THEN	論理式の結果に応じてキーワードグループ（または 2 つのグループのうちの一方）を条件に合わせて実行します。	509 ページ
IGNORE	REACT または REACTI プログラムコマンドの効果を無効にします。	511 ページ
JMOVE	ジョイントの値のリストに記載された位置にすべてのロボットのジョイントを動かします。ロボットはジョイント補間モードで協調的な動作を実行します。	512 ページ
JOG	ロボットの指定されたジョイントをジョグしたり、指定されたデカルト座標系の座標軸方向に沿ってロボットツールを移動します。	513 ページ
KEYMODE	ペンダントのキーのグループの動作を設定します。	517 ページ
KILL	プログラム実行スタックをクリアし、割り付けられている I/O デバイスを切り離します。	519 ページ
LEFTY	次の動作に関するロボットの設定変更を要求して、スカラロボットの最初の 2 つのリンクが左アーム方向を使用するようにします。	520 ページ
LOCAL	現在のプログラム内でのみ定義される恒久的変数を宣言します。	522 ページ
LOCK	プログラム応答ロックアウト優先度を与えられた値に設定します。	524 ページ
MC	モニタコマンドプログラム 内にモニタコマンドを作成します。	526 ページ
MCS	アプリケーションプログラムからモニタコマンドを呼び出します。	528 ページ
MOVE	ジョイント補間モーションを用いて所定のロケーションによって記述されるポジションと方向に至るロボットモーションを開始します。	530 ページ
MOVEC	与えられた地点によって記述される位置および方向を使用して円	531 ページ

キーワード	詳細	参照
	形／円弧経路のロボット動作を開始します。	
MOVES	直線モーションを用いて所定のロケーションによって記述されるポジションと方向に至るロボットモーションを開始します。	538 ページ
MULTIPLE	ロボット手首関節を完全に回転させることを可能にします。	539 ページ
NEXT	分岐して n 番目にネストされたループのEND文に進み、ループテストを実行し、適切であればループします。	540 ページ
NOFLIP	次の動作に関するロボットの設定変更を要求して、ロボットの手首のピッチ角がプラスの値になるようにします。	541 ページ
NONULL	eV+ システムに対して、連続経路動作の末尾で位置誤差がヌル処理されるのを待たないように命令します。	542 ページ
NOOVERLAP	eV+ システムに対して、連続経路動作の末尾で位置誤差がヌル処理されるのを待たないように命令します。	542 ページ
NULL	eV+ システムに対して、連続経路動作の末尾で位置誤差がヌル処理されるのを待機するように命令します。	546 ページ
OVERLAP	次のモーションまたは後続するすべてのモーションに対して、NOOVERLAP 制限エラーチェックを無効にします。	548 ページ
PACK	（128 文字の）文字列変数の配列の中または（配列ではない）文字列変数の中の部分文字列を置換します。	550 ページ
PANIC	外部 E-STOP ボタンを押した場合をシミュレーションし、すべてのロボットを即座に停止しますが、HIGH POWER をオフにしません。	552 ページ
PARAMETER	システムパラメータの値を設定します。	553 ページ
PAUSE	実行プログラムを停止しますがプログラムの再開は可能にします。	555 ページ
PDNT.CLEAR	該当する場合に、T20 ペンダントの現在の告知ウィンドウまたはカスタムメッセージウィンドウをクリアして T20 ペンダントをホーム画面に戻します。	556 ページ
PDNT.NOTIFY	ペンダント告知を作成します。	557 ページ
PDNT.WRITE	ペンダントのカスタムメッセージ画面を設定します。	558 ページ
PENDANT	手動制御ペンダントからの入力を返します。	560 ページ
PROCEED	アプリケーションプログラムの実行を再開します。	563 ページ
.PROGRAM	実行する際にプログラムに渡される引数を定義します。	565 ページ

キーワード	詳細	参照
PROMPT	モニタウィンドウに文字列を表示して、オペレータの入力を待ちます。	568 ページ
REACT	指定したデジタル信号の連続モニタリングを開始し、信号が変化するとサブルーチンを自動的に引き起こします。	570 ページ
REACTE	現在のプログラムタスク実行中に発生するシステムメッセージのモニタリングを開始します。	573 ページ
REACTI	指定したデジタル信号の継続的モニタリングを開始します。信号が正しく遷移すると現在のロボット動作を自動的に停止し、オプションとしてサブルーチンの呼び出しをトリガします。	575 ページ
READ	開かれたファイルまたはファイル指向ではない割り付けられたデバイスからレコードを読み取ります。ネットワークデバイスの場合、割り付けられた開いている TCP 接続から文字列を読み取ります。	577 ページ
READY	ロボットをレディ位置に移動します。	581 ページ
RELEASE	利用可能な次のプログラムタスクの実行を可能にします。	583 ページ
RESET	外部出力信号をすべて OFF にします。	584 ページ
RETRY	最後に中断されたプログラムコマンドを再度実行し、実行プログラムを継続します。	585 ページ
RETURN	現在のサブルーチンの実行を終了して、中断された実行プログラムを次のステップで再開します。	587 ページ
RETURNE	エラー応答サブルーチンの実行を終了し、サブルーチンを呼び出したステートメントの次のステップにおいて最後に中断した実行プログラムを再開します。	588 ページ
RIGHTY	次の動作に関するロボットの設定変更を要求して、スカラロボットの最初の 2 つのリンクが右アーム方向を使用するようにします。	589 ページ
RUNSIG	呼び出しプログラムタスクの実行が継続している間は指定したデジタル信号を ON または OFF にします。	591 ページ
SELECT	現在のタスクがアクセスするために、命名されたデバイスのユニットを選択します。	593 ページ
SET.EVENT	指定されたタスクに関連するイベントを設定します。	595 ページ
SET	等号の左側にあるロケーション変数の値を、右側のロケーション値に等しくなるように設定します。	596 ページ

キーワード	詳細	参照
SETBELT	指定したベルト変数のエンコーダオフセットを式の値と等しくなるように設定します。	598 ページ
SETDEVICE	デバイスを初期化するか、デバイスパラメータを設定します。実行されるオペレーションは参照されるデバイスによって異なります。	600 ページ
SIGNAL	外部デジタル出力信号または内部ソフトウェア信号を ON または OFF にします。	602 ページ
SINGLE	ロボットの手首のジョイントの回転を -180°～+180° の範囲に限定します。	604 ページ
SOLVE.ANGLES	現在のロボットに対して、指定した変換に相当するロボットジョイント位置を計算します。	605 ページ
SOLVE.TRANS	現在のロボットに対して与えられたジョイントポジションの組に相当する変換を計算します。	612 ページ
SPEED	後続するロボット動作の定格速度を設定します。	614 ページ
STOP	現在のプログラムタスクの実行を終了します。	617 ページ
SWITCH	値に基づいてシステムスイッチを有効または無効にします。	618 ページ
TIMER	指定したシステムタイマーを所定の時間値に設定します。	620 ページ
TOOL	ロボットのツール取り付けフランジを基準としたツール先端のロケーションと方向を表現するために使用する内部変換を設定します。	621 ページ
TYPE	出力指定によって記述される情報をモニタウィンドウに表示します。	622 ページ
UNTIL	DO ... UNTIL 制御構造体の末尾を示すと共に、そのループを脱出する時機を判定するために評価される式を指定します。この式の値が 0 以外になるまで、ループの実行を続けます。	625 ページ
VALUE	直後のプログラムステートメントを実行するために、CASE ステートメントの式が一致する必要がある値を示します。	626 ページ
VPARAMETER	画像センサツールパラメータの現在の値を設定します。	627 ページ
VRUN	画像センサシーケンスの実行を開始します。	629 ページ
VWAITI	指定した画像センサシーケンスが type パラメータで指定する状態になるまで待機します。	630 ページ
WAIT	1軌道サイクル分、プログラムを待機ループに入れます。条件を指定した場合、その条件が TRUE になるまで待機します。	631 ページ

キーワード	詳細	参照
WAIT.EVENT	指定したイベントが発生するまで、または指定した時間が経過するまで、プログラムの実行を中断します。	632 ページ
WHILE	条件がTRUE の場合WHILE 構造体の処理を開始し、その条件が最初からFALSE の場合は、WHILE 構造体をスキップします。	635 ページ
WINDOW	コンベアのトラッキングのため、指定したベルト変数の動作範囲の境界を設定します。	637 ページ
WRITE	開いているファイルまたは任意の I/O デバイスにレコードを書き込みます。ネットワークデバイスの場合、アタッチされたデバイスに文字列を書き込み、TCP 接続を開きます。	639 ページ

2.5 システムパラメータキーワード参照

すべてのシステムパラメータキーワードの概要を次の表に示します。

キーワード	詳細	参照
BELT.MODE	eV+ システムのコンベアトラッキング機能の特性を設定するシステムパラメータです。	641 ページ
DEVIATION	ロボットが特異点にあるとき、特異点領域のモーションに 1～100% のパス偏差を追加します。	643 ページ
JOG.TIME	ジョグオペレーションのキープアライブタイムを設定するシステムパラメータです。	644 ページ
NOT.CALIBRATED	システムに接続されているロボットのキャリブレーション状態を示すシステムパラメータです。	645 ページ
VTIMEOUT	後続するビジョンコマンドの応答が無い場合に、エラーメッセージを返すまでのタイムアウト値を設定するシステムパラメータです。	647 ページ

2.6 システムスウィッチキーワード概要

すべてのシステムスイッチキーワードの概要を次の表に示します。

キーワード	詳細	参照
AUTO.POWER.OFF	このシステムスイッチは特定の動作エラーが生じた際にハイパワーを無効にします。	648 ページ
CP	ロボットの連続パス機能を制御するシステムスイッチです。	650 ページ
DECEL.100	ACCEL プログラムコマンドキーワードに対して 100% の最大減速を有効または無効にするシステムスイッチです。	651 ページ

キーワード	詳細	参照
DELAY.IN.TOL	eV+ システムが動作区間を完了した後、粗ヌリングまたは微ヌリングのタイミングを制御するシステムスイッチです。	652 ページ
DRY.RUN	eV+ がロボットと通信するかしないかを制御するシステムスイッチです。	654 ページ
MESSAGES	モニタウィンドウに対する TYPE コマンドプログラムからの出力を有効/無効にするシステムスイッチです。	656 ページ
OBSTACLE	選択されたロボットに対して最大で 4 つの障害を有効、または無効にするシステムスイッチです。	657 ページ
POWER	ロボットハイパワーの状態を制御または監視するシステムスイッチです。	659 ページ
ROBOT	1 台のロボットまたはすべてのロボットを有効／無効にするシステムスイッチです。	661 ページ
SCALE.ACCEL	プログラム速度の関数として加速度および減速度のスケーリングを有効または無効にするシステムスイッチです。	663 ページ
SCALE.ACCEL.ROT	直線動作中、SCALE.ACCEL スイッチが直交座標回転速度を考慮するかどうかを指定するシステムスイッチです。	665 ページ
UPPER	文字列を比較するとき、各文字の大文字と小文字の違いを制御するシステムスイッチです。	666 ページ

第3章: キーワードの詳細

本項では、すべての eV+ キーワードに説明します。

3.1 関数キーワード

本項の情報を参考にして、関数キーワードと eV+ システムでの使用方法について理解してください。

ABS

指定された引数の絶対値（大きさ）を返す実数値関数です。

構文

ABS (value)

パラメータ

value　実数値式です。

例

次の例では、0.123 を返します。

```
ABS(0.123)
```

次の例では、5.462 を返します。

```
ABS(-5.462)
```

次の例では、0.013125 を返します。

```
ABS(1.3125E-2)
```

次の例では、変数「part.size」を変数「belt.scale」の絶対値で除算し、変数「belt.length」に結果を返します。

```
belt.length = part.size/ABS(belt.scale)
```

ACCEL

ロボットの設定プロファイルで定義されている加速、減速、または最大許容制限率（%）の現在の設定値を返す実数値関数です。

構文

ACCEL (value)

使用上の留意事項

ACCEL 関数は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

パラメータ

value 実数値式、返される値を選択するために整数に四捨五入した結果です。選択パラメータに次の値を設定すると、特定の情報が返されます。

- 0：選択した加速度プロファイルの番号
- 1：加速度
- 2：減速度
- 3：許容可能な最大加速率
- 4：許容可能な最大減速率
- 5：SCALE.ACCEL システムスイッチが有効な場合、プログラムの速度設定に比例して、加速度と減速度の割合を増減するプログラム速度

例

次の例では、現在の加速度の設定値を返します。

```
ACCEL(1)
```

次の例では、現在の減速度の設定値を返します。

```
ACCEL(2)
```

関連キーワード

ACCEL(プログラムコマンド)

SCALE.ACCEL

SELECT(実数値関数)

SELECT (プログラムコマンド)

ACOS

引数のアークコサインを返す実数値関数です。

構文

ACOS (value)

使用上の留意事項

パラメータの値は、-1.0～+1.0 の範囲内にする必要があります。値がこの範囲以外の場合、不正な値によるエラーが発生します。

パラメータ

value	コサイン値を定義する実数値式です。

詳細

-1.0～+1.0 の範囲内にある引数の逆コサイン（アークコサイン）値を返します。結果の値は、常に 0.0～+180.0 の範囲内にあります。

例

次の例では、値 90 を返します。

```
ACOS(0)
```

次の例では、値 180 を返します。

```
ACOS(-1)
```

次の例では、値 84.2608295 を返します。

```
ACOS(0.1)
```

次の例では、値 60 を返します。

```
ACOS(0.5)
```

> **注:** TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

COS

SIN

ASIN

TAN

ATAN2

ALIGN

入力位置をワールド座標系の最も近い軸に合わせる変換関数です。

構文

ALIGN (location)

パラメータ

location	基準として使用される変換値です。

詳細

入力位置をワールド座標系の最も近い軸に対して平行に合わせて修正した値を返します。

例

次の例では、ロボットの位置を「point1」で定義された位置を基準にして合わせ、位置変数「align.loc」に代入します。

```
SET align.loc= ALIGN(point1)
```

関連キーワード

ALIGN （プログラムコマンド）

SELECT (プログラムコマンド)

SELECT（実数値関数）

ASC

文字列内の ASCII 文字の値を返す実数値関数です。

構文

ASC(string, index)

パラメータ

string 文字を選択する文字列式です。文字列が空の場合、関数は -1 の値を返します。

index 対象文字の位置を定義するオプションの実数値式です。index が省略、または値が 0 か 1 の場合、文字列の最初の文字が選択されます。

index の値がマイナス、または文字列の長さより大きい場合、関数は -1 の値を返します。

詳細

指定された文字の ASCII 値が実数値として返されます。

例

次の例では、文字「a」の ASCII 値を返します。

```
ASC("sample", 2)
```

次の例では、変数「$name」に含まれる配列の最初の文字に対応する ASCII 値を返します。

```
ASC($name)
```

次の例では、実際の変数「i」の値を、変数「$ system」に含まれる文字列の対象文字のインデックスとして使用しています。

```
ASC($system, i)
```

関連キーワード

$CHR

VAL

ASIN

引数のアークサインを返す実数値関数です。

構文

ASIN (value)

使用上の留意事項

値のパラメータは、-1.0～+1.0 の範囲内にする必要があります。値がこの範囲以外の場合、不正な値によるエラーが発生します。

パラメータ

value	サイン値を定義する実数値式です。

詳細

-1.0～+1.0 の範囲内にある引数の逆サイン（アークサイン）を返します。結果の値は、必ず -90.0～+90.0 の範囲内にあります。

例

次の例では、0 の値を返します。

```
ASIN(0)
```

次の例では、-90 の値を返します。

```
ASIN(-1)
```

次の例では、5.73917047 の値を返します。

```
ASIN(0.1)
```

次の例では、30 の値を返します。

```
ASIN(0.5)
```

> **注:** TYPE、 PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、精度限界までの実数値を表示します。

関連キーワード

COS

ACOS

SIN

TAN

ATAN2

ATAN2

value_1/value_2 の値に等しいタンジェント値を持つ角度を返す実数値関数です。

構文

ATAN2 (value_1, value_2)

使用上の留意事項

パラメータ値が共に 0 の場合、戻り値は 0 になります。

パラメータ

value_1　実数値式です。

value_2　実数値式です。

例

次の例では、26.1067 を返します。

```
ATAN2(0.123,0.251)
```

次の例では、-6.600926 を返します。

```
ATAN2(-5.462,47.2)
```

次の例では、-90.56748 を返します。

```
ATAN2(1.3125E+2,-1.3)
```

次の例では、「rise/run」に等しいタンジェント値を持つ角度を「slope」変数に返します。

```
slope = ATAN2(rise, run)
```

注: TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、精度限界までの実数値を表示します。

関連キーワード

COS

ACOS

SIN

ASIN

TAN

BASE

最新のBASE プログラムのコマンドまたはモニタコマンドで設定された移動・回転を表す変換値を返す変換関数です。

構文

BASE

使用上の留意事項

BASE 関数は、関数を実行するタスクが選択したロボットの情報を返します。

LISTL BASE ステートメントを使用して、現在のベース設定を表示できます。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

関連キーワード

BASE(モニタコマンド)

BASE (プログラムコマンド)

LISTL

SELECT(実数値関数)

SELECT (プログラムコマンド)

BCD

実数値を 2 進化 10 進数（BCD）形式に変換する実数値関数です。

構文

BCD (value)

使用上の留意事項

BCD 関数は、BITSキーワード と併用するときに最も役立ちます。

パラメータ

value	変換される値を定義する実数式です。

詳細

BCD 関数は、0～9999 の範囲の整数値を BCD に変換します。例えば、外部出力信号セットに BCD 値を設定できます。

例

次の例では、実数値変数「digit」を BCD 値に変換し、外部デジタル出力信号 4～8 に適用しています。

```
BITS 4,4 = BCD(digit)
```

関連キーワード

DCB

BITS (プログラムコマンド)

BITS（実数値関数）

BITS(モニタコマンド)

BELT

コンベア追跡機能で追跡されるコンベアベルトに関する情報を返す実数値関数です。

構文

BELT (%belt_var, mode)

使用上の留意事項

この関数を使用する前に、BELT システムスイッチを有効にする必要があります。

SETBELT プログラムコマンドは、通常、有効ベルト位置をゼロに設定するために、BELT 関数と併用して使用されます。この操作は、ロボットがベルトに関連するモーションシーケンスを実行するたびに実行し、このシーケンスの最初のモーションを実行する直前に、実行する必要があります。



警告: ベルト追跡が有効な間、BELT 関数が返す現在のベルト位置と 8,388,607 (^H7FFFFF) を上限として指定されたベルト変数のベルト位置との差を確実にするために、ロボットがベルトを追跡するたびに SETBELT プログラムコマンドを実行することが重要です。ベルトの追跡中に、差がこの値を超えた場合、ロボットのモーションは、予測不可能になる可能性があります。

パラメータ

%belt_var	コンベアベルトの参照に使用されるベルト変数名です。すべてのベルト変数名は、パーセント記号 **(%)** で始める必要があります。
mode	返される情報を決定するコントロール値です。 mode が省略、または値が 0 の場合、BELT 関数は、ベルト変数で指定されたベルトのエンコーダカウント値のエンコーダ読取値を返します。この関数の戻り値の上限値は、絶対値 8,388,607 です。この上限値を 1 つ超えると、-8,388,608 になります。 値が -1 の場合、BELT 関数は、ベルト変数で指定されたベルトのエンコーダカウント値の最後にラッチされたエンコーダ位置を返します。この値は、8,388,607 を超える場合を除き、DEVICE（0, enc, stt, 4）で返された値と同じです。 式の値が 0 より大きい場合、エンコーダの速度は、eV+ サイクル（16 ms）毎のエンコーダカウント値の単位で返されます。

例

次の例では、ベルトエンコーダの現在の読取値に該当する、参照されたコンベアに対象の位置を設定します。

```
SETBELT %main.belt = BELT(%main.belt)
```

次の例では、ベルト変数「%b」に関連する現在のベルト速度を保存します。

```
belt.speed = BELT(%b, 1)
```

関連キーワード

DEFBELT

DEVICE

SETBELT

BITS

複数のデジタル信号を読み取り、バイナリビットパターンに対応する値を返す実数値関数です。

構文

BITS (first_sig, num_sigs**)**

使用上の留意事項

外部のデジタル入出力信号、またはソフトウェア内部の信号を参照できます。

最大 32 の信号を一度に読み取ることができます。

グループのすべての信号がシステムで使用するように設定されている場合、どのグループでも最大 32 の信号を読み取ることができます。

パラメータ

first_sig	読み取る信号の最小番号を定義する実数値式です。
num_sigs	読み取る信号数を指定するオプションの実数式です。指定されていない場合、値は 1 とみなされます。最大有効値は 32 です。

詳細

この関数は、1～32 のデジタル信号に現れるバイナリビットパターンに対応した値を返します。

この関数の戻り値のバイナリ表現は、first_sig で定義した最下位ビットと、次の num_sigs -1 で定義した最上位ビットを持ちます。

例

次の例では、現在の入力信号が以下の状態にあるとします。

```
信号:  1008   1007   1006   1005   1004   1003   1002   1001
状態:   1      1      0      1      0      1      1      0
```

次のステートメントは、4 つの信号 1003、1004、1005、および 1006 をバイナリ表現として解釈するため、変数「x」に 5 を返します。

```
x = BITS(1003, 4)
```

関連キーワード

BITS（プログラムコマンド）

IO

RESET

SIG

SIG.INS

SIGNAL(モニタコマンド)

SIGNAL (プログラムコマンド)

BMASK

個々のビットを指定してビットマスクを作成する実数値関数です。

構文

BMASK (bit, bit**, ...,** bit**)**

パラメータ

bit 1～32 の整数値で、ON にするビットを指定します。最下位ビットは 1 です。

詳細

この関数は、指定されたビットを ON にし、それ以外のすべてのビットを OFF のままにしてビットをマスクします（ON = 1、OFF = 0）。

ビット 32 は符号ビットで、設定された場合、マイナスの値を出力します。

例

次の例では、ビットマスク^B10001 を作成し、mode のビット 2 を ON にしてディスク LUN に接続します。

```
bm = BMASK(1, 5)


mode = BMASK(2)
ATTACH (lun, mode) "DISK"
```

BSTATUS

コンベア追跡システムの状態に関する情報を返す実数値関数です。

構文

BSTATUS

使用上の留意事項

BSTATUS 関数は、関数を実行するタスクが選択したロボットの情報を返します。

詳細

この関数は、通常、BELT.MODE システムパラメータのビット 4 が設定されているときに使用されます。

この関数は、コンベア追跡ソフトウェアの次の状態を示すビットフラグセットで示されるバイナリ値に等しい値を返します。

ビット 1 (LSB)	ベルト追跡（マスク値 = 1） このビットが設定されている場合、ロボットは現在ベルトを追跡中です。
ビット 2	目標位置上流（マスク値 = 2） このビットが設定されている場合、最後の動作の計画中にベルトウィンドウの上流側に目標位置が見つかっています。
ビット 3	目標位置下流（マスク値 = 4） このビットが設定されている場合、最後の動作の計画中にベルトウィンドウの下流側に目標位置が見つかっています。
ビット 4	ウィンドウ違反（マスク値 = 8） このビットが設定されている場合、最後のベルト関連動作中、ロボットがベルトを追跡している間にウィンドウ違反が発生しています。このフラグは各ベルト関連動作の開始時にクリアされます。

関連キーワード

BELT（実数値関数）

PARAMETER（モニタコマンド）

PARAMETER（プログラムコマンド）

PARAMETER（実数値関数）

WINDOW（プログラムコマンド）

WINDOW（実数値関数）

CAS

実数値変数をテスト値と比較し、一致していれば新しい値を実数値変数に設定する処理を、不可分に実行する実数値関数です。

構文

CAS (variable, test_value, new_value)

使用上の留意事項

eV+ システムは、複数のプログラムタスクで共有されるグローバル変数に対して、保護スキームを実施しません。このようなグローバル変数の使用を追跡することは、プログラマーの責任です。CAS 実数値関数（または同様の TAS関数）は、共有変数へのアクセスに対して論理的インターロックを行うために使用できます。

また、この関数は、複数のプログラムタスクがグローバル配列に同時にアクセスするときの制限回避にも使用できます。2 つ以上のタスクが同時に配列サイズの増加を試みた場合、プログラムの実行は失敗する可能性があります。グローバル配列へのアクセス制限の詳細については*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

パラメータ

variable	テストと新しい値の割り当て用に使用される実数値変数名です。
test_value	比較値を定義する実数値、変数、式です。
new_value	指定された変数に割り当てる新しい値を定義する実数値、変数、式です。

詳細

変数がテスト値と等しい場合、新しい値は変数に保存されます。それ以外の場合、変数は変更されません。変数の元の値は、関数値として返されます。

比較演算と新しい値の設定演算は、これらの演算を不可分に実行するため、割り込みをロックして実行されます。この関数は、TAS実数値関数と同様のタスク間のセマフォを設定する方法を提供します。詳細については、TAS を参照してください。

関数の実行時に変数が定義されていない場合、値は 0 として処理されます。

関連キーワード

TAS

$CHR

指定した ASCII 値に該当する 1 文字を返す文字関数です。

構文

$CHR(value)

パラメータ

value	文字に変換する値を定義する実数値式です。値は、0～255（10 進数）の範囲内にする必要があります。 値が 0～127（10 進数）の範囲の場合は、該当する ASCII 文字が返されます。

例

次の例では、文字「A」（ASCII 値 65）が返されます。

```
$CHR(65)
```

関連キーワード

ASC

$DBLB

$FLTB

$INTB

$LNGB

COM

値に対して 2 の補数演算を実行します。

構文

...COM 値 ...

使用上の留意事項

COM 関数は、整数値に対してのみ実行の意味があります。実数値の整数部のみ使用されます。小数部は無視されます。

パラメータ

value	補完される値を定義する実数値式です。

詳細

COM 関数は、ビットごとに 2 の補数演算を実行し、実数値を返します。

COM 演算は、次のステップから構成されます。

1. オペランドを 32 ビットの符号拡張整数に変換し、小数部を切り捨てます。
2. 2の補数演算を実行します。
3. 結果を実数値に戻します。

式内の演算子の評価順序については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、-41 の値を返します。

```
COM 40
```

関連キーワード

NOT

CONFIG

ロボットの幾何学的設定または動作サーボ制御機能の状態に関する情報を示す値を返す実数値関数です。

構文

CONFIG (select)

使用上の留意事項

CONFIG 関数は、関数を実行するタスクが選択したロボットの情報を返します。

eV+ システムがロボットを制御するように構成されていない場合、CONFIG コマンドを使用するとエラーになります。

パラメータ

select	オプションの実数値、変数、式であり、返される設定情報カテゴリーを選択する 0～13 の範囲の整数値として解釈されます。

詳細

この関数は、一連のビットフラグとして解釈される値を返します。この関数の戻り値は、select パラメータに応じて異なります。

select パラメータが省略、または値が 0、1、2 の場合、CONFIG 関数は、ロボットの幾何学的設定を示すビットフラグとして解釈できる値を返します。この値の各ビットは、ロボット設定の特性を示します。

select パラメータが 3、4、5 の場合、CONFIG 関数は、複数のロボット動作サーボ制御機能の設定値を示すビットフラグとして解釈できる値を返します。この値の各ビットは、動作サーボ制御機能の状態を示します。

select パラメータが 6、7、8 の場合、CONFIG 関数は、精密許容誤差の設定値を示す値を返します。FINE を ALWAYS パラメータ付きで使用する場合、この機能に制限が適用されます。詳細についてはFINE 476 ページを参照してください。

select パラメータが 9、10、11 の場合、CONFIG 関数は、粗許容誤差の設定値を示す値を返します。COARSE を ALWAYS パラメータ付きで使用する場合、この機能に制限が適用されます。詳細についてはCOARSE 422 ページを参照してください。

このグループのパラメータ値は、関数がどのロボット設定を返すかを決定します。

select = 0

この場合、ロボットの現在の設定を返します。デフォルト値は 0 です。

select = 1

ロボットが動作している場合はロボットの現在の動作における設定、またはロボットは割り付いているが動作を実行していない場合は現在の設定を返します。

select = 2

次の動作が関節補間であり、直線動作ではないと想定される場合、動作完了時のロボットの設定を返します。

Select = 1、2、3 の場合に返されるビットフラグの意味を次の表に示します。

ビット番号	ビットマスク	ビット ON の意味
1	1	右利き設定。
2	2	肘の位置（下）設定。
3	4	フリップあり設定。

select = 3

ALWAYS 識別子を指定するキーワードで定義されるロボット動作サーボ制御機能のパラメータ設定値を返します。

select = 4

現在実行中の動作、または動作が実行中ではない場合は完了した最後の動作の一時設定値を返します。

select = 5

実行される次の動作に適用される一時設定値を返します。

3、4、5 の場合に返されるビットフラグの意味を次の表に示します。

ビット番号	ビットマスク	ビットクリアの意味	ビットセット
1	1	なし	なし
2	2	FINE アサート	COARSE アサート
3	4	NULL アサート	NONULL アサート
4	8	MULTIPLE アサート	SINGLE アサート
5	^H10	CPON アサート	CPOFF アサート
6	^H20	OVERLAP アサート	NOOVERLAP アサート

select = 6

標準許容誤差の割合（%）として、永続的な設定値とする精密許容誤差を返します。

select = 7

標準許容誤差の割合（%）として、前回または現在の動作で使用される設定値とする精密許容誤差を返します。

select = 8

標準許容誤差の割合（%）として、次の動作で使用される設定値とする精密許容誤差を返します。

select = 9

標準許容誤差の割合（%）として、永続的な設定値とする粗許容誤差を返します。

select = 10

標準許容誤差の割合（%）として、前回または現在の動作で使用される設定値とする粗許容誤差を返します。

select = 11

標準許容誤差の割合（%）として、次の動作で使用される設定値とする粗許容誤差を返します。

select = 12

選択されたロボットで使用できるジョイント設定オプションを返します。返されるビットフラグの意味を次の表に示します。

ビット番号	ビットマスク	ビットセットの意味
1	1	ロボットを左利き、右利きに設定可能。
2	2	ロボットを上向き、下向きに設定可能。
3	4	ロボットをフリップあり、フリップなしに設定可能。
18	^H20000	ロボットは、OVERLAP、NOOVERLAP キーワードをサポートします。
22	^H200000	SINGLE、MULTIPLE プログラムコマンドで、ジョイントの最終回転角度を ±180 度に制限できます。

select = 13

ロボット動作の種類が返されます。戻り値のビットを次の表に示します。

ビット番号	ビットマスク	説明
1	1	このビットは、動作が関節補間の場合にセットされます。直線ライン動作の場合、クリアされます。

例

次の例では、ロボットが RIGHTY 設定と LEFTY 側設定のどちらで動いているかを確認します。モニタウィンドウに設定を示すテキストが表示されます。

```
IF (CONFIG(1)==0) OR (CONFIG(1)==4 THEN
    TYPE "Robot is moving to a lefty position."
ENDIF (CONFIG(1)==1) OR (CONFIG(1)==5 )THEN
    TYPE "Robot is moving to a lefty position."
END
```

関連キーワード

ABOVE

BELOW

COARSE

CPOFF

CPON

FINE

FLIP

LEFTY

NOFLIP

NONULL

NOOVERLAP

NULL

OVERLAP

RIGHTY

SELECT

SELECT（実数値関数）

SINGLE

STATE

COS

指定された角度のコサイン値を返す実数値関数です。

構文

COS(angle)

使用上の留意事項

角度パラメータの単位は度である必要があります。パラメータは、360 度の剰余として解釈されますが、過度に大きな値を指定した場合、戻り値の精度が低くなる可能性があります。

パラメータ

angle	角度値(度)を定義する実数値式です。

詳細

度で指定された引数のコサイン値を返します。結果の値は、必ず -1.0～+1.0 の範囲内となります。

例

次の例では、0.999962 の値を返します。

```
COS(0.5)
```

次の例では、0.9954596 の値を返します。

```
COS(-5.462)
```

次の例では、0.4999999 の値を返します。

```
COS(60)
```

次の例では、-0.659345 の値を返します。

```
COS(1.3125E+2)
```

> **注:** TYPE、 PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

ACOS

SIN

ASIN

TAN

ATAN2

CUBRT

値の立方根を返す実数値関数です。

構文

CUBRT(value)

パラメータ

value	立方根が計算される値を定義する実数値式です。

例

次の例では、0.497319 の値を返します。

```
CUBRT(0.123)
```

次の例では、2.0 の値を返します。

```
CUBRT(8)
```

次の例では、-1.7611 の値を返します。

```
CUBRT(-5.462)
```

次の例では、5.081982 の値を返します。

```
CUBRT(1.3125E+2)
```

> **注:** TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

SQR

SQRT

$DBLB

実数値をIEEE 754規格の倍精度浮動小数点形式の2進数で表現した8 バイトの文字列を返す文字列関数です。

構文

$DBLB(value, littleEndian**)**

パラメータ

value	IEEE 754規格の浮動小数点形式の 2 進数に変換する実数値式です。
littleEndian	エンディアンの種類を指定するオプションの値です。1 の場合、リトルエンディアンです。それ以外の場合はビッグエンディアンです。

詳細

この関数は、主にデータファイルの出力レコードの倍精度実数値を2進数に変換するために使用されます。

実数値は、IEEE 754規格の倍精度浮動小数点形式で2進数に変換されます。この 64 ビットの値は、連続した 8 文字（1 文字 8 ビット）として文字列に格納されます。IEEE 浮動小数点形式に関する詳細については、DBLB実数値関数を参照してください。

例

次の例では、以下のような文字列を返します。

```
$DBLB(1.215)
```

次のように文字列として返される。

```
$CHR(^H3F)+$CHR(^H3F)+$CHR(^H70)+$CHR(^HA3)+$CHR
(^HD7)+$CHR(^H0A)+$CHR(^H3D)+$CHR(^H71)
```

関連キーワード

$CHR

FLTB

$FLTB

$INTB

$TRANSB

DBLB

8 バイトの文字列をIEEE 倍精度浮動小数点数として解釈した値を返す実数値関数です。

構文

DBLB ($string, first_char**,** litteEndian**)**

パラメータ

$string	変換される 8 バイトを含む文字列式です。
first_char	8 バイトの文字列の開始位置を指定するオプションの実数値式です。 first_char が省略、または値が 0、1 の場合、文字列の先頭から 8 バイトが抽出されます。 first_charが 1 より大きい場合、抽出するバイトの開始文字位置として解釈されます。例えば、値 2 は、9 番目のバイト(2 つ目の 8 バイト)から 8 バイト分が抽出されます。 first_char に入力文字列の末尾を超えた 8 バイトを指定した場合、エラーになります。
littleEndian	エンディアンの種類を指定するオプションの値です。1 の場合、リトルエンディアンです。それ以外の場合はビッグエンディアンです。

詳細

この関数は、主に入力データレコードのバイナリ浮動小数点数を eV+ で内部的に使用できる値に変換するために使用されます。

浮動小数点数

指定された文字列の連続した 8 バイトは、IEEE 標準形式の倍精度（64ビット）浮動小数点数として解釈されます。この 64 ビットフィールドは、次のように解釈されます。

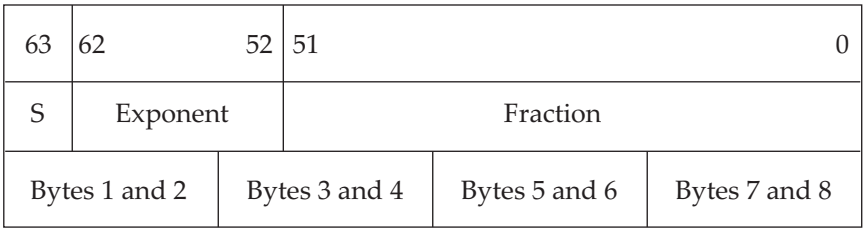


図3-1. 64 ビット浮動小数点形式

表3-1. 64 ビット浮動小数点形式の説明

項目	説明
符号	符号ビット（0 = 正、 1 = 負）
指数部	2 進指数、-1023 でバイアス
仮数部	小数点の左側を 1 とする 2 進小数

浮動小数点数の値の範囲は、次に示すように、0 < exponent < 2047 です。

$$-1^{s} * (1.fraction) * 2^{exp\ -1023}$$

倍精度実数値には、次の特殊な値が含まれます。

指数部	仮数部	説明
0	ゼロ	ゼロ値
0	非ゼロ	非正規化数
2047	ゼロ	符号付き無限大
2047	非ゼロ	非数

正規化数の範囲は、約 2.2 x 10-308 ～ 1.8 x 10307です。

例

次の例では、1.0 の値を返します。

```
DBLB($CHR(^H3F)+$CHR(^HF0)+$CHR(0)+$CHR(0)+$CHR(0)+$CHR(0)+$CHR
(0)+$CHR(0))
```

関連キーワード

ASC

$DBLB

FLTB

$FLTB

INTB

TRANSB

VAL

DCB

BCD を等しい整数値に変換する実数値関数です。

構文

DCB(value)

使用上の留意事項

4 桁までのBCD を変換できます。多くの場合、DCB 関数は、デジタル入力信号線の入力値をデコードするために、BITS 実数値関数とともに使用されます。

パラメータ

value 最大 4 桁の BCD を示すバイナリビットパターンとして解釈される実数値です。

注: 1 つの桁でも有効な BCD でない場合（1 桁が 9 以上の場合）、エラーが報告されます。

例

次の例では、外部入力信号 1001～1008（8 ビット入力）が外部デバイスから 2 つの BCD を受信した場合、指定された信号の BCD 入力と等価な整数値を実数値変数「input」に設定します。

```
input = DCB(BITS(1001, 8))
```

関連キーワード

BITS

BCD

$DECODE

指定されたブレーク文字で決定される文字列の一部を抽出する文字列関数です。

構文

$DECODE ($string_var, string_exp, mode**)**

使用上の留意事項

$DECODE は、入力された $string_var 変数を修正し、文字列値を返します。

ブレーク文字は、入力文字列の文字にかかわらず、常に評価されます。

ブレーク文字は、文字を定義する文字列内の他の文字とは独立した個々の文字として処理されます。

パラメータ

$string_var スキャンされる文字列を含む文字列変数です。関数が処理された後、関数値として返されていない元の文字列の一部がこの変数に含まれます。

注意：このパラメータは、関数によって修正されますが、文字列定数または式として指定することはできません。

プログラムが関数値の格納先として同じ変数を使用した場合、変数には、関数の戻り値が格納されます。

string_exp 入力文字列値の対象とする部分文字列を分離する、各ブレーク文字を定義する文字列定数、変数、または式です。この文字列内の文字の順番は、関数の演算には影響しません。

mode 関数が実行する演算を制御するオプションの実数値、変数、または式です。mode の値は、-3、-2、0、1 です。

mode がマイナス、0、または省略されている場合、最初のブレーク文字までの文字は、入力文字列から削除され、関数の出力として返されます。

mode が 0 以上の場合、最初のブレーク文字以外までの文字は、入力文字列から削除され、関数の出力として返されます。この場合、入力文字列のすべてのブレーク文字が返されます。

詳細

この関数は、入力文字列をスキャンし、ブレーク文字のグループで区切られた最初の部分文字列を返します。関数によって返された後、サブストリングは、入力文字列から削除されます。

返された文字列と削除された文字列は、ブレーク文字なし（mode 0）、またはすべてのブレーク文字を含む（mode 1）ことができます。$DECODE は、目的のサブストリングに対して最初のブレーク文字までのすべての文字を戻し、削除できます。または通常破棄されるすべてのブレーク文字を戻し、削除できます。

mode を変更すると、入力文字列から目的の文字グループを抽出できます（以下の最初の例を参照）。

mode が -2、-3 の場合、最初のブレーク文字までのブレーク文字以外のすべての文字と最初のブレーク文字をコピーします。

mode が -2 の場合、以下のステートメントと同等の意味になります。

```
$s = $DECODE($i,$break,0)    ;最初のブレーク文字までを抽出
$s = $s+$MID($i,1,1)         ;最初のブレーク文字に追加
$i = $MID($i,2,127)          ;ブレーク文字を削除
```

以下のステートメントには同じ機能があります。

```
$s = $DECODE($i,$break,-2) ;最初のブレーク文字までを抽出
```

ブレーク文字がある場合、modeが -3 でも -2 の場合と同等の意味になります。しかし、mode が -3 で入力文字列にブレーク文字がない場合、空の文字列を返し、入力文字列は変更されません。

例

次の例では、文字列「$input」から連続した数字を抽出します。文字列は、スペースとコンマの組み合わせで区切られた数字と仮定しています。

DO 文内の最初の例では、変数「$temp」を最初の数字を含む「$input」のサブストリングに設定し、「$input」からサブストリングを削除しています。VAL 関数は、数字列を実数値配列の次の要素に割り当て対象の実数値に変換するために使用されています。$DECODE 関数は、数字を区切る文字を抽出するために2回使用されています。見つかった文字は無視されます。

```
i = 0                                          ;配列のインデックスを設定
DO
    $temp = $DECODE($string_var," ,",0)        ;数字列を抽出
    value[i] = VAL($temp)                      ;実数値に変換
    $temp = $DECODE($string_var," ,",1)        ;スペースとコンマを破棄
    i = i+1                                    ;配列のインデックスを次に進める
UNTIL $string_var == ""                        ;入力が空になると停止
```

「$String_var」がスペース、コンマ、またはスペースとコンマの組み合わせで区切られた文字列とする連続した数値を含む場合、以下のステートメントを実行すると、「value」配列の最初の 4 つの要素は、次に示す値になります。

```
$string_var = "1234.93465.2, .4358,3458103"

value[0] = 1234.0
value[1] = 93465.2
value[2] = 0.4358
value[3] = 3458103.0
```

また、入力文字列変数（$string_var）には、空の文字 （""）が含まれます。

前述のように、通常、$DECODE 関数を使用する場合、入力、出力変数として 2 つの文字列変数を使用します。最初のブレーク文字までの文字のみを対象とし、以降の文字をすべて破棄する場合、同じ変数を入力、出力変数として使用できます。

次の例では、プログラマーが入力文字列の最後にあるすべての空白文字（スペースとタブ）を廃棄するため、等号を挟んだ両側で同じ変数を使用しています。

> **注:** ブレーク文字は、スペース文字とタブ文字を含む文字列式で指定されます。

```
$line = $DECODE($line," "+$CHR(9),0)    ;Discard trailing
blanks
```

関連キーワード

$TRUNCATE

$MID

$DEFAULT

現在または最初のシステムのデフォルトデバイス、ユニット、ディスクファイルにアクセスするためのディレクトリパスを含む文字列を返す文字列関数です。

構文

$DEFAULT (mode, empty_arg)

使用上の留意事項

mode を省略する場合でも、丸括弧を含める必要があります。

パラメータ

mode	返されるデフォルトのパスを指定するオプションの実数値、変数、または式（整数として解釈）です。 パラメータを省略、または値が 0 の場合、現在のシステムのデフォルトパスが返されます。 パラメータの値が 1 の場合、返されるデフォルトパスは、eV+ システムがディスクから起動した直後に有効となるパスとなります。
empty_arg	この値は NULL にしてください。

詳細

システムのデフォルトデバイス、ユニット、ディレクトリパスは、CD または DEFAULT コマンドで設定できます。$DEFAULT 関数は、次に示すような文字列として、現在または最初のデフォルト値を返します。

```
device>disk_unit:directory_path
```

この文字列には、設定済みの以下の情報の一部が含まれます。

device	次のいずれかの方法で設定できます。 ディスク　ローカルディスク システム　ディスクデバイス、ドライブ、サブディレクトリパスは、現在、DEFAULTモニタコマンドで設定されます。 **追加情報:** 有効なデバイスの詳細については、 ATTACHコマンドを参照してください。
disk_unit	ディスクの単位は DEFAULTコマンドで指定されます。単位を指定していない場合、コロン(:)は省略されます。

directory_path	デバイスと単位に続き、DEFAULTコマンドでの入力した値です。 追加で指定していない場合、ディレクトリパスは省略されます。

例

次の例では、デフォルトのデバイス仕様を DISK>D:¥TEST¥ と指定し、確認のためターミナルに表示しています。

```
DEFAULT = DISK>D:\TEST\ LISTS $DEFAULT()
```

関連キーワード

CD

DEFAULT

DEFINED

変数が定義されているかを判定する実数値関数です。

構文

DEFINED (var_name)

パラメータ

var_name	ロケーション、文字列、または実数値変数の名前です。スカラー変数と配列変数のどちらも可能です。ロケーション変数は、変換、精度ポイント、またはベルト変数にもなります。

詳細

指定された変数の値をテストします。値が空の場合、この関数は値 TRUE を返します。それ以外の場合は、値 FALSE を返します。

配列変数に対して特定の配列要素を指定した場合、配列の 1 つの要素がテストされます。配列要素が指定されていない場合、この関数は、いずれかの配列要素が定義されていれば TRUE を返します。

> **注:** CALL ステートメントの引数リストに実数値以外の引数（文字列、ロケーション、変換など）が指定された場合、変数が定義されているかどうかをテストできます。コールされたプログラム内で、未定義の実数値以外の引数に値を割り当てることはできません。未定義の実数値以外の引数に値を割り当てようとした場合、未定義値エラーメッセージが返されます。

ユーザー入力のテストに DEFINED を使用する場合、次の例に示すようなテストの前に、デフォルトの値を変数に割り当てておいてください。

例

次のステートメントは、変数「base_part」が定義されている場合、値 TRUE を返します。

```
DEFINED(base_part)
```

次のステートメントは、配列「corner」のいずれかの要素が定義されている場合、値 TRUE を返します。

```
DEFINED(corner[])
```

関連キーワード

STATUS

TESTP

DEST

現在のロボットモーションに対して、計画された目的地を示す変換値を返す変換関数です。

構文

DEST

使用上の留意事項

DEST 関数は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

詳細

モーションが中断されたときは、DEST はロボットがそこに向けて移動していた位置を返します。これは、モーションキーワードに適用されますが、次の事項の検討が必要です。

- 名前付きロケーションに対するモーション

```
MOVE start
MOVES #part[10]
```

> **注:** 2 番目のステートメントはプレシジョンポイントロケーション変数を参照していますが、DEST 関数は、そのモーション中の、変換値を返します。

- 名前付きロケーション、または現在のロボットのロケーションに関連して定義されたロケーションに対するモーション。

```
APPROS drop, 50.00
DEPART 30.00
MOVE SHIFT(HERE BY 50,0,10)
```

- 特別ロケーションに対するモーション。

```
READY
```

DEST 関数の返すロケーション値は、ロボットのモーションが何らかの理由で中断している場合、ロボットが停止するロケーションと同じとは限りません。例えば、非常停止操作が発生した場合、ロボットはただちに停止しますが、DEST 関数はロボットがそこに向けて動作していたロケーションを返します。

eV+ が目的地に到達できないと判断しモーションが開始されていない場合、DEST 関数は目標位置をセットしません。

例

次の例では、REACTI キーワードで開始されるリアクションで中断されているモーションを継続します。自動的にコールされるサブルーチンには、次に示すように、割り込みを処理し、元のモーションを再開するステップが含まれる可能性があります。

```
SET save = HERE
SET old.dest = DEST
old.speed = SPEED(3)
DEPART 50.0
       .
       .
       .
APPRO save, 50.0
MOVES save
SPEED old.speed
MOVES old.dest
```

関連キーワード

HERE

#PDEST

SELECT（実数値関数）

SELECT（プログラムコマンド）

DEVICE

指定されたデバイスから取得した実数値を返す実数値関数です。値は、デバイスやパラメータに応じて、データ、ステータス情報になります。

構文

DEVICE (type, unit, error, p1, p2,...)

使用上の留意事項

構文には、固有のデバイスの種類、情報のリクエストに対してのみ有用なオプションのパラメータが含まれます。

パラメータ

type	参照されるデバイスのタイプを示すオプションの実数値、変数または式です。整数値と解釈されます。以下のタイプを使用できます。 0 = ベルトエンコーダ 1 = （今後の使用のための予備） 2 = フォースプロセッサボード 3 = ロボットデバイス 4 = 画像センサ 5 = （今後の使用のための予備）
unit	デバイスユニット番号を示す実数値です。この値の範囲は 0～-1 にする必要があります。-1 は指定された種類のデバイスの最大数です。指定された種類のデバイスが1つしかない場合、この値を 0 になります。
error	この関数が成功または失敗したかを示す標準のシステムエラー番号を受け取るオプションの実数値変数です。このパラメータが省略されている場合、デバイスエラーが発生すると、プログラムの実行は停止されます。errorを指定している場合、プログラムでエラーを確認、検出する必要があります。
p1, p2, ...	リクエストの一部としてデバイスに送信されるオプションの実数値です。指定された値の数と値の意味は、特定のデバイスの種類によって異なります。

詳細

DEVICE 関数は、外部デバイスからのデータとステータス情報を返します。

例

次の例では、最新のパラメータが 1 の時に現在のベルトエンコーダ位置をキャプチャし、最新のパラメータが 4 の時にバッファからの次のラッチされたエンコーダ位置をキャプチャします。

```
IF (sv.emulate.mode) THEN
    latch.num = 1
```

```
    latch.value = DEVICE(0,pick.encoder[0]-1,sts,1)
ELSE
    latch.value = DEVICE(0,pick.encoder[0]-1,sts,4)
END
SETBELT %pick.belt[0] = latch.value
```

関連キーワード

SETDEVICE

DISTANCE

2 つのロケーション値で定義されるポイント間の距離を決定する実数値関数です。

構文

DISTANCE (location_1, location_2)

使用上の留意事項

X、Y、Z 軸のみ考慮されます。

一方のロケーションを複合変換として表現できます。

パラメータ

location_1	最初の対象ポイントを定義する変換値です。関数、変数、または複合変換として使用できます。
location_2	2番目の対象ポイントを定義する変換値です。関数、変数、または複合変換として使用できます。

詳細

指定された2つのロケーションで定義されるポイント間の距離を mm 単位で返します。ロケーションを指定する順番は関係ありません。また、ロケーションで定義される方向は、戻り値に影響しません。

例

次の例では、ロボットツールポイントの現在地と変換部で定義されたポイント間の距離を実数値変数 x の値に設定します。

```
x = DISTANCE(HERE, part)
```

関連キーワード

IDENTICAL

DURATION

モーションDURATION 仕様の 1 つに対する現在の設定を返す実数値関数です。

構文

DURATION (select)

使用上の留意事項

DURATION 関数は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

パラメータ

select どの期間の値を返すかを決める値の実数式です。

詳細

この関数は、選択されたパラメータ値に対応する、ユーザー指定のロボットモーション最小期間を秒単位で返します。モーション期間の仕様については、DURATION プログラムコマンドを参照してください。

次に示すように、返された期間をいつ適用するかをさまざまな選択値で決定します。選択パラメータの許容範囲は、2～4 です。このパラメータに対してそれ以外の値は無効となります。

Select	返されるDURATION値
2	永続的なロボットモーション最小期間（DURATION ...ALWAYSステートメントで設定）
3	現在または最後のモーションに対する一時的なモーション期間
4	次のモーションに使用される一時的なモーション期間

例

次の例では、実数値関数を使用して DURATION の設定が 2 かどうかを確認して、そうでない場合は、プログラムコマンドを使用して 2 に設定します。

```
SELECT ROBOT = 1

IF DURATION(2)<>2 THEN
    DURATION 2 ALWAYS
END
```

関連キーワード

CONFIG

DURATION (プログラムコマンド)

SELECT (プログラムコマンド)

SELECT （実数値関数）

DX

指定された変換値の X 軸の成分を返す実数値関数です。

構文

DX (location)

パラメータ

location	目的の成分の変換値です。関数、変数、または複合変換として使用できます。

詳細

この関数は、指定された変換値の各 X 軸の成分を返します。

> **注:** また、変換値の変位成分を取得するために、DECOMPOSE コマンドも使用できます。回転成分が必要な場合、DECOMPOSE コマンドを使用する必要があります。1 つ以上の要素が必要で、ロケーションが複合変換の場合、DECOMPOSE はより効果的です。

例

次の成分を持つ変換「start」があるとします。

```
125、250、-50、135、50、75
```

次の例では、125.00 の値を返します。

```
DX(start)
```

関連キーワード

DECOMPOSE

DY

DZ

RX

RY

RZ

DY

指定された変換値の Y 軸の成分を返す実数値関数です。

構文

DY (location)

パラメータ

location	目的の成分の変換値です。関数、変数、または複合変換として使用できます。

詳細

この関数は、指定された変換値の各 Y 軸の成分を返します。

> **注:** また、変換値の変位成分を取得するために、DECOMPOSE コマンドも使用できます。回転成分が必要な場合、DECOMPOSE コマンドを使用する必要があります。1 つ以上の要素が必要で、ロケーションが複合変換の場合、DECOMPOSE はより効果的です。

例

次の成分を持つ変換「start」があるとします。

```
125、250、-50、135、50、75
```

次の例では、250.00 の値を返します。

```
DY(start)
```

関連キーワード

DECOMPOSE

DX

DZ

RX

RY

RZ

DZ

指定された変換値の Z 軸の成分を返す実数値関数です。

構文

DZ (location)

パラメータ

location	目的の成分の変換値です。関数、変数、または複合変換として使用できます。

詳細

この関数は、指定された変換値の各 Z 軸の成分を返します。

> **注:** また、変換値の変位成分を取得するために、DECOMPOSE コマンドも使用できます。回転成分が必要な場合、DECOMPOSE コマンドを使用する必要があります。1 つ以上の要素が必要で、ロケーションが複合変換の場合、DECOMPOSE はより効果的です。

例

次の成分を持つ変換「start」があるとします。

```
125、250、-50、135、50、75
```

次の例では、-50.00 の値を返します。

```
DZ(start)
```

関連キーワード

DECOMPOSE

DX

DY

RX

RY

RZ

ENCLATCH

システムに含まれているエンコーダの、最新のラッチが発生した時点でのエンコーダ値を返す実数値関数です。

構文

ENCLATCH(select)

使用上の留意事項

設定されていないエンコーダをselect パラメータで指定してENCLATCH キーワードを実行した場合、エラーが返されます。

パラメータ

select	エンコーダの id を指定するオプションの整数、式、または実数値変数です。値の範囲は 1～116 です。

詳細

ENCLATCH キーワードは、最新の外部トリガーが発生したときのエンコーダの位置を示す実数値を返します。LATCHEDエンコーダの他の情報を読み出すのど同様に、DEVICEキーワードをエンコーダのラッチされた値を読み出すためにも使用できます。

外部トリガーの作用は、Sysmac Studio で設定できます。詳細については、 *『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

例

次の例では、エンコーダ 1 の位置を返して、その値を「Latch_pos」に代入します。

```
encoder_id = 1
Latch_sig = LATCHED(-1*encoder_id)
Latch_pos = ENCLATCH(encoder_id)
```

関連キーワード

LATCHED

$ENCODE

出力仕様から作成された文字列を返す文字列関数です。

構文

$ENCODE (output_specification, output_specification, ...)

パラメータ

output specification（出力仕様）には、以下の構成要素を使用できます。コンマで区切れば、順番は関係ありません。

- 文字列式。
- 表示される値の決定のために評価される
- 出力メッセージのフォーマットを決定するフォーマット制御指定子です。

詳細

この関数はプログラム内で使用できるTYPEコマンドによって通常生成される文字列を生成します。$ENCODE 関数は、文字列値を生成しますが、出力はしません。

数値の表示制御には、次のフォーマット指定子を使用できます。これらの設定は、別の指定子を使用するまで、残りの関数パラメータリストに対して有効です。

これらすべての指定子では、値が指定のフィールド幅で表示するには大きすぎる場合、フィールドはアスタリスク文字（*）で埋められます。

/D	末尾のゼロと先頭のスペース1文字を除くすべてを省略する以外は、/G15.8(以下参照)と同等のデフォルトフォーマットを使用します。

以下のフォーマット仕様では、フィールド幅として 0 を許可しています。0 の場合、値に応じて実際のフィールドサイズを変化させ、先頭のスペースはすべて削除されます。1 行の文字内または行の最後に値を表示させる場合に、役立ちます。

/En.m	小数部が m 桁のフィールド n スペース幅内で科学的記数法(-1.234E+02 など)を用いて値をフォーマットします。n が 0 以外の場合、マイナス記号(表示値がマイナスの場合)、小数点左側の 1 桁、小数点(m が 0 以外の場合)、m 桁、指数用に 4 または 5 文字を含む十分なスペースが必要になります。
/Fn.m	小数部が m 桁のフィールド n スペース幅内で固定小数点表現(-123.4 など)を用いて値をフォーマットします。
/Gn.m	フィールド n スペース幅で可能な場合、小数部の m 桁で F フォーマットの値をフォーマットします。それ以外の場合、/En.m フォーマットを使用します。

/Hn　　フィールドn スペース幅内で値を16 進の整数にフォーマットします。

/In　　フィールドn スペース幅内で値を10 進の整数にフォーマットします。

/On　　フィールドn スペース幅内で値を8 進の整数にフォーマットします。

文字列内に特殊文字を挿入する場合、次の指定子を使用できます。

/Cn　　キャリッジ・リターン(CR) と改行(LF) 文字をn 回含みます。

$ENCODE 関数から返された文字列をモニタ画面に出力する場合、フォーマット制御指定子が関数パラメータリストの最初または最後にあると、n 個の空行が表示されます。それ以外の場合、n-1 個の空行が表示されます。

/Un　　結果の文字列をモニタ画面に出力する場合、カーソルをn 行分上に移動させる必要がある文字を含みます。

/Xn　　n 個のスペースを含みます。

例

次の例では、$output に含まれる文字列に対して、カウント値のフォーマット済みの表現を追加しています。

```
$output = $output+$ENCODE(/F6.2, count)
```

次の例では、PROMPT コマンドから出力に対してフォーマット制御指定子を追加する方法を示しています。この例では、プロンプトメッセージの一部としてモータの値を表示します。

```
PROMPT $ENCODE("Start motor #",/I0,motor," (Y/N)? "),
$answer
```

この PROMPT コマンドは、モータの値が 3 のとき、次のユーザープロンプトを表示します。

```
Start motor #3 (Y/N)?
```

関連キーワード

PROMPT

TYPE

$ERROR

指定のエラーコードに関連するエラーメッセージを返す文字列関数です。

構文

$ERROR (error_code)

パラメータ

error_code	エラー状態を特定する実数値式です。マイナスの値を持ちます。

詳細

IOSTAT関数やメッセージ文字列、またERROR実数値関数により返されるエラーコードは、この関数により、対応する eV+ のエラーメッセージに変換することができます。ERROR実数値関数は、-1000 以下のエラーコードに対するエラーメッセージの変数部分を特定するために使用されなければなりません。

すべての eV+ エラーメッセージとエラーコードのリストについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、I/O エラーが発生した場合にエラーメッセージを表示します。

```
READ (5) $input
IF IOSTAT(5) < 0 THEN
    TYPE "I/O error during read: ", $ERROR(IOSTAT(5))
    HALT
END
```

関連キーワード

ERROR

IOSTAT

REACTE

ERROR

プログラムの実行を停止させた、または REACTE 命令を発生させた、最近のシステムメッセージ番号を返す実数値関数です。

構文

ERROR (source, select**)**

使用上の留意事項

REACTE ステートメントを実行した場合、現在のタスクで発生したシステムメッセージはクリアされるため、ERROR 関数を実行しても期待した通りにはメッセージは発生しません。

何らかの理由で電源が無効な間に、有効なハイパワーリクエストが発行されたことで発生する、すべての非同期メッセージの受信に FIFO バッファを利用できます。この関数を使用して、バッファにアクセスできます。

ロボットがパワーダウン初期化状態の間、非同期メッセージバッファは無効です。source パラメータが1,000 より大きい場合、この関数をコールする前に、STATE (1)が0 以外になるまでユーザープログラムでウェイトする必要があります。

> **追加情報:** エラーの詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

パラメータ

source

-2 現在のロボットに対する追加のメッセージコードを返します。

ERROR(-2, 0) 標準の eV+ メッセージを返します。

ERROR(-2, 1) 現在のロボットに対するモーターマスクを返します。このビットマスクは、ステートメント ERROR(-2,0)が返すメッセージ番号に対して参照されるモーターを示します。LSB はモーター1などを示します。ステートメントが ERROR(-2, 1)=0 の場合、メッセージは、特定のモーターには関連しません。

-1 ERROR 関数を実行するプログラムで発生した最新のメッセージ番号を返します。

0 タスク 0 として実行されたプログラムで発生した最新のメッセージ番号を返します。

0 < source ≤ 63

該当するタスク番号として実行されたプログラムで発生した最新のメッセージ番号を返します。

1001 < source ≤ 1021

非同期 FIFO バッファの要素 n-1000。*n* は以下の意味になります。

1 最新の項目

2 2 番目に新しい項目など

FIFO バッファの要素がない場合、0 を返します。FIFO バッファの有効選択パラメータ値は、0、1、3 です。

ロボットがパワーダウン初期化状態の間、非同期 FIFO バッファは無効です。source パラメータが1,000 より大きい場合、ERROR 関数を使用する前に、ユーザープログラムで STATE(1)が 0 以外の間を返すまでウェイトする必要があります。

2000 最新の eV+ システムメッセージを返します。

2001～2008

2001～2008 がロボット1～8 に対応する特定のロボットに関連した最新のメッセージを返します。

select　返されるメッセージ情報を選択する整数値として解釈される実数値、変数、式です。このパラメータが省略された場合、値は 0 になります。

0　I/O エラーを除く、最新のプログラム実行エラーのメッセージ番号を返します。指定されたプログラムタスクについては、IOSTAT を参照してください。

1　指定のプログラムタスクで発生した最新のメッセージのコードが 1100～-1199 の範囲にある場合、該当するメッセージの可変部分を数値として返します。最新のメッセージのコードが -1000～-1099 の範囲にある場合、メッセージ元のジョイントやモーターを示すビットマスクとして、該当するメッセージの可変部分を数値として返します。メッセージに可変部分がない場合、0 を返します。以下の select = 3 を参照してください。

2　指定されたプログラムタスクによって実行された MCS プログラムコマンドで発生した最新メッセージ番号を返します。

3　指定のプログラムタスクで発生した最新のメッセージに関連するロボットの番号を返します。メッセージが指定のロボットに関連していない場合、0 を返します。select = 1 を参照してください。

詳細

eV+ タスクはロボットの電源を無効にしたメッセージにアクセスできます。これらのメッセージには、以前にモニタウィンドウにのみ出力された非同期メッセージが含まれます。

この関数は、REACTE サブルーチンプログラム内でREACTEがトリガーされた理由を判断するために役立ちます。

> **注:** ERROR 関数は、IOSTAT 関数がレポートするメッセージについては、レポートしません。

eV+ のエラーとエラー番号の一覧については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

select パラメータが 1 の場合、この関数の戻り値は、6 ビットの数値として解釈されます。次の例では、値がどのように解釈されるかを示しています。

例

次の例では、以下の条件でエラーコードに該当するエラーメッセージを返します。

- この例で「code」は、ERROR(n)、または IOSTAT(lun)からの基本エラーコードです。
- この例で「vcode」は、ERROR (n,1)からのエラーコードの可変部分です。
- この例で「robot」は、ERROR(n,3)からのエラーに関連したロボット番号です。
- 「$msg」は、該当するエラーメッセージです。Null の場合もあります。

```
.PROGRAM error.string(code, vcode, robot, $msg)
        AUTO i, n
        $msg = ""
        IF code < 0 THEN
            $msg = $ERROR(code)

            IF (-1100 < code) AND (code <= -1000) THEN
                n = 1
                FOR i = 1 TO 7
                    IF vcode BAND n THEN
                        $msg = $msg+$ENCODE(i)
                    END
                    n = 2*n
                END
           END

            IF (-1200 < code) AND (code <= -1100) THEN
                $msg = $msg+$ENCODE(vcode)
            END

            IF robot AND (SELECT(ROBOT,-1) > 1) THEN
                $msg = $msg+" (Robot"+$ENCODE(robot)+")"
            END
        END

        RETURN
.END
```

関連キーワード

$ERROR

IOSTAT

MCS

REACTE

FALSE

eV+ で使用される、論理的な偽を表すための値を返す実数値関数です。

構文

FALSE

詳細

この名前付き定数は、真、偽の条件指定が必要な状況で役立ちます。戻り値は 0 です。

例

次のループプログラムの例では、サブルーチンのサイクルが実数値変数「continue」に FALSE 値を返すまで実行を継続します。

```
DO
    CALL cycle(continue)
UNTIL continue == FALSE
```

関連キーワード

OFF

ON

TRUE

$FLTB

実数値を単精度IEEE浮動小数点形式の2 進数で表現した4 バイトの文字列を返す文字列関数です。

構文

$FLTB (value)

パラメータ

value	IEEE 浮動小数点 2 進数に変換する対象の値、実数式です。

詳細

実数値は、IEEE単精度標準浮動小数点形式で2進数に変換されます。この 32 ビットの値は、連続した 4 文字（1 文字 8 ビット）として文字列に格納されます。IEEE 754規格の浮動小数点形式については、FLTB実数値関数を参照してください。

この関数は、主に実数値をデータファイルの出力レコードの2 進数に変換するために使用されます。

例

次の例では、以下に示すように文字列を返します。

```
$FLTB(1.215)
```

次のように文字列として返されます。

```
$CHR(^H3F)+$CHR(^H9B)+$CHR(^H85)+$CHR(^H1F)
```

関連キーワード

$CHR

DBLB

$DBLB

FLTB

$INTB

LNGB

$LNGB

TRANSB

FLTB

4 バイトの文字列をIEEE 単精度浮動小数点数として解釈した値を返す実数値関数です。

構文

FLTB ($string, first_char)

パラメータ

$string 変換される4 バイトを含む文字列式です。

first_char 4 バイトの文字列の開始位置を指定するオプションの実数式です。

first_char が省略、または値が 0、1 の場合、文字列の先頭から4 バイトが抽出されます。first_char が 1 より大きい場合、抽出するバイトの開始文字位置として解釈されます。

例えば、値 2 は、2、3、4、5 番目の 4 バイトが抽出されます。

first_char に入力文字列の末尾を超えた 4 バイトを指定した場合、エラーになります。

詳細

この関数は、主に入力データレコードのバイナリ浮動小数点数を eV+ が内部的に使用できる値に変換するために使用されます

指定された文字列の連続した 4 バイトは、IEEE 標準形式の単精度（32ビット）浮動小数点数として解釈されます。この 32 ビットフィールドは、次のように解釈されます。

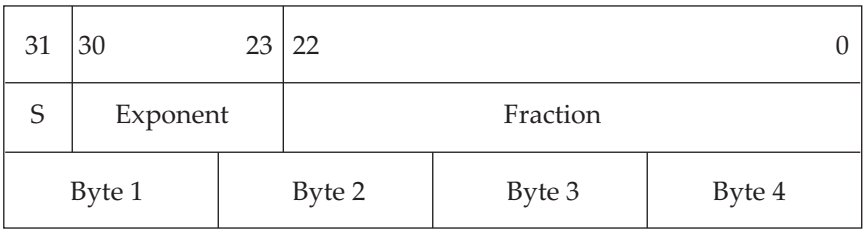


図3-2. 32 ビット浮動小数点形式

表3-2. 32 ビット浮動小数点形式の説明

項目	説明
S	符号ビット（0 = 正、 1 = 負）

項目	説明
指数部	2 進指数、-127 でバイアス
小数部	小数点の左側を 1 とする 2 進小数

浮動小数点数の値の範囲は、0 < exp < 255 です。

$$-1^{s} * (1.fraction) * 2^{exp\ -127}$$

exp = 0 の場合、値は 0、exp = 255 の場合、オバーフローエラーになります。

例

次の例では、1.0 を返します。

```
FLTB($CHR(^H3F)+$CHR(^H80)+$CHR(0)+$CHR(0))
```

次の例では、-3.0 を返します。

```
FLTB($CHR(^HC0)+$CHR(^H40)+$CHR(0)+$CHR(0))
```

関連キーワード

ASC

DBLB

$DBLB

$FLTB

INTB

LNGB

$LNGB

TRANSB

VAL

FRACT

引数の小数部を返す実数値関数です。

構文

FRACT (value)

パラメータ

value	この関数が返す小数部の実数式です。

詳細

実数値の小数部は、値が指数表記で記述されていない場合、小数点の右側部分になります。

戻り値の符号は、関数と同じ符号です。

例

次の例では、0.123 を返します。

```
FRACT(0.123)
```

次の例では、-0.462 を返します。

```
FRACT(-5.462)
```

次の例では、0.25 を返します。

```
FRACT(1.3125E+2)
```

関連キーワード

INT

FRAME

4 つの位置で定義される変換値を返す変換関数です。

構文

FRAME (location_1, location_2, location_3, location_4)

パラメータ

location_1 位置が計算されたフレームの X 軸の定義に使用される変換、複合変換、または変換値関数です。

正のX軸は、ロケーション 1 からロケーション 2 の方向にある、ロケーション 1 とロケーション 2 で定義される位置を通過する線に対して平行です。

location_2 位置が計算されたフレームの X 軸の定義に使用される変換、複合変換、または変換値関数です。

正のX軸は、ロケーション 1 からロケーション 2 の方向にある、ロケーション 1 とロケーション 2 で定義される位置を通過する線に対して平行です。

location_3 位置が計算されたフレームの Y 軸の定義に使用される変換、複合変換、または変換値関数です。

X-Y 平面は、ロケーション 1、2、3 で定義される位置を含む平面に対して平行です。

location_4 位置が計算されたフレーム変換の位置として返される変換、複合変換、または変換値関数です。

起点は、ロケーション 4 で定義されるポイントです。

詳細

ロボットを使用して、X、Y、Z の位置を正確に定義することはできますが、多くの場合、方向を正確に定義するのは難しくなります。パレタイジングなどの用途では、位置と方向が 4 つの位置で決定されるベース変換を正確に定義する場合、FRAME 関数は、非常に役立ちます。

例

次の例では、起点とする同じ X、Y、Z 位置、中心から xに向かう線に対して平行の X 軸、中心から y に向かう線とほぼ同じ方向にある Y 軸を持つ、変換 base.-frame を定義しています。

```
SET base.frame = FRAME(center, x, y, origin)
```

関連キーワード

TRANS

FREE

記憶領域の未使用の空きメモリ容量を返す実数値関数です。

構文

FREE (memory, select)

機能

記憶領域の未使用の空きメモリ容量を返します。

パラメータ

memory　対象のシステムメモリを指定するオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略された場合、値は 0 とみなされます。

- 0: プログラムメモリ
- 1: 今後の使用のための予備
- 2: 今後の使用のための予備

select　返されるメモリの情報を指定するオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略された場合、値は 0 とみなされます。

- 0: 使用可能メモリの割合(%)
- 1: 使用可能メモリ(キロバイト)(1 キロバイト= 1024 バイト)

詳細

この関数は FREE モニタコマンドが表示する情報を返します。FREE モニタコマンドとは異なり、この関数はmemoryとselectの両パラメータで決定される値だけを返します。

例

次の例では、モニタウィンドウに使用可能なメモリ容量（キロバイト）を表示します。

```
memory = FREE(0,1)
TYPE memory
```

関連キーワード

FREE（モニタコマンド）

GETC

デバイス、または指定の論理ユニットの入力レコードから入力した次の文字（バイト）を返す実数値関数です。

構文

GETC (lun, mode)

使用上の留意事項

標準の変数長レコード入出力の場合、プログラムを論理ユニットに接続する必要があります。

パラメータ

lun

アクセスされるデバイスを指定する実数値、変数または式です。整数として解釈されます。

ユニット番号に関する説明については、ATTACH コマンドを参照してください。

mode

読み取り動作モードを指定する実数値、変数、または式です。整数として解釈されます。このモードは端末に対してのみ使用されます。

その値は以下に示すように連続するビットフラグと解釈されます。モード値が指定されていない場合、すべてのビットはクリアされていると仮定されます。

ビット 1(最下位ビット): 入力待機無効(マスク値 = 1)

このビットが OFF の場合、次のバイトを受信するまでプログラムの実行は一時停止します。このビットが ON で、どのバイトも使用できない場合、関数はただちに「受信データなし」(-526)のエラーコードを返します。

> **注:** キューにバイトがある場合でも、最初の no-wait GETC で -526 エラーが返される場合もあります。

ビット 2 エコー無効(マスク値 = 2)

このビットが OFFの場合、ターミナルからの入力は、ソースにエコーバックします。このビットが ON の場合、文字はソースにエコーバックしません。

詳細

デバイスからの次のバイトが返されます。ディスクファイルなどのレコードオリエンテッドデバイスやモニタウィンドウから読み取る場合、レコードの最後にあるキャリッジリターンやラインフィード文字も返されます。ディスクファイルが最後に到達すると、Ctrl+Z 文字（10 進数の 26）が返されます。

ターミナルから読み取る場合、GETC はキーボードから入力された次の文字を返します。通常のターミナル制御機能を持つ、Ctrl+S、Ctrl+Q、Ctrl+O、Ctrl+W を除くすべての制御文字を読み取ります。

通常、返されるバイト値は、0～255（10 進数）の範囲内です。入力でエラーが発生した場合、マイナスのエラーコード番号が返されます。エラー番号に関する詳細情報については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

次の例では、コンマ、文字、制御文字、または I/O エラーを見つけるまで、ディスクファイルの文字を読み取ります。文字は、文字列変数 $field に付加されます。変数長レコードにアクセスする前に、ディスクファイルを開いておく必要があります。

```
$field = ""
c = GETC(5)
WHILE (c > ^H1F) AND (c <> ',) DO
    $field = $field+$CHR(c)
    c = GETC(5)
END
IF c < 0 THEN
    TYPE $ERROR(c)
    HALT
END
```

関連キーワード

ATTACH

READ

GET.EVENT

指定のタスクに対して設定されるイベントを返す実数値関数です。

構文

GET.EVENT (task)

使用上の留意事項

この関数を GETEVENT プログラムコマンドと混同しないでください。

パラメータ

task	返されるイベントに愛するタスクを指定するオプションの実数値、変数、式です。整数として解釈されます。有効範囲は、-1～6、または -1～64 です。パラメータを省略、または値が -1 の場合、現在のタスクが参照されます。

詳細

この関数は指定されたタスクのイベントを返します。task パラメータが -1 の値に設定されていると、現在のタスクのイベントを返します。

次に示すように、イベントは、ビットフラグのシーケンスとして解釈される値の中で返されます。

ビット 1（最下位ビット） I/O 完了（マスク値 = 1）

このビットが ON の場合、システムの入出力命令が完了したことを示します。詳細については、SET.EVENT キーワードおよび WAIT.EVENT キーワードを参照してください。

関連キーワード

CLEAR.EVENT

SET.EVENT

WAIT.EVENT

HERE

ロボットのツールポイントの現在のロケーションを示す変換値を返す変換関数です。

構文

HERE

使用上の留意事項

ロボットのフィードバック位置を表すように、各ジョイントのエンコーダーの瞬時値を読み取ることで、現在のロケーションを取得します。

HERE 関数は、関数を実行するタスクが選択しているロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

例

次の例では、ロボットの現在のロケーションと、ロボットが現在向かっている目標のロケーションとの間の距離を計算します。

```
dist = DISTANCE(HERE, DEST)
```

関連キーワード

DISTANCE

HERE(モニタコマンド)

HERE (プログラムコマンド)

SELECT (プログラムコマンド)

SELECT(実数値関数)

HOUR.METER

ロボットアワーメータの現在の値を返す実数値関数です。

構文

HOUR.METER(robot number)

詳細

ロボットアワーメータには、ロボットの電源がON になってから何時間経過したかが記録されています。メータは電源が ON になってから 1 時間半後に更新され、以降は 1 時間ごとに更新されます。

アワーメータは、ロボットコントローラの Robot Signature Card によって管理されます。Robot Signature Card が反応しない場合、HOUR.METER 関数は値 -1 を返します。

例

次の例では、ロボットアワーメータの現在の読取値を実数値変数「start」に代入します。

```
start = HOUR.METER
```

$ID

システム ID 文字列を返す文字列関数です。

構文

$ID (select)

パラメータ

select 返す ID 情報を指定する整数です。有効な整数値は以下に示す通りです。

整数	説明
-1	システム改訂メッセージを返します。
-2	eV+ システムの改訂符号と発行番号を返します。
-3	ビジョン改訂メッセージの文字列を返します。
-4	サーボ改訂メッセージの文字列を返します。

詳細

この関数は、要求されたシステムソフトウェアコンポーネントの改訂および日付を識別する文字列を返します。

例

次の例では、モニタウィンドウに eV+ システムの改訂符号と発行番号を表示します。

```
$string1=$ID(-2)
TYPE $string1
```

関連キーワード

ID（モニタコマンド）

ID（関数）

ID

現在のシステムの設定を指定する値を返す実数値関数です。

構文

ID (component , device, board)

使用上の留意事項

この関数は存在しないデバイスに対して値 0 を返します。存在しないデバイス番号は値 0 を返します。有効なデバイスに対して、要求された構成要素が有効でない場合、「Invalid argument」（無効な引数）エラーメッセージが報告されます。

パラメータ

component	実数値、変数または式であり、その値は、返される指定情報の構成要素を決定する整数として解釈されます。
device	指定されるデバイスを選択するオプションの実数値、変数、式です。整数として解釈されます。このパラメータが省略された場合、デバイス 1 (基本システム) が仮定されます。
board	device パラメータの値が 4 であるときに対象となる CPU を指定するオプションの整数です。このパラメータが省略された場合、ボード 1 (メイン CPU) が仮定されます。

詳細

ID 関数を使用することで、プログラムが ID モニタコマンドキーワードによって表示される情報にアクセスすることが可能になります。構成要素の値は、コマンドによって表示されるフィールドのものと同じです。

次の表を見て、パラメータの設定および ID 実数値関数によって返される関連情報について理解してください。

表3-3. ID パラメータの詳細

デバイス	component	board	返される情報
1: 基本システム	1: モデル番号	未使用	ロボットがサービスモードの場合、NJ シリーズロボット統合CPU ユニット のモデルコードが返されます。 ロボットがオペレーションモードの場合、ロボットのモデルコードが返されます。
	2: シリアル番号		NJ シリーズロボット統合CPU ユニットは 3000 を返します。 ロボットがサービスモードの場合、ロボットのシリアル番号が返されます。

デバイス	component	board	返される情報
	3: eV+ の内部メジャーバージョン		eV+ の内部メジャーバージョンが返されます。
	4: eV+ の内部マイナーバージョン		eV+ の内部マイナーバージョンが返されます。
	11: コントローラハードウェア設定		エミュレーションモードが有効な場合、値 4 が返されます。 エミュレーションモードが有効ではない場合、値 0 が返されます。
	14: eV+ 外部バージョン		値 3 が返されます（eV+3）。
2: T20 ペンダント	1: ペンダントのバージョン	未使用	ペンダントソフトウェアの完全なバージョン番号が返されます。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）が返されます。
	2: ペンダントのメジャーバージョン		ペンダントソフトウェアのメジャーバージョン番号が返されます。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）が返されます。
	3: ペンダントのマイナーバージョン		ペンダントソフトウェアのマイナーバージョン番号が返されます。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）が返されます。
	4: ペンダントのビルドバージョン		ペンダントソフトウェアのビルドバージョン番号が返されます。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）

デバイス	component	board	返される情報
			が返されます。
	5: ペンダントのレビジョンバージョン		ペンダントソフトウェアのレビジョンバージョン番号が返されます。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）が返されます。
	6: ペンダント接続状態		ペンダント接続の状態が返されます。 • -1: ペンダントは接続済みで、通信中 • 0: タイムアウトが発生し、ペンダントを検出することが出ません。 エミュレーションモードの実行中であるか、ペンダントが接続していない場合、エラー「Pendant Not Connected」（ペンダント未接続、-657）が返されます。
4: システムCPU	1: CPU の番号	CPU ボード (-32768～32767)	CPU の番号を返します。 存在しない CPU の場合は値 -1 を返します。 メイン CPU は値 1 を返します。
	5: CPU のタイプ	未使用	NJ シリーズロボット統合CPU ユニット は値 9 を返します。
8: 現在選択しているロボットのロボット設定	1: ロボットモデル	未使用	ロボットのモデルコードを返します。
	2: ロボットのシリアル番号		ロボットのシリアル番号を返します。
	3: モータ数		ロボットを構成しているモータ数を返します。
	4: 有効化されているロボットジョイントに対するビットフラグと解		有効化されているロボットジョイントに対するビットフラグと解釈される値を返します。 選択的に無効化可能なジョイントがロボットにない場合は、値 0 を返します。

デバイス	component	board	返される情報
	釈される値		

デバイス	component	board	返される情報
	5: キネマティックモジュール ID		ロボット制御モジュールの ID 番号を返します。
	7: 使用するために設定されたロボットジョイント数		使用するために設定されたロボットジョイント数を返します。
	8: 第 1 のロボットオプションワード		第 1 のロボットオプションワードを返します。 詳細については、ロボットオプションワード 122 ページを参照してください。
	9: 製品タイプ		ロボット製品タイプを返します。
	10: 動作計画に使用される直交座標軸数		動作計画の際に使用される軸数を返します。 この数値は、最大直交座標系速度および加速度を求めるためにロボットの加重モジュール配列において定義する必要のある値の数を示します。
	11: 第 2 のロボットオプションワード		第 2 のロボットオプションワードを返します。 詳細については、ロボットオプションワード 122 ページを参照してください。
	12: ロボットモジュールに関する情報		ロボットモジュールに関する情報を返します。ビット1(マスク1) だけが定義されています。 このビットがオンの場合、指定されたロボットは OMRON ロボットです。
	13: ロボットの安全レベル		ロボットに対して設定されている安全レベルを返します。 • 0: 安全レベルが設定されていない • 1: ISO 10218 およびEN954 によるカテゴリ1 ロボットシステムとして設定されている。 • 3: ISO 10218 およびEN954 によるカテゴリ3 ロボットシステムとして設定されている。

デバイス	component	board	返される情報
	15: eSeries ロボットタイプ		eSeries ロボットタイプを返します。 • 0 : eSeries Lite • 1 : eSeries Standard • 2 : eSeries Pro • 3 : sSeries
	16: ロボットセキュリティ ID の第 1 および第 2 の部分		次において「aaaa-bbbb」として示される、ロボットセキュリティ ID の第 1 および第 2 の部分を返します。 セキュリティ ID : aaaa-bbbb-cccc
	17: ロボットセキュリティ ID の第 3 の部分		次に示すロボットセキュリティID の第 3 の部分「cccc」を返します。 セキュリティ ID : aaaa-bbbb-cccc
	18: ロボットに現在割り付けられているタスクの数		ロボットに現在割り付けられているタスクの数を返します。
	19: ロボットジョイント制御		ロボットがジョイント制御モードで作動可能である場合は、値 -1 を返します。 ロボットがジョイント制御モードで作動可能である場合は、値 0 を返します。
10: ベルト	7: ベルト数	未使用	システムにおいて設定されているベルト数を返します。
11〜18: ロボット 1〜8 に対するロボット設定 (10 + ロボット番号)	1: ロボットモデル	未使用	ロボットのモデルコードを返します。
	2: ロボットのシリアル番号	未使用	ロボットのシリアル番号を返します。
	3: モータ数		ロボットを構成しているモータ数を返します。
	4: 有効化されているロボットジョイントに対		有効化されているロボットジョイントに対するビットフラグと解釈される値を返します。 選択的に無効化可能なジョイントがロボットにない場合は、値 0 を返しま

デバイス	component	board	返される情報
	するビットフラグと解釈される値		す。
	5: キネマティックモジュール ID		ロボット制御モジュールの ID 番号を返します。
	7: 使用するために設定されたロボットジョイント数		使用するために設定されたロボットジョイント数を返します。
	8: 第 1 のロボットオプションワード		第 1 のロボットオプションワードを返します。 詳細については、ロボットオプションワード 122 ページを参照してください。
	9: 製品タイプ		ロボット製品タイプを返します。
	10: 動作計画に使用される直交座標軸数		動作計画の際に使用される軸数を返します。 この数値は、最大直交座標系速度および加速度を求めるためにロボットの加重モジュール配列において定義する必要のある値の数を示します。
	11: 第 2 のロボットオプションワード		第 2 のロボットオプションワードを返します。 詳細については、ロボットオプションワード 122 ページを参照してください。
	12: ロボットモジュールに関する情報		ロボットモジュールに関する情報を返します。ビット1(マスク1) だけが定義されています。 このビットがオンの場合、指定されたロボットは OMRON ロボットです。
	13: ロボットの安全レベル		そのロボットに対して設定される安全レベルを返します。 • 0: 安全レベルが設定されていない • 1: ISO 10218 およびEN954 によるカテゴリ1 ロボットシステムとして設定されている。

デバイス	component	board	返される情報
			• 3: ISO 10218 およびEN954 によるカテゴリ3ロボットシステムとして設定されている。
	15: eSeries ロボットタイプ		eSeries ロボットタイプを返します。 • 0: eSeries Lite • 1: eSeries Standard • 2: eSeries Pro • 3: sSeries
	16: ロボットセキュリティ ID の第 1 および第 2 の部分		次に示すロボットセキュリティID の第1および第2 の部分「aaaa-bbbb」を返します。 および第 2 の部分を返します。 セキュリティ ID : aaaa-bbbb-cccc
	17: ロボットセキュリティ ID の第 3 の部分		次に示すロボットセキュリティID の第 3 の部分「cccc」を返します。 セキュリティ ID : aaaa-bbbb-cccc
	18: ロボットに現在割り付けられているタスクの数		ロボットに現在割り付けられているタスクの数を返します。
	19: ロボットジョイント制御		ロボットがジョイント制御モードで作動可能であるかどうかを示す値を返します。 • -1: ロボットはジョイント制御で動作可能 • 0: ロボットはジョイント制御で動作不可能

ロボットオプションワード

第 1 オプションワードの上位ビットの解釈について次の表で説明します。

表3-4. 第 1 ロボットオプションワード詳細

ビット番号	マスク値		説明
	10 進数	16 進数	
9	256	100	このビットが ON の場合、モータのリミットストップテストが有効です。

ビット番号	マスク値		説明
	10 進数	16 進数	
			これは、過剰なモータカップリングを持ち、モータのリミットストップデータ割り当てのあるロボットに対してのみ有効になります。
10	512	200	このビットが ON の場合、フリーモードパワー OFF が有効です。 フリーモード中に制限違反があると、ロボットパワーは無効になります。
11	1024	400	このビットが ON の場合、システムは起動時に自動で CALIBRATE モニタコマンドを実行します。
12	今後の使用のための予備		
13	4096	1000	このビットが ON の場合、システムは関節補間動作において直交座標空間内で静止障害物と衝突しないか確認します。 このビットが ON の場合、関節補間動作は直線動作と同じ計算上の負荷を持ちます。
14	8192	2000	今後の使用のための予備
15	今後の使用のための予備		
16			

第 2 オプションワードのビットの解釈について次の表で説明します。

表3-5. 第 2 ロボットオプションワード詳細

ビット番号	マスク値		説明
	10 進数	16 進数	
1	1	1	このビットがオンの場合、指定されたロボットは RSC を持ちます。
2	2	2	このビットが ON の場合、ロボットは長さが延長されたクイルを装備しています。
3	4	4	このビットが ON の場合、ロボットはクリーンルームオプションを装備しています。
4	8	8	今後の使用のための予備
5	16	10	このビットが ON の場合、ロボットはハイトルクオプショ

ビット番号	マスク値		説明
	10 進数	16 進数	
			ンを装備しています。
6	32	20	今後の使用のための予備
7	64	40	このビットが ON の場合、ロボットは EC 認証オプションを装備しています。
8	128	80	このビットがオンの場合、ロボットは高解像度ジョイント 4 を持ちます。
9	256	100	今後の使用のための予備
10	512	200	このビットがオンの場合、ロボットは橙色の LED を持ちます。 このオプションは eCobra ロボットモデルの場合に限って利用可能です。
11	今後の使用のための予備		
12			
13			
14			
15			
16			

関連キーワード

ID（モニタコマンド）

$ID（実数値関数）

SELECT（モニタコマンド）

SELECT（プログラムコマンド）

SELECT（実数値関数）

IDENTICAL

2 つのロケーションの値が同一かどうかを判定する実数値関数です。

構文

IDENTICAL (location1, location2)

パラメータ

location1 対象のロケーションの 1 つを定義する変換値です。関数、変数、または複合変換として使用できます。

location2 対象のロケーションの 1 つを定義する変換値です。関数、変数、または複合変換として使用できます。

詳細

この関数は、2 つの指定したロケーションの位置要素と回転要素が同一の場合、値 TRUE を返します。要素に 1 ビットでも違いがある場合は、値 FALSE を返します。

例

次の例では、フレーム「base.1」に関連する「loc」の値が変数「part」に保存される値と同一の場合、実数値変数 x の値を TRUE に設定します。

```
x = IDENTICAL(base.1:loc,part)
```

関連キーワード

DISTANCE

INRANGE

ロボットが指定されたロケーションに到達できるかどうかを示す値を返し、ロケーションに到達できない場合は追加の情報を提供する、実数値関数です。

構文

INRANGE (location)

使用上の留意事項

INRANGE 関数は、関数を実行するタスクが選択したロボットの情報を返します。

パラメータ

location	ロボットツール先端の目的の位置、方向を指定するオプションの変換関数、変数、複合変換です。このパラメータが省略されている場合、INRANGE は、選択されたロボットが現在のロケーションに到達できるかどうかを示します。

詳細

この関数は、ロボットが指定されたロケーションに到達できるかどうかを示す値を返します。戻り値 0 は、指定されたロケーションに到達できることを示します。

ロケーションに到達できない場合、戻り値は、追加情報を提供するコード化された 2 進数になります。次の表に示すように、ビットの値が 1 の場合、該当するロボットの制限に違反することを示します。

ビット番号	マスク値		ビットセットの意味
	16 進	10 進	
1	1	1	ジョイントまたはモーター 1 を制限
2	2	2	ジョイントまたはモーター 2 を制限
3	4	4	ジョイントまたはモーター 3 を制限
4	8	8	ジョイントまたはモーター 4 を制限
5	10	16	ジョイントまたはモーター 5 を制限
6	20	32	ジョイントまたはモーター 6 を制限
7	40	64	ジョイントまたはモーター 7 を制限
8	80	128	ジョイントまたはモーター 8 を制限

ビット番号	マスク値		ビットセットの意味
	16 進	10 進	
9	100	256	ジョイントまたはモーター 9 を制限
10	200	512	ジョイントまたはモーター 10 を制限
11	400	1024	ジョイントまたはモーター 11 を制限
12	800	2048	ジョイントまたはモーター 12 を制限
13	1000	4096	衝突検出
14	2000	8192	ロケーションが近すぎる
15	4000	16384	ロケーションが遠すぎる
16	8000	32768	ジョイントではなく、モーターを制限（以下参照）
17	10000	65536	方向が Quattro プラットフォームの範囲外
18	20000	131072	キネマティクスソリューションが見つかりません

モーションシステムがモータ制限とジョイント制限エラーを返すように設定されている場合、ビット 16 は、ジョイントまたはモーターがロケーションまでのモーションを制限されるかどうかを示します。ビット 16 が設定されている場合、すべてのジョイントは制限チェックをパスし、指示されたモーターは制限されています。それ以外の場合、指示されたジョイントは制限されています。

該当するビットが設定されているかを判断するために BAND キーワードで上記のマスク値を使用できます。

例

次の例では、ロボットが複合変換「pallet:hole」で定義されるロケーションに到達できる場合、値 0 を返します。ジョイント 2 と 3 がモーションを制限される場合、戻り値は 6 になります。

```
INRANGE(pallet:hole)
```

関連キーワード

SELECT（プログラムコマンド）

SELECT （実数値関数）

$INTB

16 ビットの整数の 2 進数表現を格納した 2 バイトの文字列を返す文字列関数です。

構文

$INTB (value)

パラメータ

value 値が 2 進数表現に変換される実数値式です。

詳細

実数値の整数部は 2 進数に変換され、下位 16 ビットは、2 つの 8 ビット文字として文字列に格納されます。最初に 9～16 ビット、次に 1～8 ビットが格納されます。

この関数は次と同等の意味になります。

```
$CHR(INT(value/256) BAND ^HFF) + $CHR(INT(value) BAND
^HFF)
```

この関数は、主に、整数をデータファイルの出力レコード内の 2 進数の表現に変換するために使用されます。

例

次の例では、文字列「AC」が返されます。

```
$INTB(65*256+67)
```

関連キーワード

$CHR

$DBLB

$FLTB

INTB

$LNGB

INT

値の整数部を返す実数値関数です。

構文

INT (value)

パラメータ

value	この関数がその整数部を返す実数式です。

詳細

値が指数表記で記述されていない場合、パラメータ値の小数点の左側部分を返します。

値の小数部分は、四捨五入されず切り捨てられます。

0 以外の場合、パラメータ値の符号は保持されます。

例

次の例では、0.0 を返します。

```
INT(0.123)
```

次の例では、10.0 を返します。

```
INT(10.8)
```

次の例では、-5.0 を返します。

```
INT(-5.462)
```

次の例では、131.0 を返します。

```
INT(1.3125E+2)
```

次の例では、「cost」の値を切り捨て、整数値を返します。

```
INT(cost)
```

次の例では、「cost」の値を四捨五入して、最も近い整数値を返します。「Cost」のマイナス値を正確に四捨五入するために、関数の sign が必要になります。

```
INT(cost+0.5*SIGN(cost))
```

関連キーワード

FRACT

INTB

2 バイトの文字列を符号付き16 ビット2 進整数として解釈した値を返す実数値関数です。

構文

INTB ($string, first_char**)**

パラメータ

$string	変換される 2 バイトを含む文字列式です。
first_char	2 バイトの文字列の開始位置を指定するオプションの実数式です。 first_char が省略、または値 が 0、1 の場合、文字列の先頭から 2 バイトが抽出されます。first_char が 1 より大きい場合、抽出するバイトの開始文字位置として解釈されます。例えば、値 2 は、2 バイト目の 9～16 ビットと 3 バイト目の 1～8 ビットを抽出します。first_char に入力文字列の末尾を超えたバイトのペアを指定した場合、エラーになります。

詳細

連続した2 バイトの文字列が、2 の補数の16 ビット符号付き2進整数として解釈されます。1 バイト目にはビット 9～16、2 バイト目にはビット 1～8 が含まれます。

この関数は、主に入力データレコードの 2 進数を eV+ が内部的に使用できる値に変換するために使用されます。

次の 2 つの式（INTB と ASC）は同じ機能になります。

```
value = INTB($string, first_char)
```

```
value = ASC($string,first_char)*256 + ASC($string,first_
char+1)
IF value > 32767 THEN
    value = value-65536
END
```

符号なし整数を計算するためには、以下のステートメントを使用します。

```
INTB($string) BAND ^HFFFF
```

例

次の例では、2565 の値を返します。

```
INTB($CHR(10)+$CHR(5))
```

次の例では、-1 の値を返します。

```
INTB($CHR(255)+$CHR(255))
```

関連キーワード

ASC

DBLB

FLTB

$INTB

LNGB

VAL

INVERSE

所定の変換値の逆関数となる変換値を返す変換関数です。

構文

INVERSE (transformation)

パラメータ

transformation　実数値式です。

詳細

この関数によって得られる値は数学的に、次に示す成分変換の値が恒等変換（またはヌル）になるような変換です。

```
INVERSE(trans):trans
```

オブジェクト A のロケーションを、オブジェクト B を基準として定義する変換 x を考えます。ここで INVERSE(x) はオブジェクト A を基準とするオブジェクト B のロケーションを定義する変換となります。

例

次の例では、「offset_tool1」および「offset_tool2」で表される 2 つの異なるエンドエフェクタが存在する場合にロボットを同じロケーションに移動します。

```
TOOL NULL
MOVE place_loc:INVERSE(offset_tool1)
MOVE place_loc:INVERSE(offset_tool2)
```

関連キーワード

HERE

SET

IOSTAT

論理ユニットに関連付けられたデバイスの最新の入出力命令に関するステータス情報を返す実数値関数です。

構文

IOSTAT (lun, mode)

使用上の留意事項

IOSTAT は、最新の命令に関する情報のみ返します。複数の命令が実行される場合、各命令の後に、ステータスを確認する必要があります。

> **重要:**各 I/O 命令の後に、IOSTAT を使用する必要があります。1つの I/O 命令が失敗した場合、以降の命令で想定外の動作をする可能性があります。例えば、ファイルのオープンに失敗した場合、書き込み命令を発行できない可能性があります。I/O 命令エラーは、プログラムの実行を停止しないため、REACTE で非同期に対処することはできません。

サイズが不明のファイルを読み取る場合、いつファイルの最後に到達したかを判別する方法は IOSTAT だけです。

パラメータ

lun 対象の I/O デバイスの論理ユニット番号を示す整数値の実数値式です。eV+ システムが認識する論理ユニット番号と論理ユニットがどのように I/O デバイスと関連するかについては、ATTACH を参照してください。

mode 指定した論理ユニットに返される I/O ステータスの種類を選択するオプションの式です。次の表に、さまざまなモード値の効果を示します。モード値が省略されている場合、値は 0 とみなされます。

モード	IOSTAT の戻り値
0	完了した最新の I/O 命令のステータス
1	保留中の先読みリクエストのステータス
2	最後に開いたファイル、または最後に読み取ったレコードのバイトサイズ シーケンシャルアクセスモードを使用している場合、IOSTAT(...,2)が返すバイト数には、各レコードの最後にあるキャリッジリターンとラインフィードが含まれます。
3	未完了の書き込みリクエストのステータス

詳細

ほとんどの eV+ キーワードとは異なり、I/O 関連キーワードは、エラーが検出されたとき、強制的にプログラムを停止しません。その代わりに、IOSTAT 関数でアクセスするために、エラーステータスを内部に保存します。プログラムは、この機能を利用して、多くの I/O エラーを解釈し、エラーから取得することができます。

モード 0、1、3 の場合に返される値を次の表に示します。

EOF で返される IOSTAT 値	説明
1	正常終了 - モード 3 では、この値は未完了の書込要求がないことを示します。
0	命令未完了
< 0	標準の eV+ のエラー番号 エラー番号の詳細については、eV+を参照してください。

例

次の例では、読み取り用にファイルを開き、ファイルが存在することを確認します。ファイルが存在する場合、変数「file.size」にファイルサイズをバイトで返します。ファイルが存在しない場合、メッセージ"Error opening file"を返します。

```
ATTACH (dlun, 4) "DISK"
FOPENR (dlun) "RECORD.DAT"
IF IOSTAT(dlun) < 0 THEN
    TYPE "Error opening file"
    HALT
END
file.size = IOSTAT(dlun,2)
```

次の例では、ファイルの最後に到達するまでレコードを読み取り、表示します。

```
ieeof = -504
READ (dlun) $record
WHILE IOSTAT(dlun) > 0 DO
    TYPE $record
    READ (dlun) $record
END
IF IOSTAT(dlun) == ieeof THEN
    TYPE "Normal end of file"
ELSE
    TYPE /B, "I/O error ", $ERROR(IOSTAT(dlun))
END
FCLOSE (dlun)
DETACH (dlun)
```

次の例では、TCP サーバープログラムの一部はがウェイトなしの読み取りを実行し、ステータスを確認して、クライアントが接続または切断されているかを判断します。

```
ATTACH (lun,4) "TCP"
IF IOSTAT (lun) < 0 THEN
    TYPE "Attach error: ", $ERROR(IOSTAT(lun))
```

```
END
no_wait = 1
READ (lun, handle, no_wait) $in.str
status = IOSTAT(lun)
CASE status OF
  VALUE 1:
    TYPE "Data received.  Handle =", handle
  VALUE 100:
    TYPE "New connection established.  Handle =", handle
  VALUE 101:
    TYPE "Connection closed.  Handle =", handle
  VALUE -526:
    WAIT
  ANY
    TYPE "Error during READ:  ", $ERROR(status)
    GOTO 100
END
```

関連キーワード

ATTACH

FCLOSE

FCMND

FEMPTY

FOPEN

FSEEK

READ

WRITE

LAST

配列（次元）で使用されているインデックスの最大値を返す実数値関数です。

構文

LAST (array_name[])

使用上の留意事項

オート変数を参照する場合、この関数は、どの要素に値が割り当てられているかにかかわらず、対象の配列を宣言する AUTO ステートメントで指定されたインデックス（要素番号の最大値）を返します。

パラメータ

array_name[]	テスト対象の配列名です。eV+ 配列変数のすべての種類 (実数値、位置、文字列、およびベルト) を指定できます。多次元配列の場合、少なくとも 1 つの配列の添え字を省略する必要があります。

詳細

この関数は、どの配列要素が定義されているかを判断するために使用できます。1 次元配列（part[ ]など）の場合、この関数は、要素が定義されている配列のインデックスの最大値を返します。

多次元配列（$names[,]など）の場合、この関数は、配列指定から省略されている左端の次元に定義されている要素の配列のインデックスの値の最大値を返します。

> **注:** 省略されているインデックスの右に指定されたインデックスを返すことはできません。

この関数の戻り値は、インデックスの値であり、配列要素ではありません。この値は、定義されている配列要素のカウント値ではありません。定義されている配列要素のインデックスの値の最大値です。

要求された次元に対して定義された要素が配列にない場合、-1 の値 が返されます。次の状況が発生した場合、-1 の値が返されます。

- 配列が存在しない。
- 配列の次元数が、関数呼び出しで指定した数より多い、または少ない。例えば、配列「a」が 2 次元の場合、LAST(a[ ])は -1 を返します。
- 多次元配列の指定の次元が定義されていない。例えば、LAST (a[,])が 19 を返す場合、要素 a[20,i]は存在しないので、LAST (a[20,])は -1 を返します。

この関数に指定された配列に 1つ 以上の空白のインデックスがない場合、または空白のインデックスの右側にインデックスが指定されている場合、*Illegal array index*エラーとなります。

例

配列 part[ ]の part[0]から part[10]まで、すべての要素が定義されている場合、次の例では、10 の値を返します（定義されている要素数の 11 ではなく）。

```
LAST(part[])
```

指定の 2 次元配列の要素[2,0]、[2,3]、[2,5]が定義されている場合、次の例では、2 以外の i の要素[i,j]の状態にかかわらず値 5 を返します。

```
LAST($names[2,])
```

関連キーワード

AUTO

LATCH

最新の外部トリガーが発生したときのロボットのロケーションを示す変換値を返す変換関数です。

構文

LATCH (select)

使用上の留意事項

LATCH(0)は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

パラメータ

select 次の示す値を指定するオプションの整数、式、実数値変数です。

- 0: 現在選択されているロボットのロボット位置ラッチです（デフォルト）。
- n: ロボット n のロボット位置ラッチです。

詳細

LATCH は、最後に外部トリガーが発生したときのロボットのロケーションを示す変換値を返します。

> **追加情報:** いつ外部トリガーが発生し、有効なロケーションが記録されたかを確認するために LATCHED 実数値関数を使用する必要があります。

外部トリガーのオペレーションは、*Sysmac Studio* ソフトウェアから設定することができます。

> **追加情報:** DEVICE関数を使用して外部エンコーダのラッチがかかる値を読み取ることができます。

関連キーワード

DEVICE

LATCHED

CLEAR.LATCHES

#PLATCH

LATCHED

位置ラッチの状態、トリガーした入力の状態を返す実数値関数です。

構文

LATCHED (select)

使用上の留意事項

1 台のベルトエンコーダに最大で 8 つの信号を関連付けることができます。

位置ラッチ信号範囲の設定は、Sysmac Studio ソフトウェアの eV+ システム設定エディタから設定することができます。詳細については、*『Sysmac Studio ロボット統合システム構築機能オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』*を参照してください。

パラメータ

select この関数が実行されてからラッチが発生したかどうかを判定する整数値、式または実数変数です。

- 0: 現在選択されているロボットのラッチ情報を返します。
- -n (< 0): ベルトエンコーダ n のラッチ情報を返します。
- +n (> 0): ロボット n のラッチ情報を返します。

詳細

LATCHED 関数が最後に使用された後に、ロボットのロケーションまたはベルトエンコーダの位置がラッチされている間に、位置ラッチが発生した場合、この関数は 0 以外の値を返します。次の値が返されます。ここで n はコントローラ上の任意のデジタル入力信号を示します。

- 0: ラッチイベントは検出されていません。
- n: 立ち上がりエッジが検出されました。ここで n はコントローラ上でのデジタル入力信号を表します。
- -n: 立ち下がりエッジが検出されました。ここで n はコントローラ上でのデジタル入力信号を表します。

この関数が 0 以外の値を返す場合、次のキーワードでラッチイベントのデータを検索できます。

- DEVICE - 外部エンコーダに関する位置情報を返します。
- BELT - 外部エンコーダに関する位置情報を返します。
- LATCH - ロボットのロケーション情報を変換値として返します。
- #PLATCH - ロボットのロケーション情報をプレシジョンポイントの値として返します。

注: この関数が 1 つまたは複数の 0 以外の値を返した後、ラッチバッファは空になり、それ以降に関数を使用した場合、次のラッチトリガーが発生するまで値 FALSE を返します。

関連キーワード

DEVICE

BELT

ENCLATCH

LATCH

CLEAR.LATCHES

#PLATCH

LEN

指定された文字列の文字数を返す実数値関数です。

シンタックス

LEN (string)

パラメータ

string	長さが算出される文字列定数、変数、式です。

例

次の例では、文字列「$str」の文字数の値 5 を「str.len」に返します。

```
$str = "Hello」
str.len = LEN($str)
```

$LNGB

32 ビットの整数の2 進数表現を格納した4 バイトの文字列を返す文字列関数です。

構文

$LNGB (value)

使用上の留意事項

実数値は四捨五入され、小数部は削除されます。値は、^H7FFFFFFF～-^H80000000 の範囲内にある必要があります。

パラメータ

value	2 進数に変換される値は、実数値、変数、または式です。

詳細

この関数は、主に整数値をデータファイルの出力レコードの2進数に変換するために使用されます。

実数値の整数部は、2 進数に変換されます。下位の 32 ビットは、4 つの 8 ビット文字として文字列に格納されます。25～32 ビットが最初のバイト、17～24 ビットが 2 番目のバイトの順で格納されます。

この関数が実行する演算は、次のステートメントと同等の意味になります。

```
$CHR(INT(value/^H1000000) BAND ^HFF)+ $CHR(INT(value/^H10000) BAND
^HFF)+ $CHR(INT(value/^H100) BAND ^HFF)+ $CHR(INT(value) BAND ^HFF)
```

例

次の例では、$INTB(67) と $INTB(12345) の合計値を返します。

```
$LNGB(67*65536+12345)
```

関連キーワード

$CHR

$FLTB

$INTB

LNGB

TRANSB

LNGB

4 バイトの文字列を符号付き32 ビット2 進整数として解釈した値を返す実数値関数です。

構文

LNGB ($string, first_char)

使用上の留意事項

単精度の数値は、有効数字が 24 ビットで内部に格納されるため、24 ビットを超える有効数字を含む入力値は、精度が低くなります。

倍精度の数値は、最上位ビットを符号ビットとし、有効数字が 32 ビットで保存されるため、精度を損なうことなく変換されます。

この関数は、主に入力データレコードの 2 進数を eV+ で内部的に使用できる値に変換するために使用されます。

パラメータ

$string	変換される4 バイトを含む文字列定数、変数、式です。
first_char	4 バイトの文字列の開始位置を指定する整数として解釈されるオプションの実数値、変数、式です。first_char に入力文字列の末尾を超えた連続する 4 バイトを指定した場合、エラーになります。 first_char が省略、または値が 0、1 の場合、文字列の先頭から 4 バイトが抽出されます。first_char が 1 より大きい場合、抽出するバイトの開始文字位置として解釈されます (以下参照)。

詳細

4 バイトの連続した 4バイト の文字列は、2 の補数の 32 ビット符号付き 2 進整数として解釈されます。4 バイトの1バイト目には整数の 25～32 ビット、2 バイト目には 17～24 ビットの順番でデータが含まれます。

例えば、first_char の値が 9 で、入力文字列の 9 番目の文字（バイト）に整数の 25～32 ビットが含まれる場合、文字列の 10 番目のバイトは、17～24 ビットになります。

例

次の例では、値 65541 を返します。

```
LNGB($INTB(1)+$INTB(5))
```

関連キーワード

ASC

DBLB

FLTB

INTB

$LNGB

TRANSB

VAL

MAX

値のリストにある最大値を返す実数値関数です。

構文

MAX (value, ..., value**)**

パラメータ

value	リストの各値を実定数、変数、式として指定できます。

詳細

この関数は、リストの値の中から最大値を検索し、その値を返します。

各値の符号も考慮されます。例えば、値 -10は、-100 より大きいとみなされます。

例

次の例では、変数 x、y、z の最大値を「max.value」を設定します。3 つの変数すべてが 0 未満の場合は、0 を設定します。

```
max.value = MAX(x, y, z, 0)
```

関連キーワード

MIN

OUTSIDE

$MID

指定文字列のサブストリングを返す文字列関数です。

構文

$MID (string, first_char**, num_chars)**

パラメータ

string	サブストリングから文字列変数、定数、または式を抽出します。
first_char	サブストリングの最初の文字を指定するオプションの実数値式です。
num_chars	サブストリングにコピーする文字数を指定する実数値式です。

詳細

first_char を省略、または 1 以下の値の場合、サブストリングは、文字列の最初の文字で始まります。first_char が入力文字列の長さより大きい場合、空の文字列を返します。

指定された開始文字位置から入力文字列の最後までの文字数が num_chars より少ない場合、出力文字列は、入力文字列の最後までの文字のみで構成されます。エラーが発生しても結果を返さず、出力文字列は要求された長さまで拡張されません。

例

次の例では、文字列変数「$substring」に文字列 cd が格納されます。cd は、文字列変数「$string」に含まれる文字列 abcde の 3 文字目の位置から始まる 2 文字です。

```
$string = "abcdef"
$substring = $MID($string, 3, 2)
```

関連キーワード

$DECODE

$UNPACK

MIN

値のリストにある最小値を返す実数値関数です。

構文

MIN (value, ..., value)

パラメータ

value	リストの各値を実定数、変数、式として指定できます。

詳細

この関数は、リストの値の中から最小値を検索し、その値を返します。

各値の符号も考慮されます。例えば、値 -100 は、-10 より小さいとみなされます。

例

次の例では、変数 x、y、z の最小値を「min.value」を設定します。3 つの変数すべてが 1,000 以上の場合は、値 1,000 を設定します。

```
min.value = MIN(1000, x, y, z)
```

関連キーワード

MAX

OUTSIDE

NETWORK

ネットワークの状態とロボットコントローラの IP アドレス情報を返す実数値関数です。

構文

NETWORK (component, code**)**

パラメータ

component 対象とするネットワークのコンポーネントを識別する実数値式です。

- 1 = TCP
- 3 = 今後の使用のための予備

code IP アドレスとネットワークマスクのどのバイトを得たいのかを指定するためのオプションの実数値式です。

以下に有効な値を示します。ADn は IP アドレスの n 番目のバイト、NMn はネットワークマスクの n 番目のバイトを示しています。

この値は component が 1 のときのみ使用されます。

0 =「詳細」に示す状態値を返します(デフォルト)
1 = AD1*256 + AD2 を返します
2 = AD3*256 + AD4 を返します
3 = NM1*256 + NM2 を返します
4 = NM3*256 + NM4 を返します

11 = AD1 を返します
12 = AD2 を返します
13 = AD3 を返します
14 = AD4 を返します
15 = NM1 を返します
16 = NM2 を返します
17 = NM3 を返します
18 = NM4 を返します

詳細

この関数は、状態が要求されている場合（code = 0）、次の値のいずれかを返します。

値	説明
0	ハードウェアなし
-1	ハードウェアあり
1	ドライバー動作中

例

次の例では、モニタウィンドウにロボットコントローラの IP アドレスの最後のオクテットを表示します。

```
ip[1]=NETWORK(1,14)

$last.octet=$ENCODE(ip[1])

TYPE $last.octet
```

NORMAL

変換値の切り捨て誤差を補正する変換関数です。

構文

NORMAL (transformation_value)

パラメータ

transformation_value	正規化される値を持つ変換、変換値関数、複合変換です。

詳細

NORMAL 関数は、入力された引数とほぼ同じですが、方向に関する部分についてわずかな計算誤差の蓄積分を修正した変換値を返します。

変換値を変化させる長い計算の後に、この関数を使用します。たとえば、変換の方向を徐々に変化させる手続き的動作では、計算結果を正規化する必要がある場合があります。変換内では、ロボットの方向は 3 つの直交単位ベクトルで表されます。何度も徐々に変化させると、コンピューターの計算にわずかな誤差が生じるため、これらのベクトルは直交ではない、または単位長ではない可能性があります。

例

次の例では、現在のロボットの位置を正規化した変換を求め、変数「normal.loc」に値を代入します。

```
SET normal.loc=NORMAL(HERE)
```

NOT

値の論理否定を実行する演算子です。

構文

...NOT value ...

詳細

NOT 演算子は、1 つの値に対して論理的に真（TRUE）を偽（FALSE）、またその逆の操作をします。値が 0 の場合は、この演算子により -1(TRUE)を返します。値が 0 以外の場合はこの演算子により 0(FALSE)を返します。

例

次の例では、変数「initialized」が偽の値の場合、IF 構文のキーワードが実行されます。

```
IF NOT initialized THEN
    CALL appl.setup()
    initialized = TRUE
END
```

次の例では、値 40 は 0 ではないため、論理的に TRUE として解釈されます。次のステートメントは、0（FALSE）の値を返します。

```
NOT 40
```

関連キーワード

COM

NULL

null 変換値（すべての成分が 0 の値）を返す変換関数です。

構文

NULL

使用上の留意事項

エミュレーションモード中のロボットの軌道は、現在の目標位置と同じとみなされます。そのため、実際のロボットの場合とは異なり、命令が完了したときにポジショニングのエラーは発生しません。

詳細

null 変換は、null ベクター（X = Y = Z = 0）と回転なし（yaw = pitch = roll = 0）に対応します。null 変換は、たとえば SHIFT 関数とともに使用すると、回転のない平行移動を表す変換を作成するような場合に役立ちます。

例

次の例では、既存の座標変換「old.loc」をワールド座標方向にシフトした結果として新しい座標変換「new.loc」を定義します。

```
new.loc = SHIFT(NULL BY x.shift,y.shift,z.shift):old.loc
```

次の例では、変換「test.loc」によって示されるベクトルの長さを決定します。

```
dist = DISTANCE(NULL, test.loc)
```

関連キーワード

CONFIG

NULL

SHIFT

OFF

eV+ で使用される論理偽の結果を示す値を返す実数値関数です。

構文

OFF

詳細

この名前付き定数は、ON と OFF の条件指定が必要な状況で有用です。戻り値は 0 です。

この関数は FALSE 関数と同等です。

例

次の例では、システムスイッチ POWER が OFF になるまでDO～UNTIL キーワード間のステートメントを実行します。

```
DO
    ;statement
    ;statement
    ;statement
UNTIL SWITCH(POWER) = = OFF
```

関連キーワード

FALSE

ON

ON

eV+ で使用される論理真の結果を示す値を返す実数値関数です。

構文

ON

詳細

この名前付き定数は、ON と OFF の条件指定が必要な状況で有用です。戻り値は -1 です。

この関数は TRUE 関数と同等です。

例

次の例では、システムスイッチ POWER が有効な場合、モニタウィンドウに"System switch POWER is ON"を表示します。

```
IF(SWITCH(POWER)==ON) THEN
    TYPE "System switch POWER is ON"
END
```

関連キーワード

OFF

TRUE

OUTSIDE

指定範囲以外かを判断するために、値をテストする実数値関数です。

構文

OUTSIDE (low, test, high)

パラメータ

low テスト対象の範囲の下限値を指定する実数値、式、変数です。

test 範囲をテストする実数値、式、変数です。

high テスト対象の範囲の上限値を指定する実数値、式、変数です。

詳細

テスト結果が下限値未満、または上限値より大きい場合、TRUE (-1)を返します。それ以外の場合、FALSE (0)を返します。

例

次の例では、「angle」の値を評価して、その値が 0～180 以外の場合、"Angle is out of reach." を表示します。

```
IF OUTSIDE(0,angle,180) THEN
    TYPE "Angle is out of reach."
END
```

関連キーワード

MAX

MIN

PARAMETER

指定したシステムパラメータの現在の設定を返す実数値関数です。

構文

PARAMETER (parameter_name)

PARAMETER (parameter_name[index])

パラメータ

parameter_name	値を返すシステムパラメータ名です。
index	index があるパラメーターの場合、指定したパラメータの目的の要素を指定するために必要な実数値、変数、または式です。

詳細

この関数は、指定されたシステムパラメータの現在の設定を返します。

> **追加情報:** パラメータ名には、一意に識別できる最小長の短縮名を付けることができます。

例

次の例では、プログラムの実行中に BELT.MODE システムパラメータの現在の設定をモニタウィンドウに表示する方法を示しています。

```
TYPE "The BELT.MODE parameter is set to", PARAMETER
(BELT.MODE)
```

関連キーワード

BELT.MODE

NOT.CALIBRATED

PARAMETER

PARAMETER （プログラムコマンド）

#PDEST

現在のロボットモーションに対して、計画された目的地を示すプレシジョンポイント値を返す精度ポイント関数です。

構文

#PDEST

使用上の留意事項

#PDEST 関数は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

詳細

ロボットのモーションが中断される前に、ロボットの目標位置の判断に #PDEST 関数を使用できます。

#PDEST 関数は DEST 変換関数と同等で、目的の位置情報の種類に応じて DEST と同様に使用できます。#PDEST、DEST 関数の使用に関する詳細な情報については、DEST 関数の説明を参照してください。

例

次の例では、目標位置「point1」でのジョイント値を表示します

```
MOVE point1
DECOMPOSE j[]=#PDEST
TYPE j[0],j[1],j[2],j[3],j[4],j[5]
```

関連キーワード

DEST

HERE

SELECT

SELECT（実数値関数）

#PHERE

選択されているロボットの現在のロケーションを示すプレシジョンポイント値を返す精度ポイント関数です。

構文

#PHERE

使用上の留意事項

#PHERE 関数は、プレシジョンポイントキーワードとみなされるため、使用する場合は、関数名の前に「#」を付ける必要があります。

PHERE は eV+ の予約語です。変数には使用できません。

詳細

PHERE 実数値関数は、HEREと同等です。

例

次の例では、#PHERE を使用してプレシジョンポイント値「pp」を設定しています。

```
SET #pp = #PHERE
```

関連キーワード

HERE

PI

円周率（3.141593）の値を返す実数値関数です。

構文

PI

使用上の留意事項

TYPEしかし、pi は倍精度値として保存、処理されます。LISTRモニタコマンドは、実数値を最大限の精度まで表示します。

例

次の例では、「arc_length」と「radius」の値を使用して半円の円周を返します。

```
arc_length=PI*radius
```

#PLATCH

システムに含まれているエンコーダの、最新のラッチが発生した時点でのエンコーダ値を返すプレシジョンポイント関数です。

構文

#PLATCH (select)

使用上の留意事項

設定されていないエンコーダをselect パラメータで指定してENCLATCH キーワードを実行した場合、エラーが返されます。

#PLATCH(0)は、関数を実行するタスクが選択したロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

パラメータ

select	次に示す値を指定するオプションの整数、式、実数値変数です。 • 0: 現在選択されているロボットのロボットの位置ラッチ(デフォルト) • n: ロボットn のロボット位置ラッチ

詳細

#PLATCH は、最新のトリガーが発生したときのロボットのロケーションを示すプレシジョンポイント値を返します。外部トリガーが発生し有効な位置が記録されているかを判定するために、LATCHEDキーワードを使用する必要があります。

エンコーダの他の情報を読み出すのど同様に、DEVICEキーワードをエンコーダのラッチされた値を読み出すためにも使用できます。外部トリガーの作用は、Sysmac Studio で設定できます。

関連キーワード

LATCH

LATCHED

POS

文字列内のサブストリングの開始文字位置を返す実数値関数です。

構文

POS (search_string, sub_string, start)

パラメータ

search_string	サブストリングを検索するための文字列式です。
sub_string	文字列内で検索されるサブストリングを格納した文字列式です。
start	対象の文字列中の検索開始文字位置を示すオプションの式です。

詳細

search_string 内で sub_string が始まる文字位置を返します。検索文字列内にサブストリングがない場合、値 0 を返します。

start が指定されている場合、search_string 内の検索開始文字位置を示します（値1は、先頭文字）。start が省略、または 1 未満の場合、先頭文字から検索を開始します。start が search_string の長さより大きい場合、値 0 を返します。

一致するサブストリングを照合するとき、大文字、小文字の区別はされません。

例

次の例では、5 を返します。

```
POS("file.ext", ".")
```

次の例では、0 を返します。

```
POS("file", ".")
```

次の例では、4 を返します。

```
POS("abcdefgh", "DE")
```

次の例では、6 を返します。

```
POS("1-2-3-4", "-", 5)
```

#PPOINT

指定の構成要素から成るプレシジョンポイント値を返すプレシジョンポイント関数です。

構文

#PPOINT (j1_value, j2_value, j3_value, j4_value, j5_value, j6_value, j7_value, j8_value, j9_value, j10_value, j11_value, j12_value)

使用上の留意事項

#PPOINT 関数名は、プレシジョンポイント名とみなされるため、この関数を使用する場合は、必ず関数名の前に「#」を付ける必要があります。

ロボットのジョイント数以上の値を指定した場合、超過した値は無視されます。

すべてのパラメータが省略された場合は 0 となります。

パラメータ

j1_value　　ロボットの各ジョイント位置を示すオプションの実数値式です。

j2_value

j3_value

...

j12_value

詳細

ロボットの最初のジョイントから最後のジョイントまでの指定された各構成要素から成るプレシジョンポイント値を返します。

例

次の例では、現在のロケーションから 4 軸ロボットのジョイント 2 と 3 の協調動作を開始するとします。

次のステートメントは、現在のロケーションを定義し、配列にその構成要素を入れます。

```
HERE #ref
DECOMPOSE x[] = #ref
```

次のステートメントは、変更された構成要素で定義された新しいプレシジョンポイントにロボットを移動します。

```
MOVE #PPOINT(x[0], x[1]+a, x[2]-a/2, x[3])
```

次のステートメントは、同じ最終のロケーションに移動させますが、ロボットのジョイント 2 と 3 は、この方法では同時に動きません。

```
DRIVE 2, a, 100
DRIVE 3, -a/2, 100
```

関連キーワード

DECOMPOSE

TRANS

PRIORITY

プログラムに対する現在の反応ロックアウトの優先度を返す実数値関数です。

構文

PRIORITY

使用上の留意事項

この関数は、プログラム実行の優先度ではなく、反応ロックアウトの優先順位を返します。

詳細

PRIORITY 関数は、その関数を実行しているタスクで現在設定されている反応ロックアウトの優先度を確認するために使用できます。

タスクの実行が初期化されると、各プログラムタスクの反応ロックアウトの優先度は、0 に設定されます。優先度は、LOCKコマンドでいつでもプログラムで変更できます。また、REACTまたはREACTIコマンドの記述に従って応答が発生したとき、優先度は自動的に設定されます。

例

次の例では、優先度を上げ、ロックアウトされる応答ルーチンを必要とする命令を実行し、その後、優先度を前回の値に戻しています。

```
save = PRIORITY

IF save < 10 THEN
    LOCK 10
END

LOCK save
```

関連キーワード

LOCK

REACT

REACTI

RANDOM

疑似乱数を返す実数値関数です。

構文

RANDOM

詳細

0.0～1.0 の範囲内の疑似乱数を返します。RANDOM 関数は、評価されるたびに異なる値を返します。

この関数は、 2^{24}（16,777,216）回コールされた後、シーケンスを繰り返すため、この関数が生成する数は疑似乱数となります。

例

次の例では、0.0～1.0 の範囲内のランダム数を変数「random1」に返します。

```
random1 = RANDOM
```

RX

X 軸に対する回転を示す変換を返す変換関数です。

構文

RX (angle)

パラメータ

angle	回転角度を示す実数式です。

詳細

これらの関数は、X 軸に対する回転と変位 0（X、Y、Z = 0）から成る変換値を生成します。

例

次の例では、「frame1」をX 軸を中心に45 度回転したものと同等の変換「frame2」を作成します。

```
SET frame1 = TRANS(100,100,-960,0,180,0)
SET frame2 = frame1:RX(45)
```

関連キーワード

DX

DY

DZ

RY

RZ

RY

Y 軸に対する回転を示す変換を返す変換関数です。

構文

RY (angle)

パラメータ

angle	回転角度を示す実数式です。

詳細

これらの関数は、Y 軸に対する回転と変位 0（X、Y、Z = 0）から成る変換値を生成します。

例

次の例では、「frame1」をY 軸を中心に45 度回転したものと同等の変換「frame2」を作成します。

```
SET frame1 = TRANS(100,100,-960,0,180,0)
SET frame2 = frame1:RY(45)
```

関連キーワード

DX

DY

DZ

RX

RZ

RZ

Z 軸に対する回転を示す変換を返す変換関数です。

構文

RZ (angle)

パラメータ

angle	回転角度を示す実数式です。

詳細

これらの関数は、Z 軸に対する回転と変位 0（X、Y、Z = 0）から成る変換値を生成します。

例

次の例では、「frame1」をZ 軸を中心に30 度回転したものと同等の変換「frame2」を作成します。

```
SET frame1 = TRANS(0,0,0,0,0,30)
SET frame2 = frame1:RZ(30)
```

関連キーワード

DX

DY

DZ

RX

RY

SCALE

変換パラメータの位置を倍率に従って拡大縮小した変換値を返す変換関数です。

構文

SCALE (transformation BY scale_factor)

パラメータ

transformation 拡大縮小される変換式です。

scale_factor 変換パラメータ値を拡大縮小するために使用される実数式です。

詳細

X、Y、Z 位置の構成要素が倍率パラメータで拡大されることを除き、入力変換パラメータ値と等しい値を返します。回転構成要素は、変更されません。

例

次の例では、元の変換xの値が（200, 150, 100, 10, 20, 30）の場合、変換yは値（250, 187.5, 125, 10, 20, 30）になります。

```
SET y = SCALE(x BY 1.25)
```

関連キーワード

SHIFT

SELECT

現在選択中のタスクに指定されたデバイスに関する情報を返す実数値関数です。.

構文

SELECT (device_type, mode)

パラメータ

device_type 選択するデバイスのタイプを特定するパラメータです。唯一有効な device_type は、ROBOT です。

mode 関数の device_type 番号を指定するオプションの実数値、変数、式です。整数として解釈されます。

このパラメータが省略、または値が 0 の場合、この関数は、現在選択されている device_type の番号、または device_type が選択されていない場合は 0 を返します。

> **注:** 選択されていたロボットが EtherCAT ネットワークから接続を解除された場合、EtherCAT 接続が回復してもステートメント SELECT(ROBOT) は 2 を返します。選択されていたロボットは自動的に更新されません。

mode が値 -1 の場合、この関数は EtherCAT ネットワーク上で使用可能で通信できるユニットの総数を返します。

詳細

この関数は、現在選択されている指定デバイスの番号、またはシステムコントローラーに接続されているデバイスの総数のいずれかを返します。これはまた、指定したロボットの状態を取得するためにも使用できます。

eV+ システムがロボットを制御するように設定されていない場合、常に選択されたロボットは 1、ロボットの総数は 0 になります。

SELECT(ROBOT)は、現在選択されているロボットの番号を返します。

SELECT(ROBOT,-1) は EtherCAT と通信しているロボットの番号を返します。

SELECT(ROBOT, rob_num) （ここで「rob_num」は指定されたロボットを示す）はロボットの状態を示す次の値の 1 つを返します。

- 0: 未設定 - NJ シリーズNJ シリーズロボット統合CPU ユニット設定にロボットが含まれていません。
- 1: 非同期 - ロボットの EtherCat 設定と NJ シリーズロボット統合CPU ユニット が一致しません。通常起動でロボットが接続しませんでした。
- 2: 未接続 - ロボットが EtherCat ネットワークに接続していません。
- 3: 準備完了 - ロボットは設定、同期、接続されています。

追加情報: 関連するエラー、-622、-508 および -315 については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

次の例では、現在にタスクで選択されているロボットのユニット番号を変数「our.robot」に返します。

```
our.robot = SELECT(ROBOT)
```

次の例では、コントローラーに接続されているロボットの総数を変数「num.robot」に返します。

```
num.robots = SELECT(ROBOT,-1)
```

関連キーワード

SELECT（モニタコマンド）

SELECT（プログラムコマンド）

#SET.POINT

最後の軌道評価サイクルで軌道ジェネレーターによって算出されたジョイント角の指令位置を返すプレシジョンポイント関数です。

構文

#SET.POINT

使用上の留意事項

「set.point」は、プログラム名として使用できません。

詳細

ジョイント角の位置は軌道評価サイクルごとに計算され、エンコーダのカウント値に変換された後、モーターの指令位置としてサーボに送られます。この関数を使用してこれらの位置をキャプチャすることができます。

例

次の例では、100 ms ごとにロボットジョイントの値を配列「set.points」に保存します。

```
MOVE point2
i = 0
WHILE STATE(2) == 1 DO
    DECOMPOSE angles[i,] = #SET.POINT
    i = i+1
    WAIT.EVENT , 0.1
END
```

SHIFT

指定のシフト量で変換パラメータの位置をシフトした変換値を返す変換関数です。

構文

SHIFT (transformation BY x_shift, y_shift, z_shift**)**

パラメータ

transformation	シフトする変換式です。
x_shift	変換パラメータの各位置の構成要素に加算されるオプションの実数値式です。
y_shift	
z_shift	

詳細

3 つのシフトパラメーター値がX、Y、およびZ の位置成分に加算されることを除き、入力変換パラメーター値と同じ値を返します。シフトパラメータが省略されている場合、省略された値は 0 とみなされます。

例

次の例では、元の変換 x の値が（200, 150, 100, 10, 20, 30）の場合、変換 y の値は（205, 145, 110, 10, 20, 30）になります。

```
SET y = SHIFT(x BY 5,-5,10)
```

関連キーワード

SCALE

TRANS

SIG.INS

デジタル I/O 信号がシステムに装備されているか、またはソフトウェア信号がシステムで使用できるかを示す値を返す実数値関数です。

構文

SIG.INS (signal_num)

使用上の留意事項

SIG.INS 関数キーワードは、ホスト信号を使用できる場合 TRUE、使用できない場合（マップされていない）FALSE を返します。

SIG.INS 関数を使用して、設定されていないか物理的に存在しない IOBlox ユニットに関連するデジタル I/O の状態を判定すると、次の「詳細」項で説明するようにエラーが発生します。

パラメータ

signal_num	確認対象のデジタル I/O 信号またはソフトウェア信号の番号を定義する実数式です。絶対値を使用しますが、信号番号に負の値も使用できます。

詳細

この関数は、指定のデジタル I/O 信号またはソフトウェア信号がシステムで使用できる場合、TRUE (-1)を返します。それ以外の場合は、FALSE (0.0)を返します。signal_number が 0 の場合は、常にTRUEを返します。

アプリケーションプログラムの期待通りにデジタル I/O 信号が装備されていることを確認するために、この関数を使用できます。

この関数を設定されていないIOBlox 信号に対して実行すると、-508「Device not ready」（デバイス準備未完了）エラーが発生します。

この関数を物理的に存在しない IOBlox 信号に対して実行すると、-405「Illegal digital signal」（不正なデジタル信号）エラーが発生します。

> **注:** エミュレータを使用して、この関数を設定されていない IOBlox 信号に対して実行すると、-405「Illegal digital signal」（不正なデジタル信号）エラーが発生します。

例

次の例では、デジタル I/O 信号 12 が入力信号として装備されているかどうかを確認します（信号 1012 として参照）。信号が正しく設定されていない場合、モニタウィンドウにメッセージが表示されます。

```
in.sig = 1012
IF NOT SIG.INS(in.sig) THEN
    TYPE "Digital I/O signal ", in.sig, "is not
installed"
END
```

関連キーワード

BITS（モニタコマンド）

BITS（プログラムコマンド）

BITS(実数値関数)

IO

RESET

RUNSIG

SIGNAL

SIGNAL（プログラムコマンド）

SIGN

パラメータ「value」の符号を付けて、値「1」を返す実数値関数です。

構文

SIGN (value)

パラメータ

value	実数値式です。

詳細

この関数は、パラメータの値が 0 未満の場合、-1.0 を返します。パラメータの値が 0 以上の場合、+1.0 を返します。

例

次の例では、1.0 を返します。

```
SIGN(0)
```

次の例では、1.0 を返します。

```
SIGN(0.123)
```

次の例では、-1.0 を返します。

```
SIGN(-5.462)
```

次の例では、1.0 を返します。

```
SIGN(1.3125E+2)
```

SIG

指定されたデジタル信号の状態の論理積（AND）を返す実数値関数です。

構文

SIG (signal_num, ..., signal_num)

パラメータ

signal_num	評価するデジタル I/O または内部信号の番号を示す実数値式です。マイナス値は、対象の信号の負論理を示します。

詳細

指定されたデジタル信号すべての状態に対して論理積（AND）を実行することによって得られた値 TRUE（-1）または FALSE（0）を返します。指定したすべての信号状態が TRUE の場合、SIG は TRUE を返します。それ以外の場合、SIG は FALSE を返します。

特定のロボットの信号番号に関する詳細情報については、該当するロボットのユーザーズマニュアル、および *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

システムに存在するデジタル信号のみを使用できます。現在のデジタル I/O 設定を確認するには、IOモニタコマンドや SIG.INS関数を使用できます。信号 3001 と 3002 は、現在のタスクで選択されているロボットを参照します。信号 3001 は、ハンドクローズソレノイドの状態です。信号 3002 は、ハンドオープンソレノイドの状態です。

signal_num パラメータの符号が正の場合で、信号が H のとき、信号は TRUE となります。**signal_num** パラメータの符号が負の場合で、信号が L のとき、信号は TRUE となります。

> **注:** SIG(0)は、値 TRUE を返します。

例

次の例では、以下のデジタル I/O 信号が装備され、各値が以下の通りであるとします。

- 入力信号 1001 が On
- 入力信号 1004 が Off
- 入力信号 33 が Off

次の例では、-1.0 (TRUE)を返します。

```
SIG(1001)
```

次の例では、0.0 (FALSE)を返します。

```
SIG(1004)
```

次の例では、-1.0 (TRUE)を返します。

```
SIG(-1004)
```

次の例では、0.0 (FALSE)を返します。

```
SIG(1001,1004)
```

次の例では、-1.0 (TRUE)を返します。

```
SIG(1001,-1004)
```

関連キーワード

BITS

BITS

BITS (実数値関数)

IO

RESET

RUNSIG

SIGNAL (プログラムコマンド)

SIGNAL(モニタコマンド)

SIN

指定した角度の三角関数のサイン値を返す実数値関数です。

構文

SIN (value)

使用上の留意事項

角度パラメータの単位は度である必要があります。

パラメータは、360 度の剰余として解釈されますが、過度に大きな値を指定した場合、戻り値の精度が低くなる可能性があります。

パラメータ

value	角度の値を定義する実数値式です。

詳細

引数（度単位）のサイン値を返します。結果の値は、必ず -1.0～+1.0 の範囲内となります。

例

次の例では、2.146753E-03 を返します。

```
SIN(0.123)
```

次の例では、-0.09518556 を返します。

```
SIN(-5.462)
```

次の例では、0.5 を返します。

```
SIN(30)
```

> **注:** TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTR モニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

ASIN

COS

ACOS

TAN

ATAN2

SOLVE.FLAGS

ジョイント位置の配列で指定されたロボット設定を示すビットフラグを返す実数値関数です。

構文

SOLVE.FLAGS (joints[index])

使用上の留意事項

SOLVE.FLAGS 関数は、その関数を実行するタスクが選択しているロボットの情報を返します。

eV+ システムがロボットを制御するように設定されていない場合に SOLVE.FLAGS 関数を使用するとエラーになります。

パラメータ

joints	ロボットのジョイントの位置を格納した実数配列です。配列の 1 番目の要素にはジョイント 1 の位置、2 番目の要素にはジョイント 2 の位置のように、各要素に対応するジョイントの位置を格納する必要があります。回転ジョイントの場合は、ジョイント位置の単位は度であるとみなされます。変換ジョイントの場合は、単位は mm であるとみなされます。
index	ジョイント 1 の位置を格納する配列要素を識別する整数値として解釈されるオプションの実数値、変数、式です。インデックスが指定されていない場合、要素 0 にはジョイント 1 の位置を格納する必要があります。

詳細

この関数は、指定のロボットの位置セットに対するロボットの設定（右利き、左利きなど）を示すビットフラグを返します。この関数は、SOLVE.ANGLESプログラムコマンドに必要な設定データを提供する場合に役立ちます。

この関数の戻り値のビットは、次のように解釈されます。

ビット 1（LSB） RIGHTY（マスク値 = 1）

このビットがセットされている場合、ロボットは右腕設定です。それ以外の場合は、左腕設定です。

ビット 2 BELOW（マスク値 = 2）

このビットがセットされている場合、ロボットの肘の位置は肩から手首までのラインより下に設定されています。それ以外の場合は、ロボットの肘の位置は肩から手首までのラインより上に設定されています。

> **注:** SCARA ロボットを使用中の場合、このビットは常に 0 になります。

ビット 3 FLIP（マスク値 = 4）

このビットがセットされている場合、ロボットの手首のピッチ角はマイナスの角度に設定されています。それ以外の場合は、手首のピッチ角は正の値に設定されています。

注: 4 軸 SCARA ロボットのように、ロボットの手首が 3 軸方向に対応していない場合、このビットは常に 0 です。

関連キーワード

ABOVE

BELOW

DECOMPOSE

LEFTY

RIGHTY

FLIP

NOFLIP

SELECT

SELECT

SOLVE.ANGLES

SOLVE.TRANS

SPEED

システムのモーションの速度係数の一つを返す実数値関数。

構文

SPEED (select)

使用上の留意事項

SPEED 関数は、その関数を実行するタスクが選択しているロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。

パラメータ

select どの速度係数の値を返すかを決める値の実数値式です。

詳細

この関数は、選択したパラメータ値に対応するシステムのモーション速度係数を返します。許容可能なパラメータ値と対応する速度値の戻り値を次に示します。

Select	返される速度値
1	モニタ速度
2	定常的なプログラム速度（SPEEDプログラムコマンドで設定）。
3	前回または現在のモーションの一時的なプログラム速度
4	次のモーションに使用される一時的なプログラム速度
5	定常的なプログラム回転速度
6	前回または現在の直線モーションの一時的なプログラム回転速度
7	次の直線モーションに使用される一時的なプログラム回転速度
8	プログラム速度で設定可能な最大値

> **注:** 戻り値は、速度設定を指定したSPEEDプログラムコマンドでプログラム速度が設定されていても、通常速度の割合（%）として解釈する必要があります。

例

次の例では、1 つのモーションを定常的なプログラム速度の 1/2 で実行します。

```
new.speed = SPEED(2)/2
SPEED new.speed
MOVE pick.up
```

次の例では、上記例と同じ動作をします。

```
SPEED SPEED(2)/2
MOVE pick.up
```

関連キーワード

ACCEL

DURATION

SELECT(実数値関数)

SELECT (プログラムコマンド)

SPEED(モニタコマンド)

SPEED (プログラムコマンド)

SQRT

パラメータで指定した値の平方根を返す実数値関数です。

構文

SQRT (value)

パラメータ

value	平方根が計算される値を定義する実数値式です。

詳細

引数が 0 より大きい場合、引数の平方根を返します。引数が 0 未満の場合、エラーとなります。

例

次の例では、0.3507136 を返します。

```
SQRT(0.123)
```

次の例では、2.0 を返します。

```
SQRT(4)
```

次の例では、メッセージ*Negative square root*を返します。

```
SQRT(-5.462)
```

次の例では、11.45644 を返します。

```
SQRT(1.3125E+2)
```

> **注:** TYPE、 PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTRモニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

SQR

SQR

パラメータで指定した値の二乗値を返す実数値関数です。

構文

SQR (value)

パラメータ

value	二乗される値を定義する実数値式です。

詳細

この関数は、値の二乗値を計算します。結果は、（値×値）に等しくなります。

例

次の例では、0.015129 を返します。

```
SQR(0.123)
```

次の例では、16 を返します。

```
SQR(4)
```

次の例では、29.8334 を返します。

```
SQR(-5.462)
```

次の例では、17226.56 を返します。

```
SQR(1.3125E+2)
```

> **注:** TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTRモニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

SQRT

STATE

ロボットシステムの状態に関する情報を提供する値を返す実数値関数です。

構文

STATE (select)

使用上の留意事項

STATE 関数は、関数を実行するタスクが選択したロボットの情報を返します。

> **追加情報:** ロボットの状態に関する詳細情報については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

select 返されるステータス情報のカテゴリーを選択する実数値、変数、または式です。整数値として解釈されます。情報カテゴリを次に示します。詳しくは、「詳細」項を参照してください。

- 1: 全体的なロボットの状態
- 2: 現在および過去の動作
- 3: 現在の手動制御モード
- 4: コントローラーインターフェースパネル設定とハードウェアの状態
- 5: フロントパネル運転モード選択
- 6: 現在の動作の変更モード状態
- 7: 次の動作の変更モード状態
- 8: ペンダントに接続しているロボット
- 9: ロボット動作完了までの時間
- 10: 現在の動作の完了率(%)
- 11: 現在生成中の加速度プロファイルの部分
- 12: ALTER コマンドの状態
- 13: 軌道ジェネレーター実行レート(Hz)
- 14: 今後の使用のための予備
- 15: 実行された動作数
- 16～18: 今後の使用のための予備
- 19: 今後の使用のための予備(「Invalid argument」(-407)を返す)
- 20: 加速度増加時間
- 21: 一定加速度時間
- 22: 加速度減少時間
- 23: 一定速度時間
- 24: 減速度増加時間
- 25: 一定減速度時間
- 26: 減速度減少時間
- 27: 総動作時間
- 28～29: 今後の使用のための予備
- 30: フロントパネルインジケータの状態
- 31～29: 今後の使用のための予備
- 40: ベルト追跡ステータス

詳細

次に示すように、**select** = 1の場合、この関数はロボットの全体的状態に関する情報を返します。

値	説明
0	ロボットの電源が OFF になった後、システムをリセットしています。
1	致命的なエラーが発生し、ロボットの電源を ON できません。
2	ロボット電源が ON になるのを待機しています。
3	ロボットの電源が ON になり、初期化が実行されています。
4	手動制御モードが有効です。select = 3 を参照してください。
5	CALIBRATE コマンドが実行中です。
6	今後の使用のための予備
7	ロボットがプログラム制御下にある。
8	ロボットの電源は ON だが、キャリブレーションされていないため、動作できません。
9	今後の使用のための予備
10	フロントパネルの電源ランプゆっくり点滅（毎秒 1 回）して、押されるのを待っています。点滅ボタンが押されると、ロボットの電源は ON になります。
11	フロントパネルの電源ランプが、高速で点滅（毎秒 4 回）します。[COMP/PWR] ボタンが押されると、ロボットの電源は ON になります。 **注:** フロントパネルの電源ランプを高速点滅中に押した場合、ロボットの電源は OFF になります。通常のプロセスでは、電源を再度 ON にする必要があります。

次に示すように、**select** = 2 の場合、この関数は、現在または前回のロボット動作に関する情報を返します。ロボットがプログラムの制御下にある（STATE(1) = 7）の場合に限って、これらのモードを変更することができます。

値	説明
0	動作オペレーションがまだ実行されていません。
1	通常の軌道評価が進行中です（通常の加速、減速、区間遷移を含む）。
2	計画されたロケーションで動作が停止しました。RETRY コマンドは影響ありません。
3	計画外の最終ロケーションで位置エラーがヌリングされています。
4	ベルトウィンドウの振動のため、計画外のロケーションで動作が停止しました。RETRY コマンドは、前の動作を完了します。
5	REACTI または BRAKE オペレーションがトリガーされたために減速しています。
6	REACTI または BRAKE コマンドがトリガーされたために停止しました。RETRY コマンドは、前の動作を完了します。
7	ハードウェアエラーまたは ESTOP オペレーションのために減速しています。
8	ハードウェアエラーまたは ESTOP オペレーションにより停止しました。RETRY コマンドは、前の動作を完了します。
9	強制停止状態のために減速しています。
10	強制停止状態のために停止しました。

次に示すように、**select** = 3 の場合、この関数値は、現在の手動制御モードに関する情報を返します（詳細については、JOG プログラムコマンドを参照）。

値	説明
0	手動モード（選択なし）
1	フリージョイントモード
2	各ジョイント制御

値	説明
3	ワールド座標系制御
4	ツール座標系制御
5	コンピューター制御有効
6	今後の使用のための予備
7	ジョグトゥーモード
8	アラインモード
9	フレームモード

select = 4 の場合、その戻り値はプログラムが読み取るハードウェア状態に関する情報を返します。値は、対応する情報を示すビットフラグセットして解釈されます。

値	説明
^H1	今後の使用のための予備
^H2	今後の使用のための予備
^H4	ESTOP 回路が開いています（下記参照）。
^H8	HIGH POWER ボタンが押されています（下記参照）。
^H10	非常停止チャネル 1リレーオープン
^H20	非常停止チャネル 2リレーオープン
^H40	フロントパネルキースイッチが手動状態です。
^H80	今後の使用のための予備
^H100～^HF00	ESTOP ソース: 0x0 = 非常停止なし、または未解決（解決時間 120 ms待機） 0x1 = 非常停止ソース損失による非常停止 0x2 = フロントパネルによる非常停止

値	説明
	0x3 = ペンダントによる非常停止 0x4 = ユーザー ESTOP による非常停止 0x5 = ライン ESTOP 入力による非常停止 0x6 = ミュート安全ゲートによる非常停止（自動モードのみ） 0x7 = 自動モードから手動モードへの変更による非常停止 0x8 = 手動モードから自動モードへの変更による非常停止 0x9 - 0xE = 将来の使用用に予約 0xF = 未解決の非常停止ソース
^H1000	3 点有効スイッチクローズ（手動モードのみレポート）.

select = 5 の場合、戻り値はフロントパネルのスイッチ設定を返します。

値	説明
1	自動モード。
2	手動モード。

select = 6 の場合、この関数は、リアルタイムパス修正設備（変更モード）が有効かどうかを示す値を返します。戻り値が 0 の場合、現在の動作で変更モードは無効です。戻り値が 0 以外の場合、変更モードは有効で、この値の下位バイトには、経路修正を開始した ALTON コマンドで指定されたモードに対応するビットが含まれます。

select = 7 の場合、この関数は、リアルタイムパス修正設備（変更モード）が、次の計画動作で有効かどうかを示す値を返します。戻り値が 0 の場合、次の動作で変更モードは無効です。戻り値が 0 以外の場合、変更モードは有効で、この値の下位バイトには、経路修正を開始した ALTON コマンドで指定されたモードに対応するビットが含まれます。

select = 8 の場合、手動制御ペンダントに接続しているロボットの番号を返します。

select = 9 の場合、この関数は、現在の動作が完了するまでの時間（秒単位）を返します。値 0 は、動作が進行していないことを示します。連続経路動作の場合、STATE(9)の値は、次の動作に移行するまで動作ごとに減少し、次の動作までの残り時間に変化します。STATE(9)は、次の動作を反映してリセットされるまで 0 にはなりません。

select = 10 の場合、この関数は、完了した現在の動作の割合（%）の値を返します。値 100 は、動作が進行していないことを示します。連続経路動作の場合、STATE(10)の値は、次の動作に移行するまで動作ごとに増加し、次の動作の開始を反映した値に変化します。STATE(10)は、次の動作を反映してリセットされるまで 100 にはなりません。

追加情報: このパーセンテージは 0.8% の精度で返されます。

select = 11 の場合、この関数は、選択されたロボット用に現在生成されている加速プロファイルの一部に関する情報を示す値を返します。

値	説明
0	アイドル、軌道評価なし。
1	新しい区間の加速度増加。
2	一定加速度区間。
3	加速度減少。
4	一定速度区間。
5	モーション間の移行区間中、加速度増加。
6	モーション間の移行区間中、一定加速度。
7	モーション間の移行区間中、加速度減少。
8	減速度増加。
9	一定減速度。
10	減速度減少。
11	最終エラーをヌリング。

select = 12 の場合、この関数は、ALTER プログラムコマンドが現在選択されているロボットで実行され、軌道ジェネレーターがポストされた ALTER データを処理した後にクリアされた場合、0以外に設定されたフラグを返します。このフラグは、軌道ジェネレーターによるデータ処理で ALTER コマンドの実行調整に使用できます。

select = 13 の場合、この関数は、軌道ジェネレーター実行レート（Hz 単位）を示す値を返します。起動ジェネレータを実行している場合、、この関数は 62.5 の値を返します。

select = 15 の場合、この関数は、選択されたロボットで実行されている動作の番号を返します。ロボットに接続されたプログラムが最初に実行を開始するとき、この番号は 0 になります。各実行サイクルが開始するとき、カウンターは 1 にリセットされます。軌道ジェネレーターが新しい動作を評価する（または新しい連続パス動作に移行）たびに、値はインクリメントします。STATE(15) の値の範囲は 0～^HFFFF です。^HFFFF に達した後、値は 0 に戻ります。

次の命令は、STATE(15) の値に影響を及ぼします。

ALIGN、APPRO、APPROS、DEPART、DEPARTS、DRIVE、
JMOVE、MOVE、MOVES、MOVEF、READY

以降の複数動作に影響するコマンド（ACCEL、AMOVE、ABOVE、BELOW、DURATION、FLIP、LEFTY、NOFLIP、MULTIPLE、RIGHTY、SINGLE、SPEED、SPIN、TOOL、UNIDIRECTなど）は、実際には動作を開始しないため、STATE(15) の値に影響を及ぼしません。

注: 関数キーワードは、動作を発生しないため、STATE(15) の値に影響を及ぼしません。ロケーション値関数（DEST、FRAME、INVERSE、SCALE、SHIFT、TRANSなど）は、ロケーション値を単に計算するだけです。

select が 20～27 の場合、この関数は、現在の動作（またはロボットの停止中に前回実行された動作）の計画実行時間に関する詳細情報を返します。

モニタ速度設定用に修正された残りのモーション実行時間を返す STATE(9) とは異なり、STATE(20)～STATE(27) が返す値は、計画値であり、モニタ速度値の設定で影響を受けることはありません。これらの関数の戻り値（秒単位）を次に示します。

select	戻り値の説明
20	加速度増加時間。
21	一定加速度時間。
22	加速度減少時間。
23	一定速度時間。
24	減速度増加時間。
25	一定減速度時間。
26	減速度減少時間。
27	総動作時間（STATE(20)～STATE(26) の合計）

select = 30 の場合、この関数は、フロントパネルの電源ランプのステータスを返します。この関数が返す可能性のある値を次に示します。

値	説明
0	ランプ OFF。
1	ランプ ON。

値	説明
2	ランプが 4 Hz で点滅
3	ランプが 1 Hz で点滅

select = 40 の場合、この関数は、現在選択されているロボットがベルトを追跡している場合、0 以外の値にセットされたフラグを返します。

例

次の例では、ロボット動作中に REACTI がトリガーされたかどうかを判断するために STATE 関数を使用する方法を示しています。

```
REACTI 1001
MOVES final
BREAK
CASE STATE(2) OF
  VALUE 2:
    TYPE "Motion completed normally"
  VALUE 6:
    TYPE "Motion stopped by REACTI"
  VALUE 8:
    TYPE "Motion stopped by panic button"
END
```

関連キーワード

SELECT（プログラムコマンド）

SELECT（実数値関数）

STATUS（モニタコマンド）

STATUS（実数値関数）

STATUS

アプリケーションプログラムのステータス情報を返す実数値関数です。

構文

STATUS (program_name)

パラメータ

program_name	対象のアプリケーションプログラムの名前を指定する文字列定数、変数、式です。名前には、大文字または小文字を使用できます。プログラム名を指定しない場合、文字列を空( )にできますが、パラメータは省略できません。

詳細

この関数は、指定されたプログラムの実行状態に関する情報を返します。

プログラム名が指定されていない場合（パラメータの文字列が空[ ]）、その関数呼び出しを含むプログラムのタスク番号が返されます。この関数をプログラムで使用すれば、どのシステムタスクが実行されているかを判断できます。タスクとタスク番号の詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

プログラム名が関数パラメータとして指定されている場合、次のように、該当するプログラムのステータスが返されます。

戻り値	プログラムのステータス
-1	実行されていない。
-2	定義されていない。
-3	プログラムが読み書きモードでコピー、削除、名前の変更、編集されている場合で、書き込みインターロック中
-4	プログラムに構文エラーやシンタックスに誤りがある場合で、実行不可能
-5	プログラムが 1 つ以上のタスクで実行中、または読み取り専用モードで編集されている場合で、読み取りインターロック中

注: プログラムが複数のタスクで実行されている場合、STATUS関数は値 -5 を返します。これらのタスクの1つによって、いつプログラムが停止したかについては、STATUS 関数は判断できません。STATUS 関数は、すべてのタスクがプログラムの実行を停止するまで、値 -1 を返しません。

この関数は、無効なプログラム名が指定された場合（名前が先頭が文字以外など）、定義されていないステータスを返します。

例

次の例では、アプリケーションプログラムの実行が開始されたかどうかを判断するために STATUS 関数を使用する方法を示しています。

```
IF STATUS("pc.main") == -1 THEN
    EXECUTE 1 pc.main
END
```

注: プログラムがプログラムタスクによって実行中の場合、STATUS 関数は、値 -1 を返しません。この例は、状況によっては適切でない可能性があります。別のプログラムタスクの実行を開始するテクニックについては、EXECUTEプログラムコマンドの例を参照してください。

関連キーワード

DEFINED

STATE

STATUS

TESTP

STRDIF

ソート用に 2 つの文字列をバイト単位で比較する実数値関数です。

構文

STRDIF ($a, $b)

使用上の留意事項

この関数は、常にバイトを正確に比較します。UPPERシステムスイッチの設定を無視します。

パラメータ

$a $b のバイトと比較されるバイトを含む文字列定数、変数、式です。

$b $a のバイトと比較されるバイトを含む文字列定数、変数、式です。

詳細

この関数は、UPPER システムスイッチの設定にかかわらず、変換することなく符号なしバイト値を使用してバイト単位に文字列を比較します。2 つの文字列を異なる長さにすることができます。戻り値とその意味を次に示します。

戻り値	説明
-1	$a が $b より小さい。
0	$a と $b は同じである。
1	$a が $b より大きい。

例

次の例では、2 つの文字列変数をアルファベット順にソートし、TYPE コマンドを使用して結果を表示します。

```
$name[0] = "Michael"
$name[1] = "MARK"
CASE STRDIF($name[0],$name[1]) OF
  VALUE -1, 0:
    $list[0] = $name[0]
    $list[1] = $name[1]
  VALUE 1:
    $list[0] = $name[1]
    $list[1] = $name[0]
END
TYPE "Names in alphabetic order:  ", $list[0], " ", $list
[1]
```

関連キーワード

UPPER

SWITCH

システムスイッチの設定に関する情報を返す実数値関数です。

構文

SWITCH (switch_name)

SWITCH (switch_name[index]**)**

パラメータ

switch_name	対象のシステムスイッチの名前です。
index	インデックスで条件を付けることができるスイッチの場合、**index** は、特定のスイッチ要素を対象として指定する実数値、変数、式です。

詳細

この関数は、指定のスイッチが無効の場合、FALSE (0.0)を返します。それ以外の場合は、TRUE (-1)を返します。

スイッチ名には、識別しやすいように最小長の固有の省略名を付けることができます。

システムスイッチに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

次の例では、DRY.RUN スイッチが有効であるかどうかを確認します。有効な場合、モニタウィンドウにメッセージが表示されます。

```
IF SWITCH(DRY.RUN) THEN
    TYPE "DRY RUN mode is enabled"
END
```

関連キーワード

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE (プログラムコマンド)

SWITCH(モニタコマンド)

SWITCH (プログラムコマンド)

TAN

指定した角度の三角関数のタンジェント値を返す実数値関数です。

構文

TAN (value)

使用上の留意事項

角度パラメータの単位は度である必要があります。

パラメータは、360 度の剰余として解釈されますが、過度に大きな値を指定した場合、戻り値の精度が低くなる可能性があります。

パラメータ

value	角度の値を定義する実数値式です。

詳細

引数（度単位）のタンジェント値を返します。

例

次の例では、0.5773503 を返します。

```
TAN(30)
```

次の例では、1 を返します。

```
TAN(45)
```

> **注:** TYPE、PROMPT、および同様のコマンドは、単精度値として上記例の結果を出力します。しかし、実際には、倍精度値として保存、処理されます。LISTRモニタコマンドは、実数値を最大限の精度まで表示します。

関連キーワード

ASIN

COS

ACOS

SIN

TAN

ATAN2

TAS

実数値変数の現在の値を返し、新しい値を割り当てる実数値関数です。他のプログラムタスクが同時に変数を修正できないようにするために、2つのアクションは不可分的に実行されます。

構文

TAS (variable, new_value)

使用上の留意事項

eV+ システムは、複数のプログラムタスクで共有されるグローバル変数に対して、保護スキームを実施しません。このようなグローバル変数の使用を管理することは、プログラマーの責任です。TAS 関数は、共有変数へのアクセスに対して論理的インターロックを行うために使用できます。

また、この関数は、複数のプログラムタスクがグローバル配列に同時にアクセスするときの制限管理にも使用できます。2 つ以上のタスクが同時に配列サイズの増加を試みた場合、プログラムの実行は失敗する可能性があります。詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

variable	評価と新しい値の割り当て用に使用される実数値変数名です。関数の実行時に変数が定義されていない場合、この関数は値0を返します。
new_value	指定された変数に割り当てる新しい値を定義する実数値、変数、式です。

詳細

異なるプログラムタスクが同時に実行した場合、システムプロセッサの時間を共有するため、計算や変数へのデータの保存中に別のタスクによってタスクが中断される可能性があります。2つ以上のタスクがデータを共有する場合、プログラムはインターロックスキームを実施し、データの一部が更新されないように防ぐ必要があります。

TAS 関数を使用することで、複数の eV+ タスクが共有データ構造を修正できるようになります。この関数は、ロックしているタスクがデータ構造を修正している間、それ以外のタスクをロックアウトする方法をタスクに提供します。TAS関数を使用しない場合、ポーリング法がさらに複雑になり、例えば、複数のプログラムが同時に制御変数を設定しないようにするために、変数を管理する必要があります。

共有変数の使用例として、グローバル変数を増減するこの eV+ プログラムを検討してみます。

```
.PROGRAM tas_test()
        AUTO i
        FOR i = 1 TO 1E+06
            counter = counter+1
            counter = counter-1
        END
.END
```

変数「counter」が0から開始し、プログラム「tas_test」がタスク1とタスク2で同時に実行される場合、実行が完了した後、counterの値は0になると予想されますが、通常、正しくありません。2つのタスクが同じ変数を同時に修正しているため、値が破損します。

次に示すように、インターロックを利用したプログラムに修正することがこの問題に対処できます。

```
.PROGRAM tas_test()
        AUTO i
        FOR i = 1 TO 1E+06

            WHILE TAS(locked,TRUE) DO
                WAIT
            END

            counter = counter+1
            counter = counter-1

            locked = FALSE
        END
.END
```

2つ以上のタスクでプログラムが同時に実行されている場合、共有変数にアクセスする間、各タスクが他のタスクをブロックするため、変数「counter」の値は、常にプログラムの開始前と同じ値になります。

変数が定義されていない場合、TAS関数はFALSEを返すため、プログラムの実行前に、ロックされているグローバル変数を初期化する必要はありません。ロックは、暗黙にOFF状態で始まります。

> **注:** 他のタスクがロックの解除を待機できるように、ロックはすぐに解除されます。
>
> ロックは、適用後、*常に*解除されるように注意してください。ロックが解除されなければ、他のタスクは永遠にブロックされ、*現在の*タスクもまた、 次回ロックを要求するときにブロックされます。

例

次の例では、TAS関数を使用する重要な点である、別のプログラムタスクも使用するデータに対して排他的にアクセスできるようにアプリケーションプログラムを保証する例を示します他のアプリケーションプログラムでデータにアクセスする必要がある場合は、同じコマンドシーケンスを使用する必要があります。

実数値変数「data.locked」は、データがインターロックされていない場合は値FALSE、インターロックされている場合は値TRUEになります。他のプログラムタスクがすでにこの変数をTRUEに設定しているかを検出するために、TAS関数はこの変数をTRUEに設定します。TASが評価し、値を不可分的に設定した後、両プログラムは、競合を検出せずに同時に「data.locked」をTRUEにしません。

セマフォ変数「data.locked」を使用する場合は、次に示す3つのステップが必要になります。

1. 「ロック解除」設定（FALSE）になっているロック変数を探し、「ロック」設定（TRUE）を同時に適用します。次に示すように、このタスクが自分に対してロックをアサートするとき、別のタスクがロック変数を「ロック解

除」設定（FALSE）に設定するまでループサイクルを継続します。

```
WHILE TAS(data.locked,TRUE) DO
    WAIT
END
```

2. 共有データにアクセスする目的の命令を実行する。
3. 共有データ構造のロックを解除する。

```
data.locked = FALSE
```

このWHILEループは、変数「data.locked」がFALSEになるまで、プログラムの実行をブロックします。共有データにアクセスするために、他のプログラムがセマフォ変数をロックしている場合、このプログラムはブロックされます。TAS関数は、関数が実行されるたびに変数「data.locked」をTRUEに設定しますが、変数の値がすでにTRUEになっている場合は、効果はありません。

プログラムが共有データに対する排他的アクセスを取得した後、安全にデータにアクセスできます。

上記ステップ3の最後のステートメントは、アプリケーションが他のプログラムタスクの実行でアクセスできるように、データを解放しています。

関連キーワード

CAS

TASK

プログラム実行タスクに関する情報を返す実数値関数です。

構文

TASK (select, task_num)

使用上の留意事項

select = 1で3の値が返された場合（ABORTのためにTASKが停止した）、PROCEEDプログラムコマンドを使用して、このプログラムの実行を再開できます。この状態を確認する前に、プログラムの実行が停止したことを確認するのがお勧めします。詳細については、次の例を参照してください。

パラメータ

select	返されるタスク情報のカテゴリーを選択するためのオプションの実数式です。値の範囲は0、1、2 です。このパラメータが省略された場合、値は 0 になります。
task_num	アクセスされるシステムプログラムタスクを指定するオプションの整数値です。

詳細

この関数は、システムプログラム実行タスクに関するさまざまな情報を返します。タスク実行の詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

次に示すように、選択パラメータは、返される情報の種類を決定します。

select = 0	**タスク番号:** この関数は、現在のプログラムを実行しているタスクの番号を返します。
select = 1	**タスク実行状態:** task_num パラメータで指定されたタスクの実行状態を返します。戻り値の意味については、以下を参照してください。

値	意味
-1	無効なタスク番号
0	アイドル状態
1	プログラム完了のため停止
2	プログラム実行エラーのため停止（未定義値など）

値	意味
3	ABORT
4	実行
5	PAUSE またはブレークポイントのため停止
7	シングルステップ実行のため停止

select = 2

タスクステータスビット: task_num パラメータで指定されたタスクに関する以下の情報を示す、ビットフラグセットとして解釈される整数値を返します。

ビット番号	ビットマスク	ビット ON の意味
1	1	今後の使用のための予備
2	2	タスクがロボットにアタッチされている。

例

次の例では、select = 1が3の値が返された後、プログラムの実行が停止したことを確認します。次に、PROCEEDプログラムコマンドを使用されて、プログラムの実行を再開されます。

```
IF (TASK(1,2) <> 4) THEN
    IF (TASK(1,2) == 3) THEN
        IF try.proceed THEN
           PROCEED 2
        ELSE
           EXECUTE 2 same.prog()
        END
    END
END
```

次の例では、モニタウィンドウにプログラムを実行中のタスク番号を表示します。

```
TYPE "This program is running as task number :", TASK()
```

次の例では、タスク 2 でプログラム「named pc.job.2」の実行を開始するかどうかを決定するためにTASK関数を使用する方法を示しています。

```
IF TASK(1,2) <> 4 THEN
    IF STATUS("pc.job.2") == -1 THEN
        EXECUTE 2 pc.job.2()
```

```
        ELSE
            TYPE /B, "Can't start task 2"
        END
    END
```

次の例では、現在のタスクがロボットにアタッチされているかを確認し、結果に応じてモニタウィンドウにメッセージを表示します。

```
    IF (TASK(2,TASK(0))<>2)THEN
        TYPE "robot not attached"
    END
        IF (TASK(2,TASK(0))==2)THEN
        TYPE "robot is attached"
    END
```

関連キーワード

ERROR

PROCEED（プログラムコマンド）

STATE

STATUS

STATUS（実数値関数）

$TIME

現在のシステムの日付と時刻、または指定した日付と時刻のいずれかを含む文字列値を返す文字列関数です。

構文

$TIME (date, time)

パラメータ

date 年、月、日を示すオプションの整数値です。次に示すように、値は1～12 月の範囲に解釈されます。

```
date = (year-1980)*512 + month*32 + day
```

time 午前 0 時を過ぎた時、分、秒を示すオプションの整数値です。次に示すように、値は 0～23 時の範囲に解釈されます。

```
time = hour*2048 + minute*32 + second/2
```

注: この関数は、必ず日付と時刻の両方を含む文字列を返します。date パラメータが省略され、time パラメータが指定されている場合、誤った日付の文字列が返される場合があります。

詳細

date、time パラメータの両方を省略した場合、次に示すフォーマットで現在のシステムの日付と時刻を返します。システムクロックが初期化されていない場合、空の文字列が返されます。

date、time パラメータを指定した場合、値は次に示すフォーマットで ASCII 文字列に変換され、その文字列が返されます。この操作は、TIMEで生成された出力値を復号するために使用されます。

日付と時刻は、次に示すように各要素が定義されたフォーマット（dd-mmm-yy、hh:mm:ss）で出力されます。

要素	説明
dd	月内の日（1～31）
mmm	月は、3 文字の省略文字で指定されています（JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC）。
yy	1980～1999 年は 80～99、2000～2079 年は 00～79 で表されます。
hh	日内の時間（0～23）
mm	1 時間を超えた分数（0～59）
ss	1 分を超えた秒数（0～59）

注: $TIME 関数は、date、time パラメータのいずれかが指定されている場合、システム時刻の代わりに指定された引数を変換します。しかし、両方のパラメータを示す文字列を生成しようとします。

date の値を指定しない場合、01-Jan-80（1980 年 1 月 1 日）を返します。time の値を指定しない場合、時刻のサブストリング 00:00:00 を返します。次のステートメントは、日付と時刻のみを返す場合に使用できます。

```
$date = $MID($TIME(date,),1,9)

$time = $MID($TIME(,time),11,8)
```

例

次の例では、ビットマスクを基準としてモニタウィンドウに時刻を表示します。

```
TYPE $TIME(10333,41472)
```

次の例では、モニタウィンドウに現在のシステム時間を表示します。

```
$current_time=$TIME()
TYPE $current_time
```

関連キーワード

TIME

$TIME4

$TIME4

現在のシステムの 4 桁の日付と時刻、または指定した 4 桁の日付と時刻のいずれかを含む文字列値を返す文字列関数です。

構文

$TIME4 (date, time)

パラメータ

date　　年、月、日を示すオプションの整数値です。次に示すように、値は 1～12 月の範囲に解釈されます。

```
date = (year-1980)*512 + month*32 + day
```

time　　午前 0 時を過ぎた時、分、秒を示すオプションの整数値です。次に示すように、値は 0～23 時の範囲に解釈されます。

```
time = hour*2048 + minute*32 + second/2
```

注: この関数は、必ず日付と時刻の両方を含む文字列を返します。date パラメータが省略され、time パラメータが指定されている場合、誤った日付の文字列が返される場合があります。

詳細

date、time パラメータの両方を省略した場合、次に示すフォーマットで現在のシステムの日付と時刻を返します。システムクロックが初期化されていない場合、空の文字列が返されます。

date、time パラメータを指定した場合、値は次に示すフォーマットで ASCII 文字列に変換され、その文字列が返されます。この操作は、TIMEで生成された出力値を復号するために使用されます。

日付と時刻は、次に示すように各要素が定義されたフォーマット（dd-mmm-yyyy、hh:mm:ss）で出力されます。

追加情報: 次に示す範囲以外の値を使用した場合、想定外の値を返す可能性があります。

要素	説明
dd	月内の日（1～31）
mmm	月は、3 文字の省略文字で指定されています（JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC）。
yyyy	年（1980～2079）
hh	日内の時間（0～23）
mm	1 時間を超えた分数（0～59）
ss	1 分を超えた秒数（0～59）

注: $TIME4 関数は、date、time パラメータのいずれかが指定されている場合、システム時刻の代わりに指定された引数を変換します。しかし、両方のパラメータを示す文字列を生成しようとします。date の値を指定しない場合、01-Jan-80（1980 年 1 月 1 日）を返します。time の値を指定しない場合、時刻のサブストリング 00:00:00 を返します。次のステートメントは、日付と時刻のみを返す場合に使用できます。

```
$date = $MID($TIME4(date,),1,11)

$time = $MID($TIME4(,time),13,8)
```

例

次の例では、モニタウィンドウに"31-Jan-2020 20:16:00"を表示します。

```
AUTO REAL date, year, month
AUTO REAL day, time, hour, minute, second

date = (year-1980)*512 + month*32 + day
time = hour*2048 + minute*32 + second/2

TYPE $TIME4(20543,41472)
```

関連キーワード

TIME

$TIME

TIME

所定の文字列パラメータで指定された日付または時刻のいずれかを示す整数値を返す実数値関数です。

構文

TIME (string, **select**)

パラメータ

string　次に示すフォーマットで日付と時刻を指定するオプションの文字列変数、定数、式です。(詳細については、以下を参照)

select　戻り値を選択する実数値、変数、または式(整数として解釈)です。**select** が以下の値以外の場合、エラーになります。

select	戻り値	ss の定義
1	日付	(年-1980)*512 + 月*32 + 日
2	時刻	時*2048 + 分*32 + 秒/2
3	秒	1 分を超えた秒数

詳細

この関数は、日付と時刻をコンパクト（符号なし 16 ビット）整数形式にエンコードする場合に使用できます。日付と時刻の整数値を取得した後、他の日付と時刻の値と算術的に比較し、条件の前後を判断します。

> **注:** 日付または時刻を計算するために、エンコードされた整数値を操作しないでください。例えば、エンコードされた日付の値に日数を加算しないでください。

文字列パラメータを指定した場合、日付と時刻を文字列で指定する必要があります。文字列の値は、次の形式のいずれかにする必要があります。

```
dd-mmm-yy hh:mm:ss      dd-mmm-yyyy hh:mm:ss
dd-mmm-yy hh:mm         dd-mmm-yyyy hh:mm
```

この関数は、入力された文字列が許容可能な形式でない場合、値 -1 を返します（次の例を参照）。

日付と時刻の各要素は次のように定義されます。

> **追加情報:** 次に示す範囲以外の値を使用した場合、想定外の値を返す可能性があります。

要素	説明
dd	月内の日（1～31）
mmm	3 文字の略語で指定される月名（JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV または DEC）
yy	1980～1999 年は 80～99、2000～2079 年は 00～79 で表されます。
yyyy	年（1980～2079 年）
hh	日内の時間（0～23）
mm	1 時間を超えた分数（0～59）
ss	1 分を超えた秒数（0～59：ss が省略された場合 0 とみなす）

文字列パラメータが指定されず、**select** パラメータが1の場合、システムクロックの現在の日付が返されます。さらに、システムクロックの現在の時刻は、プログラムタスクの（内部）管理データに保存されます。文字列パラメータが指定されず、**select** パラメータが 2、または 3 の場合、*前回*保存されたシステムクロックの時刻に対して、選択された時刻の値が返されます。

例

次の例では、TIME 実数値関数を使用して、プロンプトの後に有効な日付と時刻を入力する方法を示してします。

```
PROMPT "Enter the date and time (dd-mmm-yy hh:mm:ss): ", $time

WHILE (TIME($time,1) == -1) DO
    TYPE /B, "    Cannot interpret date/time."
    PROMPT "Try again (dd-mmm-yy hh:mm:ss): ", $time
END
TIME $time
```

関連キーワード

$TIME

$TIME4

TIMER

指定されたシステムタイマーの現在のタイマー値を返す実数値関数です。

構文

TIMER (timer_number)

使用上の留意事項

タイマーの精度と解像度は、選択したタイマーに応じて異なります。解像度を最大にするためには、double 型変数を使用してください。

パラメータ

timer_number 読み取るタイマーの番号を指定する整数として解釈される実数値、変数、または式です。値は、-4～15 の範囲内にする必要があります。

値	説明
1-15	タイマーの解像度は 1 ms で、最大カウント数は、> 2.E+009 です。タイマーは、TIMER キーワードで設定された時間から最大 596 時間までの間隔を測定するために使用できます。
0	解像度 1 ms、最大カウント数約 2.E+009 で eV+ システムが開始してからの秒数を返します。システム動作の最初の 596 時間のみ有効のため、通常、使用すべきではありません。
-1	eV+ システムが開始してからの時間を 16 ms ごとにカウントした値の下位 24 ビットを返します。最大 74 時間までの時間間隔を計算できます。
-2	eV+ システムが開始してからの時間を 1 ms ごとにカウントした値の下位 24 ビットを返します。最大 4.6 時間までの時間間隔を計算できます。
-3	eV+ が開始してからの時間(秒数)を 52 ビット倍精度値として返します。このタイマー

値	説明

	の解像度は 1 ms で、最大カウント数は、> 4.E+015 です。100,000 年より大きい間隔を計算できます。double 型変数を必要とし、BAND 操作またはスケール係数を使用しない場合以外のタイマー -1 および -2 などに使用されます。
	`AUTO DOUBLE start_time, interval`
	`start_time = TIMER(-3)`
	`...`
	`interval = TIMER(-3)-start_time`

-4	現在のロボット位置、またはこのタスク用のベルトエンコーダラッチの倍精度時刻を返します。タイマーの解像度は、1 us です。この時刻は、128 秒間のみ有効です。

詳細

タイマー-1 と -2

タイマー 1～15 を使用しない、または 15 個以上のタイマーを必要としない場合、次のようにタイマー -1 と -2 を使用できます。

```
AUTO DOUBLE start_time, interval, scale
scale = 62.5                        )
start_time = TIMER(-1)
...
interval = ((TIMER(-1)-start_time) BAND ^HFFFFFF)/scale
```

timer_number = -3 の設定は、double 型値を使用できる場合に限り、間隔を計算する望ましい手段です

時間計算に使用される eV+ 変数の種類は、最大解像度で計算できる最大間隔に影響するため、次の事項を検討する必要があります。

- 標準の実数値変数は 24 ビットで、時間間隔を 16,777,216 (2^{24})カウントに制限する解像度になります。この制限は、msタイマーの場合は約 4.6 時間、16 ms タイマーの場合は 74 時間になります。
- double 型実数値変数は 52 ビット精度で、さまざまなタイマーの最大解像度を保存できます。明示的に指定しない場合、これがデフォルトになります。

例

次の例では、TIMER プログラムコマンドと TIMER 関数を使用して、サブルーチンの実行時間を測定する方法を示しています。

```
TIMER 1 = 0
CALL test.routine()
```

```
TYPE "Elapsed time =", TIMER(1)," seconds"
```

関連キーワード

TIMER

TOOL

最新の TOOL 命令で指定された変換値を返す変換関数です。

構文

TOOL

使用上の留意事項

TOOL 関数は、その関数を実行するタスクが選択しているロボットの情報を返します。

> **注:** この関数を実行するタスクがロボットを選択していない場合、この関数の出力は無効になります。
>
> **追加情報:** モニタコマンドステートメント LISTLTOOL を現在のツール設定表示に使用できます。

例

次の例では、モニタウィンドウに現在の TOOL 変換値を表示します。

```
LISTL TOOL
```

次の例では、現在の TOOL 値を保存します。

```
SET save.tool = TOOL
```

関連キーワード

TOOL (プログラムコマンド)

TPS

1秒あたりのシステムクロックのチック数を返す実数値関数です。

構文

TPS

例

次の例では、TPS 関数とWAIT キーワードを使用してタイムアウトを5 秒に設定し、システムクロックの1チックごとにイベントの発生をテストする方法を示します。

```
FOR ticks = 1 TO 5*TPS
    IF SIG(1001) THEN
        TYPE "Signal ON"
        HALT
    END
    WAIT
END
TYPE "Time-out while waiting for signal 1001"
```

TRANS

指定の X、Y、Z の位置変位と y、p、r の回転方向から計算された変換値を返す変換関数です。

構文

TRANS (X_value, Y_value, Z_value, y_value, p_value, r_value)

使用上の留意事項

パラメータが省略されている場合、省略された値は 0 とみなされます。

パラメータ

X_value Y_value Z_value	変位に関する、X、Y、および Z の各成分のオプションの式です。
y_value p_value r_value	yaw、pitch、および roll 方向の各成分のオプションの式です。

詳細

入力パラメータ値は、ロケーション変数に割り当てることができる変換値の計算、または複合変換やモーションリクエストで使用されます。

例

次の例では、「r」が半径、「angle」が円の回転角である円上のポイントを返します。

```
TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
```

次の例では、回転角 1 度ごとにロボットのツールポイントを円周に沿って移動させます。「frame」は、円の中心位置と円弧平面を定義する変換です。「r」は円の半径、「angle」は円の回転角です。

```
FOR angle = 0 TO 360-1
    MOVE frame:TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0,
0)
END
```

関連キーワード

DECOMPOSE

DX

DY

DZ

FRAME

#PPOINT

SET

SHIFT

TRANSB

$TRANSB

2 進数に表現された変換値を含む 48 バイトの文字列を返す文字列関数です。

構文

$TRANSB (transformation type)

パラメータ

transformation	文字列に変換される値を定義する変換変数、または変換関数(または合成変換) です。
type	位置座標の種類(0 = フロート、1 = ダブル) を指定する実数値式です。

詳細

この関数は、主にデータファイルの出力レコードの変換値を 2 進数に変換するために使用されます。

この関数は、指定された変換値を 2 進数で表される 12 個の内部構成要素に変換します。変換を定義する 12 個の値は、3 x 4 の変換マトリックスの構成要素であり、列ごとに保存されます。12 個の各 32 ビット値は、連続した 4 文字（1 文字 8 ビット）として文字列に格納され、最終的に、48、96 文字になります。

変換には、IEEE 単精度または倍精度標準浮動小数点形式が使用されます。浮動小数点数形式の詳細については、FLTB、DBLB 実数値関数を参照してください。

関連キーワード

$FLTB

FLTB

TRANSB

TRANSB

48 バイトまたは 96 バイトの文字列で示される変換値を返す変換関数です。

構文

TRANSB (string, first_char, type)

パラメータ

string	変換される 48 バイトまたは 96 バイトを含む文字列式です。
first_char	48 バイトの文字列の開始位置を指定するオプションの実数式です。 first_char が省略、または値が 0、1 の場合、文字列の先頭から 48 バイトまたは 96 バイトが抽出されます。first_char が 1 より大きい場合、抽出するバイトの開始文字位置として解釈されます。例えば、値 2 の場合、49 番目または 97 番目のバイトから抽出されます。first_char に入力文字列の末尾を超えた 48 バイトまたは 96 バイトを指定した場合、エラーになります。
type	位置座標の種類(0 = フロート、1 = ダブル) を指定する実数値式です。

詳細

この関数は、主に入力データレコードの変換値を示す バイナリ表記を eV+ が内部的に使用できる値に変換するために使用されます

指定された文字列の連続した 48 バイトまたは 96 バイトは、IEEE 標準形式の 12 個の単精度（32 ビット）浮動小数点数、または倍精度（64 ビット）浮動小数点数として解釈されます。浮動小数点数形式の詳細については、DBLBおよび FLTB関数の説明を参照してください。12 個の値は、列ごとに保存された 3 x 4 の変換マトリックスの構成要素として解釈されます。

関連キーワード

DBLB

FLTB

TRANS

$TRANSB

TRUE

eV+ で使用される、論理的な真を表すための値を返す実数値関数です。

構文

...TRUE

詳細

この名前付き定数は、真、偽の条件指定が必要な状況で役立ちます。戻り値は -1 です。

例

次の例では、サブルーチンのプロセスが実数値変数エラーのため値 TRUE を返すまで実行を継続します。

```
DO
    CALL process(error)
UNTIL error == TRUE
```

次の例では、無限に実行します。

```
WHILE TRUE DO
    CALL move.part()
END
```

関連キーワード

OFF

ON

FALSE

$TRUNCATE

ASCII の NULL 文字が見つかるまで入力文字列のすべての文字を返す文字列関数です。

構文

$TRUNCATE (string)

パラメータ

string	切り詰める文字列を指定する文字列変数、定数、または式です。

詳細

この関数は、ブレーク文字として指定された ASCII NUL (^H00)を使用した $DECODE関数と似た動作をします。$TRUNCATE と $DECODE関数の違いを次に示します。

- 文字列式を入力できます。
- 入力文字列は修正されません。

使用上の留意事項

単純なため、$TRUNCATE 関数は、$DECODE 関数よりも実行速度が速くなります。

例

次の例では、文字列変数「$substring」に「abcdef」と等しい値を設定しています。結果が入力から明らかに分かる場合に$TRUNCATE命令を実行したくない人為的な状況です。しかし、この関数が無作為に文字列式をスキャンし、NULL で区切られた最初のサブストリングを返すことを示しています。

```
$substring = $TRUNCATE("abcdef"+$CHR(0)+"ghijk")
```

関連キーワード

$DECODE

$UNPACK

128 文字の文字列変数配列からサブストリングを返す文字列関数です。

構文

$UNPACK (string_array[index]**, first_char, num_chars)**

パラメータ

string_array	サブストリングが抽出される文字列配列変数です。配列内の各文字列は定義され、128 文字長と想定されます。
index	最初の配列要素を特定するオプションの整数値です。first_char の値は、この index で指定された要素に関連すると解釈されます。 index が指定されていない場合、要素 0 とみなされます。
first_char	文字列配列内にあるサブストリングの最初の文字列位置を指定する実数値式です。値 1 は、指定した文字列配列要素の最初の文字に対応します。この値は、0 以上に設定する必要があります。 first_char の値は、128 以上に設定できます。この場合、アクセスされる配列要素は、関数コールで指定された要素に従います。例えば、値 130 は、index で指定された次の配列要素の 2 番目の文字に相当します。
num_chars	関数が返す文字数を指定する実数値式です。この値は、0～128 の範囲に設定できます。

詳細

この関数は、文字列配列からサブストリングを抽出します。サブストリングが文字列配列の 2 つの要素に重複することができます。例えば、要素[3]の 127 番目の文字から始まる 10 文字のサブストリングが、要素[3]の最後の 2 文字から要素[4]の最初の 8 文字で構成することができます。

文字列配列を効率的に利用するために、この関数は、配列要素のすべてが定義され、128 文字長であると想定します。多次元配列の場合、サブストリングの位置を特定するために、一番右側の配列インデックスのみ増加されます。例えば、要素[2,3]の次の要素は、要素[2,4]になります。

例

次の例では、文字列変数「$substring」に文字列配列「$list[]」から抽出されるサブストリングと等しい値を設定しています。サブストリングは、要素「$list[3]」で始まるように指定されています。しかし、最初の文字の位置が 130 のため、11 文字のサブストリングは、実際には「$list[4]」の 2 番目から 12 番目の文字で構成されます。

```
$substring = $UNPACK($list[3], 130, 11)
```

関連キーワード

$MID

PACK

VAL

入力文字列の文字で示される実数値を返す実数値関数です。

構文

VAL (string)

使用上の留意事項

入力文字列を指数表記の数字にすることができます。

入力文字列の先頭に、基数表記（^H）を含めることができます。

入力文字列の数字部分の前に、+ または - 符号を含めることができますが、オプションの基数の後にする必要があります。

数字または基数表記として解釈できない文字は、変換される文字の終わりとして扱われます。

パラメータ

string	文字列定数、変数、式です。

例

次の例では、実数値 123 を返します。

```
VAL("123 Elm Street")
```

次の例では、実数値 0.012 を返します。

```
VAL("1.2E-2")
```

次の例では、実数値 255 を返します。

```
VAL("^HFF")
```

関連キーワード

ASC

$ENCODE

FLTB

INTB

LNGB

VLOCATION

指定されたビジョンシーケンスの実行によるデカルト変換結果を返す変換関数です。x、y、z、yaw、pitch、roll で示される変換結果が戻り値となります。

構文

VLOCATION(**$ip**, **sequence_id**, tool_id, instance_id, result_id, index_id, frame_id)

使用上の留意事項

詳細については、*『ロボット用画像センサソフトウェアACE Sightリファレンスマニュアル (SBCE-400)』*を参照してください。

パラメータ

$ip	ビジョンサーバーの IP アドレスです。ビジョンサーバーは、Robot Vision Manager ビジョンソフトウェアが動作している PC であり、標準の IP アドレス形式 (192.168.1.120など) を使用します。
sequence_id	ビジョンシーケンスのインデックスです。最初のシーケンスは 1 です。
tool_id	ビジョンシーケンスでのツールインデックスです。オプションです。最初のツールは1 です。
instance_id	結果フレームを指定するためのインスタンスインデックスです。オプションです。
result_id	オプションの結果識別子 (ID) です。 通常、この値は 1311 です。グリッパーのオフセット位置に対して、この値を 1400 に設定し、グリッパーの各オフセットごとに1 ずつ増加することができます。最大値は 1499 です。
index_id	内部使用のための予備 (値は常に 1 です)。
frame_id	指定されたインスタンスに含まれる結果の検索に使用する、フレームのインデックスです。オプションです。

詳細

オプションのパラメータに値が指定されていない場合、デフォルトの値は 1 とみなされます。

グローバルな値を検索するには、次のパラメータ設定値を使用します。

- sequence_id: -1
- tool_id:-1

カメラの値を検索するには、次のパラメータ設定値を使用します。

- sequence_id: -1
- tool_id: cameraIndex

ロボットに関連付けられたカメラの値を検索するには、次のパラメータ設定値を使用します。

- sequence_id: -1
- tool_id: cameraIndex
- index_id: robotIndex

シーケンスの値を検索するには、次のパラメータ設定値を使用します。

- sequence_id: sequenceIndex
- tool_id:-1

ベルトキャリブレーション関連の値を検索するには、次の表の情報を使用します。

Property	sequence_id	tool_id	instance_id	result_id	index_id	frame_id
Frame	-1	cameraindex	n/a	10000	robotindex	n/a
UpstreamLimit				10001		
DownstreamLimit				10002		
NearsideLimit				10003		
VisionOrigin				10050		

ベルトラッチキャリブレーションのオフセットを検索するには、次の値を使用します。

- Property: LatchCalibrationOffset
- sequence_id: -1
- tool_id: Sysmac Studio のRobot Vision Manager Latch Calibration のキーワードマッピングパラメータで定義された参照番号。
- instance_id: n/a
- result_id: 10010
- index_id: robotIndex
- frame_id: n/a

例

次の例では、一番目のグリッパーのオフセットを使用することを示すresult_id 1311 を指定して、見つかったインスタンスの位置を読み出します。これは、result_id に 1400 を指定した場合と同様の結果になります。

```
SET location = VLOCATION($ip, 1, 2, 1, 1311)
```

次の例では、グリッパーの最初のオフセット位置が1400 の6 つのグリッパーオフセット位置 を検索します。

```
SET location = VLOCATION ($ip, 1, 2, 1, 1400)
SET location = VLOCATION ($ip, 1, 2, 1, 1401)
SET location = VLOCATION ($ip, 1, 2, 1, 1402)
SET location = VLOCATION ($ip, 1, 2, 1, 1403)
SET location = VLOCATION ($ip, 1, 2, 1, 1404)
SET location = VLOCATION ($ip, 1, 2, 1, 1405)
```

次の例では、BeltCalibrationFrameの インデックスである10000を指定して、 ベルトフレームの位置を読み出しています。

```
VLOCATION ($ip, -1, cameraIndex, , 10000, robotIndex)
```

次の例では、VisionOrigin インデックスが10050 のビジョン原点の位置を検索します。

```
VLOCATION ($ip, -1, cameraIndex, , 10050, robotIndex)
```

関連キーワード

VPARAMETER

VRESULT

VSTATE

VPARAMETER

ビジョンツールパラメータの現在の値を返す変換関数です。

構文

VPARAMETER (**$ip**, **sequence_id**, tool_id, parameter_id, index_id, object_id)

使用上の留意事項

詳細については、*『ロボット用画像センサソフトウェアACE Sightリファレンスマニュアル (SBCE-400)』*を参照してください。

パラメータ

$ip	ビジョンサーバーの IP アドレスです。ビジョンサーバーは、Robot Vision Manager ビジョンソフトウェアが動作している PC であり、標準の IP アドレス形式 (192.168.1.120など) を使用します。
sequence_id	ビジョンシーケンスのインデックスです。最初のシーケンスは 1 です。
tool_id	ビジョンシーケンスでのツールインデックスです。オプションです。最初のツールは1 です。
parameter_id	オプションのパラメータ識別子 (ID) です。
index_id	内部使用のための予備値は常に 1 です。
object_id	一部のパラメータは、配列の特定の値にアクセスするためにオブジェクトのインデックスが必要になります。

詳細

オプションのパラメータに値が指定されていない場合、デフォルトの値は 1 とみなされます。

グローバルな値を読み出すには、次のパラメータ設定値を使用します。

- sequence_id: -1
- tool_id:-1

カメラの値を読み出すには、次のパラメータ設定値を使用します。

- sequence_id: -1
- tool_id: cameraIndex

シーケンスの値を読み出すには、次のパラメータ設定値を使用します。

- sequence_id: sequenceIndex
- tool_id: -1

Scale (10004)に対してベルトキャリブレーション関連の値を読み出すには、次のパラメータ設定値を使用します。

- sequence_id = -1
- tool_id = cameraIndex
- index_id = robotIndex
- object_id = n/a

Mode (10200)に対してシーケンス関連の値を読み出すには、次のパラメータ設定値を使用します。

- sequence_id = sequenceIndex
- tool_id = -1
- index_id = n/a
- object_id = n/a

例

次の例では、ベルトキャリブレーションのスケール値を読み出しています。

```
scalevalue = VPARAMETER ($ip, -1, cameraIndex, 10004, robotIndex)
```

関連キーワード

VLOCATION

VRESULT

VSTATE

VRESULT

ビジョンツールの指定された結果、または指定されたツールの状態を返す実数値関数です。

構文

VRESULT (**$ip**, **sequence_id**, tool_id, instance_id, result_id, index_id, frame_id)

使用上の留意事項

詳細については、*『ロボット用画像センサソフトウェアACE Sightリファレンスマニュアル (SBCE-400)』*を参照してください。

パラメータ

$ip	ビジョンサーバーの IP アドレスです。ビジョンサーバーは、Robot Vision Manager ビジョンソフトウェアが動作している PC であり、標準の IP アドレス形式 (192.168.1.120など) を使用します。
sequence_id	ビジョンシーケンスのインデックスです。最初のシーケンスは 1 です。
tool_id	ビジョンシーケンスでのツールインデックスです。オプションです。最初のツールは1 です。
instance_id	結果フレームを指定するためのインスタンスインデックスです。オプションです。結果フレームが指定されていない場合、ツールによって返されるすべてのインスタンスのインデックスになります。
result_id	オプションの結果識別子 (ID) です。
index_id	内部使用のための予備値は常に 1 です。
frame_id	指定されたインスタンスに含まれる結果の検索に使用する、フレームのインデックスです。オプションです。

詳細

オプションのパラメータに値が指定されていない場合、デフォルトの値は 1 とみなされます。

特定のツールに対して VRESULT 関数が発行されると、この関数は、そのツールが VRESULT コードをサポートしているかを確認します。指定されたツールがコードをサポートしていない場合、VRESULT 関数は親のツールがコードをサポートしているかを確認するために、対象を親のツールに移します。このプロセスはコードをサポートしているツールが見つかるまで継続します。有効なツールが見つからない場合、無効なビジョン結果というエラーが生成されます。

たとえば、Blob Analyzer ツールに対して Arc Finder ツールを配置する場合が考えられます。このアプリケーションの場合、Blob Analyzer ツールは多くの小塊オブジェクトの位置を特定し、個々の小塊オブジェクトにArc Finder ツールを追加します。Arc Finder インスタンスに関連する小塊の領域を要求すると、VRESULT 関数は Arc Finder ツールはそのコードをサポートしていないと認識するため、対象を親のツール（Blob Analyzer ツール）に移して、指定された円弧の検出結果に関連する小塊オブジェクトを見つけます。そして小塊の検出結果がVRESULT コードをサポートしていることを確認し、データを返します。

一部のビジョンツールは、フレームの情報源とみなされます。Blob Analyzer とLocator ツールは、最も一般的に使用されるフレームの情報源です。これらのツールを実行すると、結果は独立したフレーム、またはフレームのグループとしてマークされます。フレームの情報源に関連するビジョンツールは、対応するフレームに対して検出結果を関連付けます。この場合、frame_id パラメータを使用して、結果を抽出することができます。

上記の例のように、Blob Analyzer ツールに対して Arc Finder ツールを使用して、Blob Analyzerが5 つの異なる検出結果の位置を特定した場合、Arc FinderツールはBlob Analyzer が返す個々の結果に対して5つの別々のArc Finder 操作を実行します。Arc Finder は Blob Analyzer が返す結果のインデックスと一致するフレーム番号と各結果を関連付けます。BlobAnalyzer の4 番目の検出結果と関連付けられたArc Finder の検出結果を要求したい場合、index_id を 1、frame_id を 4 に設定します（結果フレーム 4 の最初のインスタンスを要求）。さらにこの場合では、frame_id を-1 に設定することで、ArcFinder のすべての検出結果にアクセスすることができます。

> **注:** 一部の子のビジョンツールは、各フレーム内に複数の結果を持つこともあり、また全く結果を持たないこともあります。

例

次の例では、Locator ツールが検出した結果から、インスタンスカウントが1310であるインスタンスの数を取得します。

```
instance_count = VRESULT($ip, 1, 2, 1, 1310)
```

関連キーワード

VLOCATION

VPARAMETER

VSTATE

VSTATE

シーケンスの実行状態を返す実数値関数です。

構文

VSTATE (**$ip**, **sequence_id**)

使用上の留意事項

詳細については、*『ロボット用画像センサソフトウェアACE Sightリファレンスマニュアル (SBCE-400)』*を参照してください。

パラメータ

$ip	ビジョンサーバーの IP アドレスです。ビジョンサーバーは、Robot Vision Manager ビジョンソフトウェアが動作している PC であり、標準の IP アドレス形式 (192.168.1.120など) を使用します。
sequence_id	ビジョンシーケンスのインデックスです。最初のシーケンスは 1 です。

詳細

次の表に戻り値の詳細を示します。

値	説明
0	アイドル状態
1	動作中
2	一時停止
3	完了
4	エラー
5	起動中

例

次の例では、ビジョンシーケンスが完了するまで待機します。

```
DO
       WAIT
UNTIL VSTATE($ip, 1) == 3
```

関連キーワード

VLOCATION

VPARAMETER

VRESULT

WINDOW

ベルト相対変換値で記述された位置の、作動しているコンベアベルトの作業範囲の事前定義された境界に対する相対的な位置を示す値を返す、実数値関数です。

構文

WINDOW (transformation, time, mode)

使用上の留意事項

複合変換で参照されるベルト変数は、DEFBELTキーワードを使用してあらかじめ定義する必要があります。

パラメータ

transformation	ベルトに対して相対的に定義される複合変換値です。複合変換は、ベルト変数で始める必要があります。
time	変換が評価されるときに予想時間を指定するオプションの実数式です。この関数の結果は、現在のベルト位置と速度に基づいた予想時間(秒単位)です。このパラメータは、モーションが予想時間内に正しく完了できるかを評価するために使用されます。 time を 0(デフォルト)にした場合、ロケーションのその瞬間の値を評価します。マイナスの値は 0 に変換され、32,768/60 秒より大きい値は、32,768/60 に設定されます。
mode	関数の結果がベルトウィンドウの内側または外側の距離を示すかどうかを指定するオプションの実数式です。

詳細

戻り値は mm 単位の距離であり、以下で説明するような意味となります。

> **注:** 上流と下流の定義は、 BELT.MODEシステムパラメータの値に依存します。

- mode 式の値が 0（デフォルト）以下の場合、戻り値は次のように解釈されます。

0	ロケーションはウィンドウの外側にある。
< 0	ロケーションはウィンドウの内側で、下流のウィンドウ境界線に最も近い位置にある。距離は ABS(value_returned) で表される。
>0	ロケーションはウィンドウの内側で、上流のウィンドウ境界線に最も近い位置にある。境界線までの距離は、戻り値で表される。

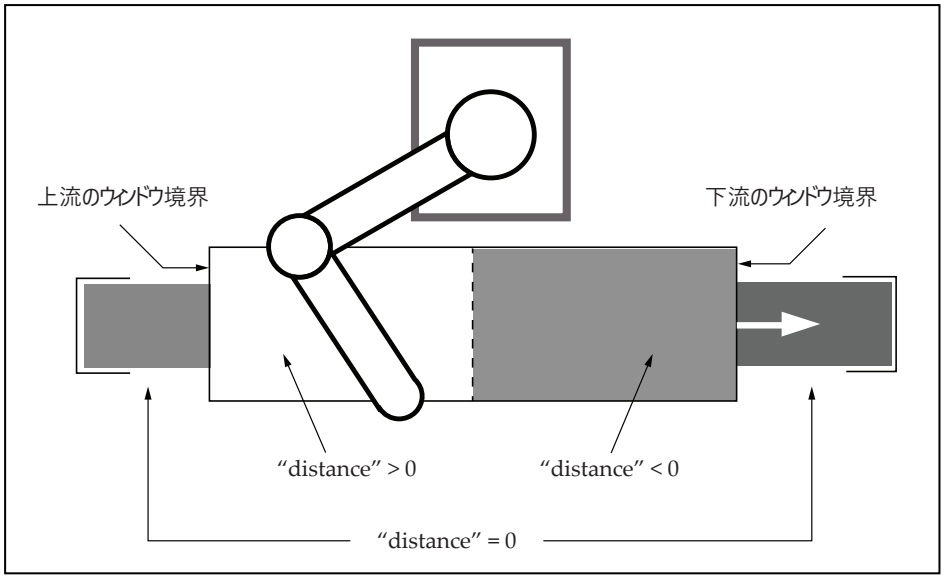


図3-3. mode が 0 以下の場合の WINDOW 関数

- mode 式の値が 0 より大きい場合、戻り値は次のように解釈されます。

0	ロケーションはウィンドウの内側にある。
< 0	ロケーションは上流ウィンドウの境界線より上流にある。距離は ABS(value_returned) で表される。
>0	ロケーションは下流ウィンドウの境界線より下流にある。距離は戻り値で表される。

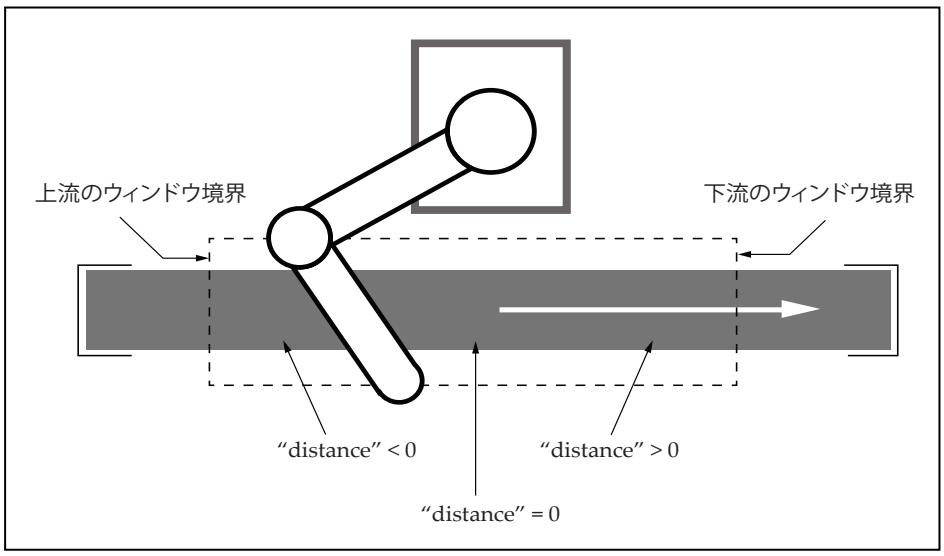


図3-4. mode が 0 より大きい場合の WINDOW 関数

注: WINDOW 関数の戻り値は、mode の値が 0 以下の場合のウィンドウ中央での不連続性を除き、テスト位置が下流に移動するにつれて正の値が大きくなります。

例

次の例では、ロケーションが 2 秒以内に「%belt1」の動作ウィンドウの外側になる場合、変数「distance」を 0 以外の値に設定します。ロケーションがウィンドウ内の場合、変数「distance」は 0 になります。

```
distance = WINDOW(%belt1:pick.up, 2, 1)
```

ロケーションが 2 秒後に %belt1 の動作ウィンドウの外側にある場合、distance は 0 以外の値になります。ロケーションがウィンドウ内の場合、distance は 0 になります。

関連キーワード

BELT(実数値関数)

BELT.MODE

BSTATUS

DEFBELT

SETBELT

3.2 モニタコマンドキーワード

本項の情報を参考にして、モニタコマンドキーワードと eV+ システムを用いたその使用方法を理解します。

ABORT

実行可能プログラムの実行を終了させるモニタコマンドです。

構文

ABORT task

使用上の留意事項

ABORT はディスクまたはネットワーク通信論理ユニット上で DETACH または FCLOSE 操作を強制するものではありません。プログラムで複数のファイルを開き、プログラムの実行を再開しない場合、KILL コマンドですべてのファイルを閉じて、論理ユニットを切り離してください。

パラメータ

task	終了するプログラムタスクを指定するオプションの実数値、変数または式です。整数として解釈されます。

詳細

現在実行しているステップの完了後に、指定したアクティブなプログラムの実行を終了します。このタスクがロボットを制御している場合、ロボットの動作は現在の動作が完了すると終了します。PROCEED コマンドでプログラムの実行を再開できます。

タスク番号が指定されず、システムがデバッグモードでない場合、ABORT コマンドは、タスク番号 0 にアクセスします。

アボート中のタスクをモニタコマンドで開始した場合、次の形式で完了メッセージが表示されます。

```
Program task # stopped at program_name, step step_number date time
```

しかし、タスクが別のタスクから（EXECUTE プログラムコマンドで）開始された場合、完了メッセージは表示されません。

関連キーワード

ABORT（プログラムコマンド）

CYCLE.END（モニタコマンド）

CYCLE.END （プログラムコマンド）

ESTOP(モニタコマンド)

ESTOP （プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

KILL（モニタコマンド）

KILL （プログラムコマンド）

PANIC（モニタコマンド）

PANIC （プログラムコマンド）

PROCEED

RETRY

STATUS

BASE

ロボットに対してワールド基準フレームを平行移動、回転するモニタコマンドです。

構文

BASE X_shift, Y_shift, Z_shift, Z_rotation

使用上の留意事項

BASE コマンドは、SELECT キーワードで選択したロボットに適用されます。プログラムの実行中、このコマンドを使用できます。ロボットが実行中のコマンドで接続されている場合、エラーが発生します。

eV+ システムがロボットを制御するように構成されていない場合、BASE コマンドを実行するとエラーになります。

パラメータ

X_shift	新しい座標系に対して通常のワールド座標系の原点の X 成分を示すオプションの実数値式です。値 0 は、値を指定していないとみなされます。
Y_shift	新しい座標系に対して通常のワールド座標系の原点の Y 成分を示すオプションの実数値式です。値 0 は、値を指定していないとみなされます。
Z_shift	新しい座標系に対して通常のワールド座標系の原点の Z 成分を示すオプションの実数値式です。値 0 は、値を指定していないとみなされます。
Z_rotation	新しい座標系に対して通常のワールド座標系の原点の Z 軸成分の回転を示すオプションの実数値式です。値 0 は、値を指定していないとみなされます。

詳細

初期化すると、ロボットの基準フレームの原点は、運動学的モデルに定義されます。スカラ型ロボットの場合、X-Y 面は、ロボット取付面にあり、X 軸は 0 に等しいジョイント 1 で定義された方向になり、Z 軸は、ジョイント 1 の軸と一致します。ロボットの基準フレームのデフォルトロケーションについては、ロボットユーザーマニュアルを参照してください。

BASE コマンドは、上記指定のように基準フレームをオフセットし、回転します。ロケーションがアプリケーション用に定義された後、ロボットを移動する場合に役立ちます。ロボットがロボット基準フレームに対する変換で定義されているロケーション（dX、dY、dZ で示され、Z_degrees で回転された位置）にロボットを移動するときに発生する、ロケーション誤差を補正するために、このコマンドを使用できます。以前に定義したロケーションに対するモーションが、まだ適切に機能することを保証します。

さらに、BASE コマンドを使用して、X、Y 座標軸を再調整することで、SHIFT 関数が目的の非標準の方向へ変位することができます。

注: BASE コマンドは、精度位置として定義されているロケーションには影響しません。BASE コマンドのパラメータには、通常のロケーションに対するロボットの変位を記述します。BASE 関数を LISTL コマンドと共に使用することで、（「LISTL BASE」ステートメントを使用して）モニタウィンドウに現在の BASE 設定を表示することができます

例

次の例では、ロボットが正の X 方向に「xbase」mm、負の Z 方向に 50.5 mmシフトされ、Z軸に対して 30 度回転されているため、ワールド基準フレームを再定義します。

```
BASE xbase,, -50.5, 30
```

次の例では、すべてのロケーションを公称位置から負のX方向に 100 mm、正の Z 方向に 50 mm 効果的にシフトするために、ワールド基準フレームを再定義します。このステートメントの引数は、ロボットに対するロボットの基準フレームの移動量を示し、ロボットに対するロケーションに逆の効果を持ちます。

```
BASE 100,, -50
```

関連キーワード

BASE（変換関数）

BASE (プログラムコマンド)

SELECT（モニタコマンド）

SELECT (プログラムコマンド)

SELECT（実数値関数）

BITS

値に基づきデジタル信号グループをセット、またはクリアするモニタコマンドです。

構文

BITS first_sig, num_sigs **= value**

使用上の留意事項

外部デジタル出力信号、または内部ソフトウェア信号を参照できます。指定の信号には入力用に設定された信号を含めないでください。

一度に 32 以上の信号を設定することはできません。

グループ内のすべての信号がシステムでの使用のために設定されているのであれば最大 32 の信号の任意のグループを設定することができます。

パラメータ

first_sig	影響を受ける信号の最小数を定義する実数式です。
num_sigs	影響を受ける信号の数を定義するオプションの実数値式です。指定しない場合、値 1 が仮定されます。有効な最大値は 32 です。
value	指定された信号に設定される値を定義する実数式です。この値の 2 進表現が「num_sigs」以上のビットを持つ場合、最下位の「num_sigs」信号のみ影響を受けます。

詳細

等号の右辺の値に基づき、1 つ、または複数の外部出力信号やソフトウェア信号をセット、またはクリアします。この操作は、値を整数値に四捨五入し、整数値のバイナリ表現の各ビットに基づき、信号数をセット、またはクリアします。

例

次の例では、外部出力信号 1～4（4 ビット）を BCD 桁 7 のバイナリ表現に設定します。

```
BITS 1,4 = BCD(7)
```

次の例では、外部出力信号 9～16（8 ビット）を現在の監視速度設定のバイナリ表現に設定します。現在、モニタ速度が 50%（バイナリ値：110010）に設定されている場合、コマンドの後、信号 9～16 は次に示すようにセットされます。

```
BITS 9,8 = SPEED(1)

 9 → 0 (off)      13 → 1 (ON)
10 → 1 (on)       14 → 1 (ON)
11 → 0 (off)      15 → 0 (OFF)
```

```
12 → 0 (off)      16 → 0 (OFF)
```

次の例では、外部出力信号 1～8（8 ビット）を定数 255（11111111）に設定します。信号 1～8 によってすべて ON になります。

```
BITS 1,8 = 255
```

関連キーワード

BITS (プログラムコマンド)

BITS(実数値関数)

RESET

SIG(実数値関数)

SIG.INS(実数値関数)

SIGNAL（モニタコマンド）

SIGNAL (プログラムコマンド)

CALIBRATE

ロボットのポジショニングシステムを初期化するモニタコマンドです。

構文

CALIBRATE mode, robot

使用上の留意事項

このキーワードは一般的にモードを指定しないで発行されます。

DRY.RUN システムスイッチが有効な場合、CALIBRATE オペレーションには効果がありません。

ロボットを動かす場合、CALIBRATE プログラムコマンドまたはモニタコマンドのキーワードは、システム電源をオンにする、またはeV+システムをディスクから起動するたびに実行される必要があります。起動を有効にしたときにCALIBRATEを実行するオプションがロボットにある場合、ロボットは起動時にCALIBRATIONオペレーションを実行します。このオプションが有効ではない（またはそのロボットでサポートされていない）場合、ハイパワーが有効になった後で、このキーワードを実行しなければなりません。

起動時にCALIBRATE を実行するオプションの有効化に関する詳細については『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。

ロボットがキャリブレーションされていない場合、CALIBRATEプログラムコマンドまたはモニタコマンドが実行されるまでロボットを手動制御ペンダントを使用して、またはプログラム制御の下で動かすことはできません。

複数のロボットがシステムコントローラに接続していて、「robot」パラメータが省略されている（または 0 である）場合、このキーワード命令は、ROBOTシステムスイッチで無効になっている以外のすべてのロボットを順番にキャリブレーションしようとします。プログラム制御で移動できる前に、すべての有効なロボットをキャリブレーションする必要があります。

> **注:** オプションのフロントパネル、またはリモートフロントパネルが装備されている場合、このコマンドを処理するために、コントローラーのキースイッチを自動モードに設定する必要があります。

CALIBRATEキーワードは、そのそれぞれのロボットタイプで異なるオペレーションをする可能性があります。CALIBRATEキーワードは通常、全てのロボットのジョイントを動かします。CALIBRATEキーワードを発行できる位置は制御するロボットのタイプによって異なります。唯一の制限は、キャリブレーションプロセス中に範囲外に移動しないように、ロボットを作業範囲の限界から十分離れた位置にする必要があることです。

パラメータ

mode　キャリブレーションの実行部分を示す実数式です。

モード値	解釈
0（または省略）	システムが制御するすべてのロボットに対して、通常のキャリブレーションを実行します次のオペレーションが実行されます。 • メインのキャリブレーションプログラムがまだメモリに存在しない場合、ロードします。 • ロボット固有のルーチンをロードし、キャリブレーションを実行し、ルーチンを削除する、メインのキャリブレーションプログラムをロード、実行、削除し、フラグセットをモニタします。 • ロードされている場合は、メインのキャリブレーションプログラムを削除します。プログラムがプロセスの最後に削除されているかにかかわらず、ロボット固有のルーチンは、メインのプログラムから削除されることに注意してください。
1	メインのキャリブレーションプログラムがまだメモリに存在しない場合、ロード、実行し、フラグセットをモニタします。これにより、メインプログラムが適用可能なロボット固有のルーチンにロードされます。キャリブレーションプロセスは実際には実行されないことに注意してください。
2	メインのキャリブレーションプログラム（メモリに存在する必要がある）を実行し、実行およびモニタフラグをセットします。これにより、システムのロボットをキャリブレーションし、すべてのキャリブレーションプログラムがメモリに残されます。
3	メインのキャリブレーションプログラム（メモリに存在する必要がある）を実行し、削除し、フラグセットをモニタします。これによりロボット固有のすべてのキャリブレーションルーチンがメモリから削除されます。次に、メインキャリブレーションプログラムをメモリから削除します。キャリブレーションプロセスは実際には実行されないことに注意してください。

詳細

キャリブレーションオペレーションが開始されると、eV+システムは、ロボットがキャリブレーションされず、ロボット制御プログラムの実行を許可しないかのよう

に、振舞いますロボットがキャリブレーションされていない場合、ペンダントのCOMP モードランプは点灯しません。

システムの電源スイッチがオフになるとロボットはキャリブレーションされていない状態になります。安全上の処置として、特定のサーボエラーが発生した場合もロボットはキャリブレーションされません。

メインのキャリブレーションプログラムの読み込みが含まれる場合、CALIBRATEオペレーションによってディスクファイル「cal_util.v2」が読み込まれます。便宜上、ロード動作では、次に示す順番で次のディレクトリのファイルを検索します。

1. デフォルトのカレントディレクトリ
2. eV+システムを起動したローカルディスクの¥CALIB¥（TFTPデバイスからシステムを起動した場合はスキップします）
3. DISK>D:¥CALIB¥

キャリブレーションプログラムは、タスク 0 で実行されます。タスク0がすでにアクティブの場合、CALIBRATEはオペレーションは失敗します。

関連キーワード

CALIBRATE（プログラムコマンド）

NOT.CALIBRATED

CD

ディスクアクセス用のデフォルトパスを表示、または変更するモニタコマンドです。

構文

CD path

パラメータ

[path]	対象のディスクディレクトリパスを指定するオプションの文字列です。通常、このパラメータには、ディレクトリ名とバックスラッシュ文字を含めます。パス指定の末尾にバックスラッシュ文字がない場合、eV+ システムはバックスラッシュ文字を追加します。 パスのパラメータを省略した場合、現在のディレクトリパスが表示されます。

詳細

このコマンドは、次に示す入力と同じ意味になります。

```
default disk = path
```

ディスクディレクトリのパス指定については、DEFAULT コマンドを参照してください。

例

デフォルトのパスを表示するには、次のように入力します。

```
cd
```

デフォルトのパスを C:¥TEST¥JOBS¥ に変更するには、次のように入力します。この例で示すように、デフォルトのパスを変更するには、あらかじめパスが存在している必要があります。末尾のバックスラッシュ文字は省略できます。

```
cd c:\test\jobs\
```

ディレクトリパスを 1 つ上に移動する場合は、次のように入力します。

```
cd ..
```

この例では、次のように入力して、1 つ下に移動しています（c:¥TEST¥JOBS from c:¥TESTに戻る）。

```
cd jobs
```

関連キーワード

DEFAULT

COMMANDS

モニタコマンドプログラム の処理を開始するモニタコマンドです。

構文

COMMANDS program

使用上の留意事項

プログラムタスク 0 の実行中に COMMANDS コマンドを発行できますが、プログラムが完了する、または CTRL+C が押されて COMMANDS コマンドがアボートされるまで、システムキーボードから入力できません。

モニタコマンドプログラム のすべてのコマンド行は「MC」から始まる必要があります。

このコマンドを処理するために、フロントパネルのキースイッチを自動モードに設定する必要があります。

パラメータ

program	処理する予定の モニタコマンドプログラム の名称です。

詳細

COMMANDS は指定された モニタコマンドプログラム の処理を開始します。このとき モニタコマンドプログラム はメモリに既に存在している必要があります。次のいずれかが発生するまで、プログラムの処理は続行します。

- モニタコマンドプログラム の最後に達した。
- システムキーボードの CTRL+C シーケンスが押された。
- プログラム内に COMMANDS コマンドを見つけた。
- エラーが発生した。

COMMANDS コマンドが モニタコマンドプログラム に含まれている場合、新しい モニタコマンドプログラム が呼び出され、最初の モニタコマンドプログラム は無視されます。モニタコマンドプログラム は 互いにリンクできますが、実行可能なプログラムに関してはリターンパスを作成することはできません。モニタコマンドプログラム によって（EXECUTE コマンドを使用して）呼び出された実行可能プログラムは、eV+ 言語で使用可能なすべての制御キーワードを使用することができます。

自動スタート機能は、コントローラーの電源が ON で、eV+ システムがディスクからロードされる場合に、COMMANDS コマンドを自動的に発行する手段を提供します。詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、「setup」という名前の モニタコマンドプログラム の処理を開始します。

```
commands setup
```

関連キーワード

CYCLE.END

DO

COPY

既存のプログラムのコピーとして新しいプログラムを作成するモニタコマンドです。

構文

COPY new_program = old_program

使用上の留意事項

COPY コマンドを使用して、実行しているプログラムをコピーできます。COPY コマンドは、システムメモリ内にあるプログラムだけをコピーし、ディスクファイルはコピーしません（ディスクファイルをコピーする場合は、FCOPY コマンドを参照してください）。

パラメータ

new_program	作成するプログラムに指定するプログラム名です。
old_program	コピー対象のプログラム名です。

詳細

既存のプログラムの正確なコピーとして新しいプログラムを作成します。プログラムの新しいコピーは、GLOBAL プログラムモジュール内に配置されます。COPY 命令の後、新しいプログラムまたは古いプログラムを必要に応じて編集できます。

> **注:** すでにシステムメモリ内に指定された新しい名前と同じ名前のプログラムが存在する場合、COPY 命令は実行されず、エラーメッセージが表示されます。この場合、コピーする前に、まず、競合するプログラムを削除、または名前を変更するか、別の名前を使用する必要があります。

例

次の例では、プログラム「test」のコピーを作成し、作成された新しいコピーにプログラム名「test.cpy」を割り当てています。

```
COPY test.cpy = test
```

関連キーワード

FCOPY（モニタコマンド）

FCOPY （プログラムコマンド）

RENAME

CYCLE.END

次回、STOP プログラム操作、または同等の操作が実行されたとき、指定された実行可能なプログラムを終了します。プログラムの実行が完了するまで、コマンドプログラムの処理を一時停止します。

構文

CYCLE.END task, stop_flag

使用上の留意事項

指定のプログラムタスクがアクティブではない場合、CYCLE.END コマンドは効果を持ちません。

CYCLE.END コマンドは、指定のタスクの実行が完了するまで、すべてのキーボード入力をブロックします。CTRL+C でキーボードのブロックが解除されます。この場合、「stop_flag」が TRUE であれば CYCLE.END コマンドはプログラムを終了します。

パラメータ

task	監視または終了するプログラムタスクを指定する、整数と解釈されるオプションの実数値、変数または式です。 タスク番号が指定されていない場合、CYCLE.END コマンドはタスク番号 0 にアクセスします。
stop_flag	論理値(TRUE または FALSE)として解釈されるオプションの実数値、変数、式です。パラメータが省略されている、または 0 の場合、指定のタスクは停止しませんが、CYCLE.END の他のすべての効果は維持されます。パラメータが 0 以外の場合、現在のサイクルの最後に指定のタスクは停止します。

詳細

「stop_flag」パラメータが TRUE の場合、指定されたプログラムタスクは停止し、実行予定のプログラムサイクル数にかかわらず、次回、STOP 操作または同等の操作を実行します。

> **注:** CYCLE.END は、連続した内部ループを持つプログラムを終了しません。このようなプログラムは ABORT 操作で終了させる必要があります。

「stop_flag」パラメータにかかわらず、このコマンドは、プログラムが実際に終了するまで待機します。内部でループするプログラムを終了した場合、現在の実行サイクルは終了しないため、CYCLE.END コマンドは永遠に待機することになります。

プログラムの終了を待機している CYCLE.END コマンドからシステムキーボードを解放するには、CTRL+C を押します。

例

次のコマンドプログラムの例では、コマンドプログラムが、次のコマンドを発行する前に、あるプログラムの終了をどのように待機するかを示しています。

```
MC EXECUTE 1 setup
MC CYCLE.END 1
MC EXECUTE 1 main.1
MC EXECUTE main
MC CYCLE.END
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

HALT

KILL（モニタコマンド）

KILL （プログラムコマンド）

PROCEED

RETRY

STATUS（モニタコマンド）

STATUS（実数値関数）

STOP

DEFAULT

アクセスされる eV+ ディスク論理デバイスと物理デバイス間のデフォルトの関係を定義するモニタコマンドです。また、現在のデフォルト値を表示します。

構文

DEFAULT DISK = physical_device>unit:directory_path

使用上の留意事項

DEFAULTコマンドの代わりに、よりシンプルな CD コマンドを使用できます。

パラメータ

physical_device	ディスク論理デバイスと関連する物理デバイスのオプション名です。許容可能なデバイス名は「DISK」（かっこ不要）を指定する必要があります。また、「DI」と短縮できます。 以前使用していたデフォルトの物理デバイスを変更していない場合、物理デバイスを省略するときは、文字「>」を省略する必要があります。 物理デバイスを指定するとき、ユニットを指定しない場合、デフォルトのユニットは取り消されます。
unit	目的のデフォルトのユニットを指定するオプションの文字列です。 • DISK 物理デバイス（コントローラーベースのディスクなど）の場合、このパラメータは、対象のディスクドライブ名です。 • TFTP デバイスの場合、unit は、アクセスされる TFTP サーバーの名前または IP アドレス（ドット付きの 10 進フォーマット）です。 ユニットを指定しない場合、文字「:」を入力するしないでください。この場合、以前のデフォルトのユニットは変更されません（上記以外）。
directory_path	対象のディレクトリパスを指定するオプションの文字列です。通常、このパラメータには、ファイル名とバックスラッシュ文字が含まれます。ディスクデバイスの場合、パス指定の最後にバックスラッシュ文字がない場合、eV+ は、バックスラッシュ文字を追加します。 現在または指定された物理デバイスがディスクデバイスでない場合、先頭の「¥」は、最上位ディレクトリにあるディレクトリパスの開始を指定します。このパスは、 現在定義されているデフォル

トのパス（絶対パス）を置き換えます。先頭の「¥」がない場合、このパスが現在のデフォルトのパス（相対パス）に付加されることを意味します。

注: 「unit」と「directory_path」パラメータを省略した場合、デフォルトのユニットとディレクトリパスは、取り消されます。パスのパラメータを省略した場合、現在のディレクトリパスが表示されます。

「unit」パラメータに現在のデフォルトとは異なるユニットを指定した場合、指定されたディレクトリパスは、必ず最上位レベルのディレクトリから始まります。

特殊な場合として、他のシステムを参照する場合に役立つ非標準のディレクトリ（「...」や「/.../」など）は許可されます。

詳細

次の説明では、「ディレクトリ仕様」という用語は、物理デバイス、ディスクユニット、ディレクトリパスの組み合わせについて言及するために、使用されます。

ディスク関連の eV+ 命令（FDIRECTORY、LOAD、STORE、FOPEN_など）が処理されるとき、eV+ システムは自動的に現在の「デフォルト」ディレクトリ仕様とコマンドや関数に提供されるディレクトリ仕様を組み合わせます。このような状況で使用されるディレクトリ仕様を設定するために、DEFAULT コマンドを使用できます（次の例を参照）。

注: DEFAULT コマンドは指定のデフォルトデバイス、ユニット、ディレクトリが実際にアクセスできるかは確認しません。

現在のデフォルトのディレクトリ仕様をモニタウィンドウに表示する場合、パラメータを指定せずに DEFAULT コマンドを入力できます。

eV+ システムがディスクから起動された場合、システムディスクに保存されている設定に従って、初期のデフォルトのディスク関係は設定されます。

注: OMRONは、デフォルトディスクユニットを「DISK>D」に設定した eV+ システム起動ディスクを提供しています。デフォルトのユニットとディレクトリパスは Sysmac Studio で変更できます。詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』*を参照してください。

DEFAULT コマンドが処理された後、続くディスク操作（およびDEFAULT コマンド）は、必要に応じて新しいデフォルトディレクトリ仕様を使用します。次のルールは、デフォルトの仕様とコマンドや関数に含まれるディレクトリ仕様の組み合わせから生じるディレクトリ仕様を決定します。

1. ユニットが指定されていない場合、現在のデフォルトのユニットが使用されます。指定のパスがバックスラッシュ（¥）で始まらない場合、指定されたディレクトリパスはデフォルトのディレクトリパスに付加されます。それ以外の場合は、デフォルトのディレクトリパスは無視されます。しかし、ユニットとディレクトリパスの両方が省略されている場合、DEFAULT コマンドは、デフォルトのユニットとディレクトリパスの両方を取り消します。
2. 指定されたユニットが現在のデフォルトのユニットと同じで、指定のパスがバックスラッシュ（¥）で始まらない場合、指定のディレクトリパス（ある場合）はデフォルトのディレクトリパスに付加されます。それ以外の場合は、デフォルトのディレクトリパスは無視されます。
3. 指定されたユニットが現在のデフォルトのユニットと異なる場合、指定されたディレクトリパスは、必ず指定されたユニットの最上位ディレクトリから始まります。デフォルトのディレクトリパスは無視されます。

ディスクディレクトリの詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、デフォルトのディレクトリ仕様を表示、または設定するために、DEFAULT コマンドを使用する方法について示しています。

DEFAULT	ディスクの現在のデフォルト値を表示します。
DEFAULT = A:	デフォルトのディスクドライブをユニット「A」に変更します。
DEFAULT = DI>D:¥ROB1	デフォルトのサブディレクトリを（ローカル）物理デバイス「DISK」のディスクユニット「D」のサブディレクトリ「ROB1」に変更します。
DEFAULT = TEST¥DEMO	上記のコマンドの後に使用した場合、デフォルトのディレクトリパスを「¥ROB1¥TEST¥DEMO¥」（ディスクユニット「D」）に変更します。

次の例では、デフォルトのディレクトリ仕様がどのように特定の状況に適用されるかを示しています。いずれの場合でも、現在のデフォルトのディレクトリ仕様は「DISK>D:¥ROB1¥」ととします。

次の例では、デフォルトのディレクトリ仕様全体が使用されます。

- ユーザーが入力するコマンド：FDIRECTORY *.V2
- 処理されるコマンド：FDIRECTORY DISK>D:¥ROB1¥*.V2

次の例では、デバイス、ディスク、ルートディレクトリはデフォルトのディレクトリ仕様から取得されます。

- ユーザーが入力するコマンド：FDIRECTORY JOB1¥FEED*.*
- 処理されるコマンド：FDIRECTORY DISK>D:¥ROB1¥JOB1¥FEED*.*

次の例では、デバイスとディスクはデフォルトのディレクトリ仕様から取得され、指定のパスがバックスラッシュで始まっているため、デフォルトのルートディレクトリは無視されます。

- ユーザーが入力するコマンド：FDIRECTORY ¥JOB1¥FEED*.*
- 処理されるコマンド：FDIRECTORY DISK>D:¥JOB1¥FEED*.*

関連キーワード

CD

$DEFAULT

FDIRECTORY

FSET

LOAD

STORE

FOPEN

FOPENA

FOPEND

FOPENR

FOPENW

DELETE

システムメモリから指定されたプログラムを削除するモニタコマンドです。

構文

DELETE program, ..., program

使用上の留意事項

STATUS 関数が示すように、実行中のプログラム、または実行スタックに存在するプログラムは削除できません。

DELETE は、ディスクファイルを削除しませんが、システムメモリからプログラムを削除します。削除したプログラムが事前にディスクに保存されていれば、そのプログラムを LOAD コマンドを使用して再読み込みすることができます。FDELETE コマンドは記憶ディスクからディスクファイルを削除します。

> **追加情報:** プログラムをモジュールにグループ化することは、優れたプログラミング習慣です。グループ化すれば、DELETE コマンドの代わりに DELETEM コマンドを使用できます。

パラメータ

program 削除対象のプログラム名です。

詳細

DELETE コマンドは名前付きプログラムを完全に削除します。このコマンドはプログラム自体を削除し（DELETEP コマンドと同様）、また、名前付きプログラムが排他的に使用する次の項目すべても削除します。

-

> **注:** STATUS 関数が示すように、プログラムとその参照項目がアクティブなプログラム実行スタックにある場合、これらは削除されません。スタックにある参照対象のプログラムを削除する前に、適切なプログラム実行スタックをクリアするには、KILL コマンドを使用する必要があります。

例

次の例では、「assembly」という名称のプログラムを削除します。

```
DELETE assembly
```

関連キーワード

DELETEL

DELETEM

DELETEP

DELETER

DELETES

FDELETE（モニタコマンド）

FDELETE（プログラムコマンド）

DELETEL

システムメモリから名前付きロケーション変数を削除するモニタコマンドです。

構文

DELETEL @task:program **loc_variable**, ..., loc_variable

使用上の留意事項

配列要素を指定した場合、要素は削除されます。

配列名を明示的インデックスなしに指定した場合（DELETEL a[ ]など）、配列全体が削除されます。

多次元配列で 1 つ以上の最も右のインデックスを省略した場合、これらのインデックスで定義されたすべての要素が削除されます。次の例では、この削除について示しています。

この例では、要素 a[3,2,0]から a[3,2,last]を削除します。

```
DELETEL a[3,2,]
```

この例では、要素 a[3,i,j]のすべての i と j の要素を削除します。

```
DELETEL a[3,,]
```

この例では、配列全体を削除します。

```
DELETEL a[,,]
```

パラメータ

@task:program	これらのオプションのパラメータは、ロケーション変数のコンテキストを指定します。ロケーション変数は、指定のコンテキストから参照されるかのように扱われます。コンテキストが指定されていない場合、ロケーション変数はグローバル変数とみなされます。変数のコンテキストの詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
loc_variable	削除対象のロケーション変数名です。

詳細

任意の番号のロケーション変数を削除します（変換および/または精度ポイント）。これ以上必要ではないロケーション変数が使用しているメモリの保存スペースを回復するために、この命令を使用できます。

ロケーション変数は削除された後、どの eV+ 命令や関数を使用しても参照できません。削除された変数を参照しようとすると、これまでその変数が定義されていないかのように、エラーメッセージが表示されます。

例

次の例では、変換「pick」と精度ポイント「#park」をメモリから削除します。

```
DELETEL pick, #park
```

次の例では、プログラム「main」のローカル変数である変換変数「temp」を削除します。

```
DELETEL @main temp
```

関連キーワード

DELETE

DELETER

DELETES

FDELETE

DELETEM

システムメモリから指定したプログラムモジュールを削除するモニタコマンドです。

構文

DELETEM module

使用上の留意事項

モジュールのプログラムがインターロックされている場合、モジュールは削除されません（以下参照）。

モジュール内のプログラムがメモリ内の他のプログラムから参照されていても、プログラムは削除されます。

DELETEM は、システムメモリからプログラムモジュールを削除します。関連するディスクファイルは削除されません。

パラメータ

module　削除対象のプログラムモジュール名です。

詳細

プログラムモジュールとモジュールのプログラムすべてがシステムメモリから削除されます。必要ではないプログラムが使用しているメモリスペースを回復するために、この命令を使用できます。

DELETEコマンドとは異なり、DELETEM は、サブルーチンも指定のモジュールに含まれている場合を除き、削除対象のプログラムが参照しているサブルーチンは削除しません。削除対象のプログラムが参照している変数は削除されません。

モジュール内のプログラムがインターロックされている場合（STATUS実数値関数を参照）、これらのプログラムとモジュールは削除されません。実行スタックで参照されているプログラムを含むモジュールを削除する前に、適切なプログラム実行スタックをクリアするためにKILLコマンドを使用する必要があります。

プログラムモジュールの詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、「main.package」という名称のプログラムモジュールとこのモジュール内のすべてのプログラムを削除します（どのプログラムもインターロックされていないと仮定）。

```
DELETEM main.package
```

関連キーワード

DELETE

DELETEP

FDELETE

MDIRECTORY

MODULE

STOREM

DELETEP

システムメモリから指定したプログラムを削除するモニタコマンドです。

構文

DELETEP program, ..., program

使用上の留意事項

STATUS関数で示されているように、実行中またはアクティブな実行スタック上にあるプログラムは削除できません。

削除対象のプログラムが参照しているサブルーチンと変数は削除されません。詳細については、DELETE を参照してください。

DELETEP はメモリからプログラムを削除します。関連するディスクファイルは削除されません。

> **追加情報:** プログラムをモジュールにグループ化することは、優れたプログラミング習慣です。グループ化すれば、DELETEM コマンドの代わりにDELETEM コマンドを使用できます。

パラメータ

program 削除対象のプログラム名です。

詳細

任意の数のプログラムをシステムメモリから削除します。必要ではないプログラムが使用しているメモリスペースを回復するために、この命令を使用できます。

DELETE コマンドとは異なり、DELETEP は、指定したプログラムが参照しているサブルーチンまたは変数を削除しません。

> **注:** STATUS関数で示されているように、アクティブなプログラム実行スタック上にあるプログラムは削除されません。スタックで参照されているプログラムを削除する前に、適切なプログラム実行スタックをクリアするために KILL コマンドを使用する必要があります。

例

次の例では、名前付きプログラム「test.one」を削除します。

```
DELETEP test.one
```

関連キーワード

DELETE

DELETEM

FDELETE

KILL

STATUS

DELETER

システムメモリから名前付き実数値変数を削除するモニタコマンドです。

構文

DELETER @task:program **real_variable**, ..., real_variable

使用上の留意事項

DELETER は外部変数を削除しません。

パラメータ

@task:program	これらのオプションのパラメータは、実数値変数のコンテキストを指定します。実数値変数は、指定のコンテキストから参照されるかのように扱われます。コンテキストが指定されていない場合、実数値変数はグローバル変数とみなされます。変数コンテキストについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
real_variable	削除対象の実数値変数名です。

詳細

任意の番号の実数値変数を削除します。これ以上必要ではない実数値変数が使用しているメモリの保存スペースを回復するために、この命令を使用できます。

実数値変数は削除されてしまうと、いかなる eV+オペレーションや関数を使用しても参照することができません。削除された変数を参照しようとすると、これまでその変数が定義されていないかのように、エラーメッセージが表示されます。

配列要素が指定されている場合、その要素は削除されます。配列名を明示的インデックスなしに指定した場合（DELETER a[ ]など）、配列全体が削除されます。

多次元配列で最も右のインデックスを 1 つ以上省略した場合、これらのインデックスで定義されたすべての要素が削除されます。次の例では、この削除について示しています。

この例では、要素 a[3,2,0]から a[3,2,last]を削除します。

```
DELETER a[3,2,]
```

この例では、要素 a[3, i, j]のすべての i と j の要素を削除します。

```
DELETER a[3, , ]
```

この例では、配列全体を削除します。

```
DELETER a[, , ]
```

例

次の例では、実数値変数「count」と実数値配列の要素「x[2]」を削除します。

```
DELETER count, x[2]
```

次の例では、プログラム「insert」のローカル変数である実数値配列「part」を削除します。

```
DELETER @insert part[]
```

関連キーワード

DELETE

DELETEL

DELETES

FDELETE

DELETES

システムメモリから指定した文字列変数を削除するモニタコマンドです。

構文

DELETES @task:program **string_var**, ..., string_var

パラメータ

@task:program	これらのオプションのパラメータは、ロケーション変数のコンテキストを指定します。ロケーション変数は、指定のコンテキストから参照されるかのように扱われます。コンテキストが指定されていない場合、ロケーション変数はグローバル変数とみなされます。変数のコンテキストの詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
string_var	削除対象の文字列変数名です。

詳細

任意の番号の文字列変数を削除します。これ以上必要ではない文字列変数が使用しているメモリの保存スペースを回復するために、この命令を使用できます。

文字列変数は削除された後、どの eV+ 命令や関数を使用しても参照できません。削除された変数を参照しようとすると、これまでその変数が定義されていないかのように、エラーメッセージが表示されます。

配列要素が指定されている場合、その要素は削除されます。配列名を明示的インデックスなしに指定した場合（DELETES $a[ ]など）、配列全体が削除されます。

多次元配列で最も右のインデックスを 1 つ以上省略した場合、これらのインデックスで定義されたすべての要素が削除されます。次の例では、この削除について示しています。

（例えば、コマンド「DELETES $a[3,2,]」は、要素「$a[3,2,0]」と「$a[3,2,last]」を削除します。コマンド「DELETES $a[3, ,]」は、要素 a[3, i, j]のすべての i と j の要素を削除します。コマンド「DELETES $a[, ,]」は、配列全体を削除します）。

この例では、要素 $a[3,2,0]から $a[3,2,last]を削除します。

```
DELETES $a[3,2,]
```

この例では、要素 $a[3, i, j]のすべての i と j の要素を削除します。

```
DELETES $a[3, , ]
```

この例では、配列全体を削除します。

```
DELETES $a[, , ]
```

例

この例では、文字列変数「$input」を削除します。

```
DELETES $input
```

次の例では、プログラム「menu」のローカル変数である文字列変数「$response」を削除します。

```
DELETES @menu $response
```

関連キーワード

DELETE

DELETEL

DELETER

FDELETE

DIRECTORY

システムメモリ内の一部またはすべてのプログラムの名前を表示するモニタコマンドです。

構文

DIRECTORY /switch wildcard_spec

使用上の留意事項

このコマンドは、システムメモリ内のプログラムを一覧表示します。ディスクドライブのファイルは一覧表示しません（FDIRECTORY コマンドを参照）。

パラメータ

switch　次の switch 値を持つオプションの修飾子です。

Switchの値	目的
/S	保護されているプログラムは表示しない
/?	非実行プログラムのみ表示する
/M	修正済みプログラムのみ表示する

wildcard_spec　「?」または「*」文字のワイルドカードを含むことができるオプションの文字列です。どちらのワイルドカードも同様に機能し、0、1、または複数の文字にマッチできます。

詳細

各プログラムのリストに次の情報が表示されることがあります。

- プログラムが実行できない場合、左側にクエスチョンマーク「?」が表示されます。通常、プログラムにプログラミングエラーが含まれていることを示します。
- どのプログラム名についても、メモリ内のコピーが、最後にロードまたは保存されてから修正されている場合、プログラム名の前に「M」が表示されます。
- プログラムがアクセス制限されている場合、制限の種類を示すコード文字が左側に表示されます。
  - プログラムが保護されている場合、「P」が表示され、表示、編集、または保存できないことを示します。
  - プログラムが読み取り専用の場合、「R」が表示され、編集、または保存できないことを示します。
- プログラム名が表示されます。
- プログラムが保護されていない場合、プログラムの.PROGRAM コマンド全

体が表示されます。.PROGRAM ラインにコメントがある場合、プログラムが必要とするパラメータが表示されます。

- プログラムが保護されている場合、.PROGRAM コマンドはプログラム名の後に切り詰められます。

例

次の例では、保護されているプログラムを含むコントローラメモリ内にあるすべての実行不可能なプログラムを表示します。

```
DIRECTORY /?
```

関連キーワード

FDIRECTORY

MDIRECTORY

DISABLE

1 つ以上のシステムスイッチを OFF するモニタコマンドです。

構文

DISABLE switch, ..., switch

使用上の留意事項

プログラムの実行中、DISABLE コマンドを使用できます。

指定のスイッチがインデックスの修飾子を受け付け、インデックスが 0 または括弧の有無を問わず省略されている場合、スイッチ配列のすべての要素は無効になります。

パラメータ

switch OFF するシステムスイッチ名です。

その名称はそのスイッチを一意に特定する最小長さに短縮することができます。例えば、文字「ME」から始まる名前のスイッチは他にはないため、MESSAGES スイッチを「ME」と呼ぶことができます。

詳細

システムスイッチはeV+ システムのさまざまな動作を制御します。使用可能なシステムスイッチとその機能については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

スイッチが OFF の場合、スイッチが制御する機能は、機能しない、または使用できなくなります。ENABLE コマンドまたはプログラムコマンドでスイッチを ON にすると、関連する機能が機能する、または使用できます。

> **注:** システムスイッチは、すべてのプログラムタスクで共有されます。1 つ以上のタスクがスイッチを正しくに設定できない可能性があるため、複数のタスクでスイッチを無効、有効にする場合には、よく検討する必要があります。
>
> DRY.RUN スイッチを無効にしても、タスク0 に対して次の EXECUTE モニタコマンドまたはプログラムコマンドが処理される、ロボットに対するATTACH プログラムコマンドが実行される、またはCALIBRATE モニタコマンドまたはプログラムコマンドが処理されるまでは、影響を与えません。

SWITCH モニタコマンドまたは SWITCH 関数を使用して、常にスイッチの状態を判断できます。DISABLE プログラムコマンドを使用した場合と同様に、SWITCH プログラムコマンドを使用してスイッチを無効にすることができます。

例

次の例では、CP（連続パス）スイッチを OFF にします。

```
DISABLE CP
```

関連キーワード

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

SWITCH(モニタコマンド)

SWITCH

SWITCH（実数値関数）

DO

実行可能なプログラム、または指定されたタスクやプログラムコンテキストの次のステップであるかのようにキーワードを実行するモニタコマンドです。

構文

DO @task:program instruction

使用上の留意事項

指定されたプログラムタスクは現在実行中であってはいけません。

DO コマンドで使用できるキーワードについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

パラメータ

program instruction

実行されるオプションの eV+ キーワードです。キーワードが指定されていない場合、（指定または想定されるコンテキストにかかわらず）DO コマンドで実行された最後のキーワードが繰り返されます。



警告: DO コマンドの入力時にプログラム命令を指定しない場合、前回の DO 操作が再度実行されるため、想定外のロボットモーションが発生する可能性があります。

task

操作を実行するプログラムタスクを指定するオプションの整数値です。

操作で参照される変数のコンテキストを判断するためにも、このパラメータが使用されます。

「task」を省略した場合、コロン（「：」）も省略する必要があります。メインのプログラムタスク (0)または現在のデバッグタスクが使用されます。

program

操作で参照される変数またはステートメントのラベルのコンテキストを指定するオプションのプログラム名です。「program<」が省略されている場合、コロン（「：」）も省略する必要があります。「task」で指定された最上位スタックのプログラム（または現在のデバッグプログラム）が使用されます。変数コンテキストについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

コマンドパラメータがすべて省略されている場合、指定された最後のコンテキストが使用されます。グローバルコンテキストまたは現在のデバッグプログラムが最初のデフォルトです。

詳細

eV+ 言語キーワードは、プログラムに含まれ、そのプログラムが実行される場合のみ、処理できます。DO コマンドは、プログラムを作成せずに単一のキーワードを実行できます。

DO モニタコマンドは、プログラム内に含まれているかのように、単一の操作を実行します。ロボットの移動（「DO READY」など）やプログラムステップ実行シーケンスの変更（「DO @ 2 GOTO 100など」）にこのコマンドを使用できます。

「@」が存在しない、または指定タスクの最上位スタックにプログラムがない、プログラムが無効な場合のみDO コマンドで新しいグローバル変数を作成できます。

DO コマンドが処理されるとき、次の動作が発生する可能性があります。

- DO コマンドで実行されるモーションのため、参照されるプログラムタスクの一時的なロボット設定、または軌道制御パラメータの設定が取り消されます。
- PAUSE コマンドで中断されているプログラムで REACT、REACTE、REACTI、RUNSIG コマンドがアクティブな場合、DO コマンドの操作実行中、これらのコマンドは再度有効になります。

例

次の例では、変換「safe.location」で定義されているロケーションまでの直線モーションを実行します。

```
DO MOVES safe.location
```

次の例では、プログラムタスク1として操作を実行し、プログラム 「io.check」のローカル変数 i に値 5 に割り当てます。

```
DO @1:io.check i = 5
```

関連キーワード

DOS

DO （プログラムコマンド）

EXECUTE

EXECUTE

SSTEP

XSTEP

ENABLE

1 つ以上のシステムスイッチを ON するモニタコマンドです。

構文

ENABLE switch, ..., switch

使用上の留意事項

プログラムの実行中、ENABLE モニタコマンドを使用できます。

指定のスイッチがインデックスの修飾子を受け付け、インデックスが 0 または括弧の有無を問わず省略されている場合、スイッチ配列のすべての要素は有効になります。

パラメータ

switch ON にするシステムスイッチの名称です。

その名称はそのスイッチを一意に特定する最小長さに短縮することができます。例えば、文字「ME」から始まる名前のスイッチは他にはないため、MESSAGES スイッチを「ME」と呼ぶことができます。

詳細

システムスイッチはeV+ システムのさまざまな動作を制御します。ハイパワーを有効にする命令が発行された場合、すべてのロボットに対して範囲外エラーを確認します。エラーがある場合、各ロボットに対してモニタウィンドウにメッセージが表示されます。使用可能なシステムスイッチとその機能については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

スイッチが ON の場合、スイッチが制御する機能は、機能する、または使用できます。DISABLE モニタコマンドまたはプログラムコマンドでスイッチを OFF にすると、関連する機能が機能しない、または使用できなくなります。

> **注:** システムスイッチは、すべてのプログラムタスクで共有されます。1 つ以上のタスクがスイッチを正しく設定できない可能性があるため、複数のタスクでスイッチを無効、有効にする場合には、よく検討する必要があります。
>
> DRY.RUN スイッチを無効にしても、タスク0 に対して次の EXECUTE モニタコマンドまたはプログラムコマンドが処理される、ロボットに対するATTACH プログラムコマンドが実行される、またはCALIBRATE モニタコマンドまたはプログラムコマンドが処理されるまでは、影響を与えません。

SWITCH モニタコマンドまたは SWITCH 関数を使用して、常にスイッチの状態を判断できます。

例

次の例では、MESSAGES システムスイッチを ON にします。

ENABLE MESSAGES

関連キーワード

DISABLE(モニタコマンド)

DISABLE（プログラムコマンド）

SWITCH（モニタコマンド）

SWITCH (プログラムコマンド)

SWITCH（実数値関数）

ESTOP

非常停止信号を受信した場合と同じ方法でロボットを停止するモニタコマンドです。

構文

ESTOP robot_num

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値と解釈されるオプションの実数値、変数、または式です。このパラメータが省略されているか 0 の場合、すべてのロボットに対する操作が変更されます。それ以外の場合、指定されたロボットに対する操作だけが影響を受けます。

詳細

このコマンドは非常停止と電力停止シーケンスをただちに開始します。システムの設定によっては、ブレーキをかける前やハイパワーをデアサートする前に、停止の制御が含まれない場合もあります。

指定された robot_num が接続されていない場合、エラーは発生しません。

例

次の例ではロボット2 に対して非常停止と電力停止シーケンスを開始します。

```
ESTOP 2
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

BRAKE

ESTOP （プログラムコマンド）

PANIC（モニタコマンド）

PANIC （プログラムコマンド）

STATE

EXECUTE

制御プログラムの実行を開始するモニタコマンドです。

構文

EXECUTE /C task program(param_list), cycles, step

使用上の留意事項

プログラムは、指定のプログラムタスクとしてすでにアクティブな状態であってはいけません。

パラメータ

/C
選択されたロボットを条件付きで割り付けるオプションの修飾子です。修飾子には、タスク0の実行開始時にのみ効果があります。

task
起動されるプログラムタスクを指定するオプションの整数値です。

program
実行対象のプログラム名です。名前が省略されている場合、選択されたタスクに対してEXECUTE モニタコマンド、プログラム コマンド、または PRIME モニタコマンドで指定された最後のプログラム名が使用されます。



警告: EXECUTE コマンドの入力時にプログラムを指定しない場合、前回のプログラムが再度実行されるため、予想外のロボットモーションが発生する可能性があります。

param_list
コンマで区切られた、定数、変数または式のオプションのリストであり、指定したプログラムの .PROGRAM ステートメント内の引数のタイプおよび数に対応する必要があります。プログラムが引数を必要としない場合、リストを空白にし、括弧を省略できます。

省略されたパラメータをスキップする場合、コンマを使用してプログラムパラメータを省略できます。リストの最後のパラメータが省略されている場合、コンマは必要ありません。省略されたパラメータは呼び出されたプログラムに未定義として引き渡され、DEFINED 関数で検出することができます。

パラメータは、次に示すように、開始した新しいタスクのコンテキストで評価されます。

cycles
実行されるプログラム実行サイクルの数を指定する、整数と解釈されるオプションの実数値、変数または式です。省略した場合、サイクルカウントは 1 と仮定されます。無制限のサイクルについては任意のマイナス値を指定します。最大ループカウント値は、32767です。

step
実行プログラムを開始するステップを指定する、整数と解釈されるオプションの実数値、変数または式です。省略されている場合、プログ

ラムの実行はプログラムの最初の実行可能なステートメント(最初の空白とコメントの行およびすべての AUTO、GLOBAL、LOCAL ステートメントの後) で開始します。

詳細

このコマンドは、指定された制御プログラムの実行を開始します。プログラムは、指定されたプログラムステップから実行を開始し、サイクルパラメータで指定された回数だけ実行されます。プログラムが指定されていない場合、システムは、選択されたプログラムタスクが実行した最後のプログラムを再度実行します。

> **注:** プログラムでは、EXECUTE プログラムコマンドを使用してその他の関連プログラムの実行を開始することができます。プログラムが他のプログラムの実行を開始すると、開始させた方のプログラムは STATUS および ERROR 関数を使用してもう一方のプログラムのステータスを監視します。

タスク番号が指定されていない場合、EXECUTE コマンドは、タスク番号 0 にアクセスします。

オプションの /C 修飾子には、タスク 0 の実行開始時にのみ効果があります。/C が指定されず、ロボットを接続できない場合、タスク 0 の EXECUTE コマンドは失敗します。割り付けのためには、ロボットのキャリブレーションが実施されていて、アーム電源が有効になっているかまたは DRY.RUN システムスイッチが有効である必要があります。/C が指定されている場合、タスク 0 の EXECUTE コマンドは、ロボットの接続を試みますが、ロボットを接続できない場合にエラー表示なしに実行を続行できます。

プログラムの実行が開始されるときは、必ず特定のデフォルトの条件が想定されます。次のプログラムコマンドと同等の意味になります。

```
CPON ALWAYS
DURATION 0 ALWAYS
FINE 100 ALWAYS
LOCK 0
MULTIPLE ALWAYS
NULL ALWAYS
OVERLAP ALWAYS
SPEED 100,100 ALWAYS
SELECT ROBOT = 1
```

以降の動作のために、ロボットの設定は保存されます。

STOP プログラムコマンドを実行した場合、RETURN プログラムコマンドをトップレベルプログラムで実行した場合、またはそのプログラムの最後に定義したステップに遭遇した場合、実行サイクルは終了します。値は、-32768～32767 の範囲に設定できます。cycles を省略、または値が 0 または 1 の場合、プログラムは一度だけ実行されます。cycles にマイナス値を指定すると、HALT プログラムコマンドが実行される、エラーが発生する、またはユーザー（他のプログラム）がプログラムの実行をアボートするまで、そのプログラムを継続的に実行します。

> **注:** 実行サイクルが開始するたびに、実行パラメータはデフォルトの値にリセットされます。このパラメータには、モーション速度、ロボット設定、サーボモードが含まれます。現在選択されているロボットは変更されません。

step が指定されている場合、プログラムはそのステップで実行を開始します。以降のサイクルは、必ずプログラムの最初の実行可能なステップから始まります。

例

次の例では、「assembly」という名称のプログラムの実行を（タスク 0 として）開始し、HALT プログラムコマンドが実行される、またはランタイムエラーが発生するまで実行を継続します。

```
EXECUTE assembly,-1
```

次の例では、プログラム名「test」のプログラムタスク 2 の実行を開始します。パラメータ値 1 と 2 がプログラムに渡されます。

```
EXECUTE 2 test(1,2)
```

次の例では、プログラムタスク 0 または現在のデバッグタスクが実行した最後のプログラムの実行を開始します。プログラムに渡されるパラメータはありません。

```
EXECUTE
```

関連キーワード

ABORT

ABORT（プログラムコマンド）

CALL

CYCLE.END（モニタコマンド）

CYCLE.END （プログラムコマンド）

EXECUTE（プログラムコマンド）

KILL（モニタコマンド）

KILL （プログラムコマンド）

PRIME

PROCEED

RETRY(モニタコマンド)

SSTEP

STATUS

STATE

XSTEP

FCOPY

既存のディスクファイルの情報を新しいディスクファイルにコピーするモニタコマンドです。

構文

FCOPY new_file = old_file

パラメータ

new_file
作成する新しいディスクファイルのファイル指定です。ピリオド(「.」)とファイルの拡張子が省略されている場合、デフォルトでは、拡張子が空白になります。必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。

old_file
既存のディスクファイル指定です。ピリオド(「.」)とファイルの拡張子が省略されている場合、デフォルトでは、拡張子が空白になります。必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。

詳細

新しいファイルが既に存在している、または古いファイルが存在していない場合、エラーが報告され、コピーは実行されません。既存のファイルを上書きすることはできません。既存のファイルをまず FDELETE コマンドを使用して消去する必要があります。

コピーする予定のファイルが読み取り専用属性の場合、新しいファイルも同じ属性になります。保護属性のあるファイルをコピーすることはできません。ファイル保護属性に関する説明については FDIRECTORY を参照してください。

ファイルをコピーすると、ファイル作成日付が標準ファイル属性と共に保存されます。影響を受ける属性は、ファイルがアーカイブされたことを示すためにクリアされるアーカイブビットのみです。

通常、ファイル指定は、次の 6 つの要素から構成されます。

1. オプションの物理デバイス（DISK> など）
2. オプションのディスクユニット（D: など）
3. オプションのディレクトリパス（DEMO¥ など）
4. ファイル名（NEWFILE など）
5. ピリオド文字（「.」）
6. ファイル拡張子（V2 など）

例

次の例では、ディスクデバイス "D" に "newfile.v2" という名称のファイルを作成します。このファイルはディスクデバイス "D" にある "oldfile.v2" という名称の既存ファイルの完全なコピーです。

```
FCOPY D:\newfile.v2 = D:\oldfile.v2
```

関連キーワード

COPY

DEFAULT

FCOPY （プログラムコマンド）

FRENAME

FDELETE

ファイル指定に一致する 1 つ以上のディスクファイルを削除するモニタコマンドです。

構文

FDELETE file_spec

使用上の留意事項

ファイルを削除した場合、ファイルに含まれる情報を回復することはできません。

このコマンドは、サブディレクトリのファイルも削除します。FDIRECTORY コマンドを共に使用して、サブディレクトリのファイルを削除します。

パラメータ

file_spec

削除するファイルのファイル指定です。file_spec には、オプションの物理デバイス、ディスクユニット、ディレクトリパスを含めることができます。ファイル名と拡張子を指定する必要があります。ファイル名または拡張子にワイルドカードに相当する文字を使用できます(以下参照) 。

必要に応じて、現在のデフォルトのディスクユニットとディレクトリパスが適用されます。DEFAULT コマンドを参照してください。.

ファイル指定にピリオドおよびファイル拡張子が含まれていない場合、拡張子は空であるとみなされます。

詳細

ファイル名と拡張子にワイルドカード文字（アスタリスク、「*」）を使用できます。ファイル名または拡張子内のワイルドカード文字は、その位置にある文字にかかわらず、受け付けることを示します。ファイル名または拡張子の最後にあるワイルドカード文字は、それ以降の文字にかかわらず、受け付けることを示します。

ワイルドカード指定に一致するすべてのファイルが削除されます。ワイルドカード機能を使用する場合、最初に同じファイル指定で FDIRECTORYコマンドを発行することが良いでしょう。削除予定のファイルの一覧を確認した後、同じファイル指定で FDELETE コマンドを発行できます。

例

次の例では、確認プロンプトで「Y」を選択するようにユーザーに促します。

次の例では、デフォルトのデバイスからディスクファイル「f3.lc」を削除します。

```
FDELETE f3.lc
```

次の例では、拡張子「V2」のディスクファイルすべてをディスク「D」から削除します。

```
FDELETE D:*.v2
```

次の例では、ファイル名が「abc」で始まるディスクファイルと拡張子が「b」で始まるディスクファイルすべてをディスク「A」から削除します。

```
FDELETE A:abc*.b*
```

関連キーワード

DEFAULT

DELETE

FDELETE

FDIRECTORY

FDIRECTORY

ディスクのファイル情報とストレージのスペース残量を表示、ディスクのサブディレクトリを作成、削除するモニタコマンドです。

構文

FDIRECTORY /qualifier file_spec

使用上の留意事項

パラメータ/修飾子を指定している場合、キーワードとスラッシュ（/）の間にスペースを入れることはできません。

サブディレクトリは、最大 16 レベルまでネストできます。ディレクトリパス指定の合計の長さは、デフォルトを含み 80 文字以内にする必要があります。

サブディレクトリにファイルが含まれている場合、またはそのファイルにアクセスしている場合（例えば FOPEND 演算の後など）はサブディレクトリを削除できません。

パラメータ

/qualifier	作成、または削除されるサブディレクトリを指定する、「/C」または「/D」のオプションの修飾子です。省略された場合、ディスクのリストが生成されます。
file_spec	表示するファイル、または作成、削除するサブディレクトリを選択するオプションのファイル指定文字列です。 ディレクトリが表示されている場合、ファイル指定には、物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、拡張子を使用できます。ファイル名または拡張子を省略、または文字と一致するワイルドカードを使用できます(次に示す項目 6 を参照)。 サブディレクトリを作成または削除する場合、ファイル名と拡張子を省略する必要があります。ディレクトリを指定し、最後に「\」の文字を付ける必要があります。 このコマンドには、ファイル指定の一部にデフォルトのディレクトリ指定(DEFAULT モニタコマンドで設定)を使用して、入力を支援する機能があります。

詳細

ファイル指定または修飾子を入力していない場合、デフォルトのディレクトリ全体のディレクトリ情報が表示されます。オプションのファイル指定を使用して、物理デバイス、ディスクドライブ、ディレクトリパスを指定し、選択したファイルを表示できます。

ディレクトリ情報を表示するとき、コマンドは、デフォルトのディレクトリ指定から取得した一部を含む、使用されているディレクトリパスを最初に表示します。次に、ファイル指定を満たす各ファイルに対して、次の情報を表示します。

- ファイル名
- ファイル拡張子
- ファイルの使用容量（キロバイト）
- ファイルの特殊属性コード
- ファイルが書き込まれた日時（表示されない場合があります）

モニタウィンドウで CTRL+C を入力すると、画面をアボートできます。

修飾子「/C」（作成）、または「/D」（削除）をキーボードから追加して、サブディレクトリを作成、削除できます。現在のデフォルトとコマンドラインからの入力を組み合わせたディレクトリ指定で示される最新のサブディレクトリが指定されたとみなされます。サブディレクトリを削除する場合、確認のメッセージが表示されます。

> **注:** サブディレクトリを作成、削除する場合、ディレクトリ指定の中間サブディレクトリすべてがすでに存在している必要があります。存在しない場合、作成または削除できません。サブディレクトリにディスクファイルが含まれている場合、サブディレクトリを削除できません。
>
> **重要:**ステートメント「FDELETE *.*」を使用して、サブディレクトリのすべてのファイルを削除できます。DEFAULT コマンドを使用すれば、このコマンドを発行する前に正しいサブディレクトリにいることが確認できます。

完全なファイル指定は、次の要素で構成されます。

1. 文字「>」を最後に付けたオプションの物理デバイス名。許容可能なデバイス名は、「DI」と短縮できる「DISK」です。

   物理デバイスを指定しない場合、文字「>」を入力する必要はありません。

2. コロン（「:」）を最後に付けたオプションのディスクユニット指定。ディスクユニットが指定されていない場合、現在のデフォルトのディスクが指定されます。一般的な eV+ ディスクファイルの場合、ユニットは、「A」または「C」など対象のディスクドライブのレター名になります。

指定する場合、ユニットの最後にはコロン（「:」）を付ける必要があります。

3. 対象のサブディレクトリを指定するオプションのディレクトリパス。このパラメータには、ファイル名とバックスラッシュ（¥）文字を使用する必要があります。

   先頭のスラッシュ（¥）は、そのディレクトリパスが始まる最上位のディレクトリを指定します。現在定義されているデフォルトのパスは使用されません。シングルスラッシュ（¥）文字で指定されたディレクトリパスは、最上位のディレクトリがアクセスされることを示します。

   先頭のスラッシュ（¥）がない場合、このパスが現在のデフォルトのパスに付加されることを意味します。指定されたユニットが現在のデフォルトのユニットと異なる場合、先頭のスラッシュ（¥）を指定し

ていなくても、ディレクトリパスは、必ず最上位のディレクトリから始まります。

4. 1～8 文字のオプションのファイル名。
5. ピリオド文字（.）。

   注: 拡張子を入力しない場合、省略できます。ピリオドを省略した場合、ファイル拡張子に「*」を指定したことと同じ意味になります。

6. 最大 3 文字のオプションのファイル拡張子。

ファイル名および拡張子にはワイルドカード文字（*）を使用できます。ファイル名または拡張子内のワイルドカードは、その位置に任意の文字を受け入れることができることを示します。ファイル名または拡張子の末尾にワイルドカードがある場合、任意の後続文字を受け入れることができることを示します。ディレクトリパスの指定にワイルドカード文字は使用できません。

ディスクファイルには、ファイルの使用を制限する独自の属性を持たせることができます。eV+ プログラムパッケージで使用する属性を次に示します。属性がファイルに適用されている場合、FDIRECTORY コマンドによって該当する文字が表示されます。

P　保護ファイル

システムメモリにファイルをロードし、ファイル内のプログラムを実行できます。実行中、プログラムを編集、表示またはトレースすることはできません。プログラムをメモリからディスクに保存することはできません。

FCOPY モニタコマンドを使用して、保護ファイルをあるディスクから別のディスクにコピーすることはできません。また、FLIST モニタコマンドを使用して表示したり、アプリケーションプログラムでアクセスすることもできません。

R　読み取り専用ファイル

システムメモリにファイルをロードし、ファイル内のプログラムを実行および表示することができます。プログラムを編集することや、メモリからディスクに保存することはできません。

読み取り専用ファイルは FCOPY モニタコマンドを使用してあるディスクから別のディスクにコピーすることが可能であり、FLIST モニタコマンドを使用して表示することも可能です。アプリケーションプログラムは上書きできません。

例

次の例では、現在のデフォルトディレクトリのデフォルトディスクにあるすべてのファイルの情報を直接表示します。

```
FDIRECTORY
```

次の例では、ディスク「A」のサブディレクトリ「v1」にある「demo」の名前が付いたすべてのファイルの情報を表示します。

```
FDIRECTORY A:¥v1¥demo
```

次の例では、デフォルトのディレクトリにある「f」で始まり「n」で終わる 3 文字のファイル名のすべてのファイルを表示します。

```
FDIRECTORY f*n
```

次の例では、デフォルトのディレクトリにある「f」で始まるすべてのファイルを表示します。

```
FDIRECTORY f*
```

次の例では、デフォルトのディレクトリにある拡張子「lc」のすべてのファイルを表示します。

```
FDIRECTORY .lc
```

次の例では、デフォルトのディスクの最上位ディレクトリにある拡張子「v2」のすべてのファイルを表示します。

```
FDIRECTORY ¥.v2
```

次の例では、ディスク「C」の最上位ディレクトリにサブディレクトリ「v1」を作成します。

```
FDIRECTORY/C C:¥v1¥
```

次の例では、ディスク「C」のサブディレクトリ「v1」にサブディレクトリ「t」を作成します。

```
FDIRECTORY/C C:¥v1¥t¥
```

次の例では、ディスク「A」の現在のデフォルトのディレクトリパス（現在のデフォルトのユニットが「A」でない場合は最上位のディレクトリ）にあるサブディレクトリ「t」を削除します。

```
FDIRECTORY/D A:t¥
```

関連キーワード

FCOPY

FLIST

FDELETE

FOPEND

DEFAULT

FLIST

指定したディスクファイルの内容をモニタウィンドウにリスト表示するモニタコマンドです。

構文

FLIST file_spec

使用上の留意事項

リスト表示を停止するには、CTRL+C を押します。

リスト表示は、システムメモリ内のプログラムやデータには影響しません。

パラメータ

file_spec	リスト表示するファイルのファイル指定です。file_spec には、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名およびオプションの拡張子を含めることができます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については、DEFAULT コマンドを参照してください。

詳細

このコマンドは、標準の ASCII 文字を含むディスクファイルの内容をモニタウィンドウにリスト表示します。このコマンドは、ディスクに保存されている eV+ アプリケーションプログラムをメモリに読み込むことなく確認できます。このコマンドを使用すると、eV+ の「STORE_」コマンドで作成したすべてのディスクファイルを読み取ることができます。

保護属性のディスクファイルはリスト表示できません。保護ファイルは、FDIRECTORY コマンドの出力で「P」として確認できます。

例

次の例では、ディスクファイル「test.v2」の内容をモニタウィンドウにリスト表示します。

```
FLIST test.v2
```

関連キーワード

DEFAULT

FDIRECTORY

LISTP

FREE

現在使用されていない使用可能なシステムメモリの割合（%）を表示するモニタコマンドです。

構文

FREE

詳細

このコマンドはシステムメモリの使用状況についての情報を表示します。この情報は、現在使用されていない、アプリケーションプログラムや変数で使用可能なメモリの割合（%）を示します。

次に示すように FREE コマンドが返す情報はパーセント表示（nn.nn）されます。

```
% unused program memory = nn.nn
```

eV+ システムソフトウェアを含まないすべてのシステムプログラムメモリをアプリケーションプログラムとデータの保存用に使用できます。ビジョンタスクやサーボタスクが CPU 上で動作している場合、これらのプログラムはプログラムメモリとみなされます。

FREEコマンドではプログラムメモリに使用可能な領域が、少ないながらもまだあるように表示されているにも関わらず、一部の操作ではエラーメッセージ"Not enough storage area"が表示される場合があります。 この原因は、未使用のメモリが断片化されてアプリケーションの情報を保存するには十分な大きさがないためと考えられます。この状態が発生した場合、メモリのすべての内容をディスクに保存したうえで、ZEROモニタコマンドでメモリをクリアしたのち、ディスクからプログラムや変数を再読み込みします。

これ以外のエラーメッセージが表示された場合は、メモリが破損している可能性があります。プログラムを保存して、もう一度 FREE コマンドを発行してください。エラーが繰り返し発生する場合は、コントローラを再起動してください。再起動した後もエラーが繰り返し発生する場合は、お近くの OMRON 担当者までお問い合わせください。

> **追加情報:** エラーについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、現在使用されていない使用可能なシステムメモリの割合（%）を返します。

```
FREE
```

次の情報が返されます（例）。

```
% unused program memory = 67.03
```

関連キーワード

FREE

ZERO

FRENAME

ディスクファイル名を変更をするモニタコマンドです。

構文

FRENAME new_file = old_file

パラメータ

new_file

ディスクファイルの新しい名前です。

ファイルの場所を指定する必要がある場合、ファイル名と拡張子の他に、オプションの物理デバイス、ディスクユニットおよびディレクトリパスをこのパラメータに含めることができます。

必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます（詳細については、DEFAULT コマンドを参照してください）。この名前がすでに使用されている場合は、エラーが発生します。

old_file

名前を変更するファイル名です。

ファイルが存在しない場合、エラーが発生します。デバイス、ディスクユニット、ディレクトリパスを指定する必要はありません。

詳細

ファイル名のみ変更されます。ファイルの内容とサイズには影響しません。

次の例に示すように、ディスクユニット（指定している場合）は左側に指定する必要があります。

例

次の例では、ディスク「D」の既存のファイル「data.v2」の名前を「parts.v2」に変更します。

```
FRENAME D:\parts.v2 = data.v2
```

関連キーワード

DEFAULT

FCOPY

RENAME

FSET

ネットワークデバイスの属性を設定または修正するモニタコマンドです。

構文

FSET device attribute_list

パラメータ

device	変更される属性を持つ物理デバイスの名前です。 名前は短縮できます。ユニット番号と名前の説明については、ATTACH プログラムコマンドを参照してください。
attribute_list	対象のデバイスの属性を指定するキーワードと引数のリストです。 有効なキーワードの詳細については、FSET プログラムコマンドの説明を参照してください。

詳細

FSET を TCP と共に使用

FSET コマンドでネットワークに新しいノードを定義して、TCP デバイスにアクセスできます。

FSET コマンドでこのデバイスにアクセスする場合、次の属性を使用できます。

- /ADDRESS

  IP アドレス

- /NODE

  ノード名

例

次の例では、「SERVER2」という名前の新しいネットワークノードを IP アドレス 192.9.200.22 で定義します。

```
fset tcp /node 'SERVER2' /address 192 9 200 22
```

関連キーワード

ATTACH

FSET （プログラムコマンド）

HERE

変換、またはプレシジョンポイント変数の値を現在のロボットのロケーションと等しい値に定義するモニタコマンドです。

構文

HERE @task:program **loc_variable**

使用上の留意事項

タスクが指定されていない場合、HERE コマンドは、eV+ モニタが（SELECT モニタコマンドを使用して）選択したロボットに関する情報を返します。タスクが指定されている場合、このコマンドは、そのタスクが（SELECT プログラム命令を使用して）選択したロボットのロケーションを返します。

eV+ システムがロボットを制御するように設定されていない場合、HERE コマンドを実行するとエラーになります。

パラメータ

@task:program	これらのオプションのパラメータは、ロケーション変数のコンテキストを指定します。ロケーション変数は、指定されたコンテキストから参照されるかのようにして処理されます。コンテキストが指定されていない場合、ロケーション変数はグローバル変数とみなされます。
loc_variable	変換変数、プレシジョンポイント変数、または最後に変換変数を指定する複合変換です。

詳細

このコマンドは、変換、またはプレシジョンポイント変数の値を現在のロボットのロケーションと等しい値に定義します。

通常、ロボットのロケーションは、読み取ったジョイントエンコーダのその瞬間の値で決定されます。ロボットのバックラッシュまたは直線性補正が有効な場合、命令されたロボットのロケーションが代わりに使用されます。

複合変換が指定されている場合、複合変換の右端の要素のみが、このコマンドで指定した値になります。複合変換の他の変換が定義されていない場合、エラーメッセージが表示されます。

例

次の例では、変換「place」を現在のロボットのロケーションと等しい値に定義します。

```
HERE place
```

次の例では、現在のロボットのロケーションをプログラム「test」のローカル変数として扱われているプレシジョンポイント「#pick」に割り当てます。

```
HERE @test #pick
```

関連キーワード

HERE（プログラムコマンド）

HERE（変換関数）

SELECT

SELECT（実数値関数）

WHERE

ID

システムのコンポーネントに関する識別情報を表示するモニタコマンドです。

構文

ID

使用上の留意事項

ロボットが接続されていない状態でこのコマンドを発行した場合、ロボットセクションは表示されません。

詳細

オプションのワードにはコード化されたシステムに関する情報が含まれます（16進値形式）。この情報はシステムのトラブルシューティングに役立ちます。

ユーザー定義の起動メッセージが定義されている場合、ID モニタコマンドが発行されたときにそのメッセージがモニタウィンドウに表示されます。

次の例にID モニタコマンドが発行されたときに表示される情報を示します。

```
Software:    3.0 87-1000 (Edit C8, 14-Jun-2019, Production Release)
Controller:  000-0000000
             eV+ Emulator
Security ID: 0000-0000-0000
Robot 1:     571-0 1000-0-0 6
             Adept eCobra 800 Robot (Professional).
```

ソフトウェアの情報は以下に示す形式で返されます。

```
Software: <version>.<revision> <opt1>-<opt2> (<ID message>)
```

version	eV+ のバージョン番号です。この番号は実数値関数ステートメント ID(3) が返す値と同じ値です。
revision	eV+ の改訂番号です。この番号は実数値関数ステートメント ID(4) が返す値と同じ値です。
opt1	システムオプションの 1 番目のワードです。このワードは実数値関数ステートメント ID(5) が返す値と同じ値です。
opt2	システムオプションの 2 番目のワードです。このワードは実数値関数ステートメント ID(6) が返す値と同じ値です。
ID message	識別文字列です。この番号は実数値関数ステートメント $ID(-1) が返す値と同じ値です。

コントローラの情報は以下に示す形式で返されます。

```
Controller: <model>-<serial><revision> <opt>
```

model	コントローラのモデル番号です。この番号は関数 ID(1) が返す値と同じ値です。
serial	コントローラのシリアル番号です。この番号は関数 ID(2) が返す値と同じ値です。
revision	16 進値で表示されるコントローラのハードウェアリビジョ

	ン/IDです（この値は関数 ID(10) で返されます）。
options	コントローラのオプションワードです。このワードは関数 ID(8) が返す値と同じ値です。

セキュリティID の情報は以下に示す形式で返されます。

```
SecurityID: <aaaa-bbbb-cccc>
```

aaaa-bbbb-cccc	オプションのライセンスを特定のコントローラと関連付けるために使用する 12 文字の 16 進値で表されるカードセキュリティコードです。

プロセッサの情報は以下に示す形式で返されます。

```
Processor n: <version>.<revision> <type>-<opt> <Memory>Mb
```

n	CPU の番号です。この番号は関数 ID(1,4, *n*) が返す値と同じ値です。
version	CPU のハードウェアバージョンコードです。このコードは関数 ID(3,4, *n*) が返す値と同じ値です。
revision	CPU のハードウェアリビジョンンコードです。このコードは関数 ID(4,4, *n)* が返す値と同じ値です。
type	CPU の種類です。これは関数 ID(5,4, *n*) が返す値と同じ値です。
opt	このプロセッサで実行中のシステムソフトウェアコンポーネントを示します。これは ID(6,4, *n*) が返す値です。 Bit 1 = eV+ Bit 2 = Vision Bit 3 = Servo
Memory	RAM のサイズ（単位はメガバイト）です。これは式 ID (7,4, *n*) / 1024 が返す値と同じ値です。

```
Robot n: <model>-<serial> <opt1-opt2> <cat> <module> <module name>
```

model	ロボットのモデル番号です。この番号は関数 ID(1,10+n) が返す値です。
serial	ロボットのシリアル番号です。この番号は関数 ID(2,10+n) が返す値です。
opt1	ロボットのオプションの 1 番目のワードです。このワードは関数 Id(8,10+n) が返す値です。
opt2	ロボットのオプションの 2 番目のワードです。このワードは関数 Id(11,10+n) が返す値です。
cat	ロボットのセーフティレベルを示す 1 桁の数字です。この数字は関数 Id(13,10+n) が返す値です。
module *module name*	これらの値は、運動学的デバイスモジュールの番号（関数 Id(5,10+n) が返す値）と名前（関数では参照不可）です。

例

次の条件下で ID モニタコマンドを発行するとモニタウィンドウに次の情報が表示されます。

- システムがエミュレーションモードで動作している。
- システムにeCobra 800が一台設定されている。
- eV+ソフトウェアのバージョンが 3、リビジョンが 0。
- システムの1番目のオプションのワード = 87^H (10000111)

```
Software:    3.0 87-1000 (Edit C8, 14-Jun-2019, Production Release)
Controller:  000-0000000
             eV+ Emulator
Security ID: 0000-0000-0000
Robot 1:     571-0 1000-0-0 6
             Adept eCobra 800 Robot (Professional).
```

関連キーワード

ID（実数値関数）

IO

外部デジタル入出力信号、または内部ソフトウェア信号の現在の状態を表示するモニタコマンドです。

構文

IO signal_group

使用上の留意事項

このモニタコマンドは、ホストにマップされている信号の値を返します。マップされていない信号に対しては「-」を返します。

パラメータ

signal_group　　表示するデジタル信号を選択するオプションの整数値です。

詳細

システムデジタル信号のモニタに IO コマンドを使用できます。signal_group が指定されていない場合、すべての入力信号と出力信号が表示されます。

signal_group が指定されている場合、0～3の値にする必要があります。システムに多くの信号が装備されている場合、信号グループを表示すると役立ちます。

信号グループ	表示される信号
0	デジタル出力信号
1	デジタル入力信号
2	システムソフトウェア信号
3	3000 シリーズのデジタル出力信号
4	ホストにマッピングされている信号

ON の信号には「1」、OFF の信号には「0」、ハードウェア設定のない信号には「-」が表示されます。

例

次の例では、IO 0 コマンドを使用した表示のサンプルです。

```
0032-0001  ----  ----  ----  ----  ----  ----  0000  0110
0064-0033  0000  0000  0000  0000  0100  0000  0000  0000
```

この表示は、信号 2、3、47 が ONであることを示します。それ以外は OFF または装備されていないことを示しています。

関連キーワード

RESET

SIG

SIG.INS

SIGNAL（モニタコマンド）

SINGLE（プログラムコマンド）

JOG

ロボットの指定されたジョイントをジョグしたり、指定されたデカルト座標系の座標軸方向に沿ってロボットツールを移動します。JOG が実行されるたびに、ロボットは最大 300 ms 動作します。

構文

JOG (status) **robot, mode, axis, speed,** location, appro_dist

使用上の留意事項

JOG を実行する度に、ロボットは JOG.TIME システムパラメータで指定した時間だけ動きます。

COMP 以外のモードを使用している場合、どのタスクも指定されたロボットに割り付けることができません。割り付けを試みた場合、エラーメッセージ「Robot interlocked」（ロボットがインターロックしました）が表示されます。

JOG コマンドでロボットを動かした後、（ペンダントで手動モードが選択されていたかのように）システムは手動モードのままとなります。JOGプログラムコマンドのmode 5（またはペンダント）を使用して、COMP モードに戻すことができます。それ以外の場合にタスクがロボットへの割り付けを実行しようとすると、エラー「COMP mode disabled」（COMP モード無効）が返されます。

ジョイントが範囲外の場合、JOG コマンドを使用してジョイントを範囲内に戻すことができます。詳細については、次の項を参照してください。

パラメータ

status 成功の場合1、それ以外の場合はeV+ エラーコードを返すための状態変数です。オプションです。

robot ロボットの番号を指定します。

mode 次に示すジョグモードを指定します。

-1 キープアライブモード。JOG.TIME システムパラメータで指定した時間まで前のコマンドを継続します。

1 フリージョイントモード。speedの値が正なら、指定されたジョイントをフリーモードにします。speedの値が負なら、指定されたジョイントのフリーモードを解除します。

2 各ジョイント制御

3 ワールド座標系制御

4 ツール座標系制御

5 COMP モードに戻します。

6 今後の使用のための予備

7 指定されたロケーションに向かって指定された速度でジョグします。

8 最も近いワールド軸に対してロボットツールのZ 軸が平行になるようにジョグします。

9 指定されたロケーションによって定義された座標系に対するデカルト制御

axis 指定されたジョグモードに従って目的の動作をするために（上記参照）、ジョイント番号または直交座標（X=1、Y=2、...）を指定します。

モード 7、8 の場合、このパラメータは無視されますが値は必ず指定する必要があります。

speed 動作の速度と方向を指定します。これは、手動モードでの速度率（%）として解釈されます。100 以上の値は 100%、-100 以下の値は -100% として解釈されます。

フリーモードが指定されている場合、speedの値が正なら、指定されたジョイントをフリーモードにします。speedの値が負なら、指定されたジョイントのフリーモードを解除します。

location ロボットが移動する目標位置を指定する、変換、プレシジョンポイント、位置関数または複合変換です。オプションです。モード7、9 以外のすべてのモードでは、このパラメータは無視されるため省略することができます。

appro_dist 指定したロケーションと実際に希望する目標位置の間の、ロボットツールZ 軸に沿った距離を指定する実数値式です。オプションです。

距離の値が正の場合、指定されたロケーションからツールを後方（ツールZ軸のマイナス方向）にオフセットします。距離の値が負の場合、ツールを前方（ツールZ軸のプラス方向）にオフセットします。このパラメータはモード 7 以外では使用しません。

詳細

状態変数が与えられ、エラーがある場合、JOG コマンドは直接エラーを返さず、単に状態変数にエラーを返します。

JOG コマンドを実行する度に、ロボットは JOG.TIME システムパラメータで指定した時間だけ動きます。前回の動作が完了する前に、別の JOG を実行できます。長期間の円滑なモーションについて、次のJOGコマンドを前のJOGコマンドのJOG.TIME値の内で実行する必要があります。この目的のためにキープアライブ

モードを使用できます。キープアライブモードは、JOG.TIME のタイムアウト後には効果がありません。ロボットが停止する前だけ有効となります。

コマンドが処理されるとき、次のエラー状態が報告される可能性があります。

- Mode 1: 指定されたジョイントにフリーモードが許可されていない場合、エラー「Illegal joint number」（-609、不正なジョイント番号）が返されます。
- Mode 2: ロボットがジョイント制御をサポートしていない場合、エラー「Joint control of robot not possible」（-938、ロボットのジョイント制御ができません）が返されます。
- Mode 3、4、8、9: ロボットが直交座標制御をサポートしていない場合、エラー「Cartesian control of robot not possible」（-635、ロボットの直交座標制御ができません）が返されます。
- Mode 7: ロケーションに到達できない場合、動作が可能な範囲内で動作は停止し、停止時にエラー「Location out of range」（-610、ロケーション範囲外）が返されます。動作中に他の動作エラー（障害物に遭遇など）が発生した場合、関連したエラーが報告されます。
- Mode 7、9: ロケーションが指定されていない場合、エラー「Missing argument」（-454、引数がありません）が返されます。モード 7 では、ロケーションが変換で指定されている場合、指定のロケーションに向かって直線動作が実行されます。ロケーションがプレシジョンポイントで指定されている場合、関節補間動作が実行されます。しかし、ロボットがロケーションを指定する方法に関連付けられた動作の種類を許可していない場合（ロボットが関節補間動作を許可していないなど）、ロボットが許可している方法で動作が実行されます。

ロボットのジョイントが範囲外の場合、次のいずれかの方法でジョイントを範囲内にすることができます。

- ペンダントで MAN モードに入り、手動でジョイントを制御します。
- ペンダントを COMP モードにして、JOG コマンドを使用して範囲内にジョイントを戻します（JOG はペンダントが COMP モードの場合のみ許可されます）。

> **注:** ジョイントが範囲外のとき、COMP モードで出来ることは制限されます。この場合、すべての動作コマンド（JOG 以外）はエラー「Position out of range」（位置範囲外）を返します。さらに、JOG では、ジョイントを範囲内に戻す方向にのみジョイントを動かすことができます。
>
> **追加情報:** エラーについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例ではジョイントモードでマイナス方向にジョイント3 をジョグします。

```
JOG 1, 2, 3, -10
```

次の例ではワールドモードでX 軸方向にジョグします。

```
JOG 1, 3, 1, 10
```

次の例ではツールモードでY 軸方向にジョグします。

```
JOG 1, 4, 2, 10
```

次の例では「loc1」として定義されているロケーションに向かってジョグします。

```
JOG 1, 7, 1, 10,loc1
```

次の例では「loc1」として定義されているロケーションの50 mm 上の位置に向かってジョグします。

```
JOG 1, 7, 1, 10, loc1, 50
```

関連キーワード

DRIVE

JMOVE

JOG（プログラムコマンド）

JOG.TIME

MOVE（プログラムコマンド）

KILL

プログラム実行スタックをクリアし、接続されている I/O デバイスを切り離すモニタコマンドです。

構文

KILL task

使用上の留意事項

指定のプログラムが実行している間、KILL モニタコマンドは使用できません。

指定のタスク実行スタックが空の場合、KILL モニタコマンドは効果がありません。

パラメータ

task	クリアするプログラムタスクを指定する、整数として解釈されるオプションの実数値、変数、または式です。

詳細

この命令は、選択したプログラム実行スタックをクリアし、開いているファイルを閉じ、異常なプログラム終了で接続された状態になっている I/O デバイスを切り離します。

プログラムが PAUSE プログラム命令を実行した場合、プログラムが ABORT モニタコマンドまたはプログラムコマンドによって終了した場合、または I/O デバイスが割り付けられているもしくはファイルが開いている間にエラー状態が発生した場合、この状況が生じます。アクセスが制限されている I/O デバイスが接続されている場合、切り離されるまで他のプログラムタスクはそのデバイスを使用できません。

タスク番号が指定されていない場合、KILL コマンドは、タスク番号 0 にアクセスします。

例

次の例では、プログラムが実行していない場合、タスク 10 からプログラムを削除します。

```
KILL 10
```

関連キーワード

ABORT

EXECUTE

STATUS

LIST

式の値を表示するモニタコマンドです。

構文

LIST @task:program expression, ..., expression

パラメータ

@task:program	これらのオプションのパラメータは、指定されたロケーション変数のコンテキストを指定します。ロケーション変数は、指定のコンテキストから参照されるかのようにして処理されます。コンテキストが指定されていない場合、ロケーション変数はグローバル変数とみなされます。変数コンテキストに関する詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。 グローバルコンテキストが使用され、BASE、DEST、HERE または TOOL キーワードが参照されている場合、この関数は eV+ モニタが選択したロボットに関する情報を返します(詳細については SELECT モニタコマンドを参照してください) 。
expression	オプションの実数型の定数、関数、変数または式であり、その値が表示されます。

詳細

このコマンドは、どの種類の式でも表示できます。LISTx コマンドとは異なり、右側のインデックスが空白の場合、複数の配列要素は表示されません。

エラーが発生した場合、該当する式に対してエラーメッセージを含む行が表示されます。

エラーが発生していない場合、式の値は、次に示すいずれかのデータタイプ形式で表示されます。値の形式はタイプによって異なります。

```
type = value
```

データの種類	値
REAL	単一の数値
TRANS	X、Y、Z、y、p、r を示す 6 つの数値
PPOINT	プレシジョンポイントの J1 から Jn までの N 個の数値
STRING	引用符付きの文字列
UNKNOWN	* で始まるエラー文字列

関連キーワード

FLIST（モニタコマンド）

LISTL（モニタコマンド）

LISTP

LISTR（モニタコマンド）

LISTS（モニタコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

SELECT

LISTL

リストされたロケーションの値を表示するモニタコマンドです。

構文

LISTL /N @task:program location, ..., location

パラメータ

/N	/N が指定されている場合、変数名のみ表示され、値は表示されません。配列の次元を示す空の括弧付きの各配列が 1 行で表示されます。 引数リストにロケーションパラメータが含まれている場合、/N は無視されます。
@task:program	これらのオプションのパラメータは、指定されたロケーション変数のコンテキストを指定します。ロケーション変数は、指定のコンテキストから参照されるかのようにして処理されます。コンテキストが指定されていない場合、ロケーション変数はグローバル変数とみなされます。変数コンテキストに関する詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。 グローバルコンテキストが使用され、BASE、DEST、HERE または TOOL関数が参照されている場合、この関数は eV+ モニタが選択したロボットに関する情報を返します(詳細については、SELECT モニタコマンドを参照してください) 。
location	値が表示されるオプションの変換、プレシジョンポイント、ロケーション関数、または複合変換です。

詳細

パラメータが指定されていない場合、すべてのグローバルロケーション変数の値がアルファベット順に表示されます。プログラムコンテキストを指定し、変数がリストにない場合、対象のプログラムにローカルなすべてのロケーション変数が表示されます。

各ロケーションパラメータには、ワイルドカード文字をいくつでも使用できます。各ワイルドカードは、0、1、または複数の文字に対応できます。

配列要素を指定した場合、その要素が表示されます。配列名を明示的インデックスなしに指定した場合、配列全体が表示されます。多次元配列で最も右のインデックスを 1 つ以上省略した場合、これらのインデックスで定義されたすべての要素が表示されます。例えば、ステートメント
「LISTL a[3,2,]」は、要素a[3,2,0]からa[3,2,last]を表示します。

例

次の例では、変換「pick」、複合変換「hold:par」、モニタウィンドウの現在のツール変換の値を表示します。

```
LISTL pick,hold:part,TOOL
```

次の例では、プログラム「test」でローカルに定義されたすべてのロケーション変数の値を表示します。

```
LISTL @test
```

関連キーワード

LISTL（モニタコマンド）

LISTP

LISTR（モニタコマンド）

LISTS（モニタコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

LISTP

リストされたユーザープログラムのすべてのステップを表示するモニタコマンドです。

構文

LISTP program, ..., program

使用上の留意事項

保護されているプログラムは表示できません。

プログラムは、システムメモリ内にある必要があります。

パラメータ

program　　表示対象のアプリケーションプログラム名です。

詳細

コマンドラインに複数のプログラムが指定されている場合、指定したすべてのプログラムがモニタウィンドウに表示されます。プログラム名が指定されていない場合、このコマンドは、システムメモリにあるアクセス保護されていないすべてのプログラムを表示します。

> **追加情報:** プログラムが実行不可能とレポートし、V+ Editor にエラーが表示されない場合、このモニタコマンドを実行すると、左側の余白にクエスチョンマーク（?）を表示し、問題のあるプログラム行を示します。

関連キーワード

FLIST

LIST

LISTL

LISTR

LISTS

LISTR

指定の実数式の値を表示するモニタコマンドです。

構文

LISTR /N @task:program expression, ..., expression

パラメータ

/N	/N が指定されている場合、変数名のみ表示され、値は表示されません。配列の次元を示す空の括弧付きの各配列が 1 行で表示されます。 引数リストに式のパラメータが含まれている場合、/N は無視されます。
@task:program	これらのオプションのパラメータは、実数値変数または指定されたタスク専用関数のコンテキストを指定します。実数値変数と関数は、指定されたコンテキストから参照されるかのように処理されます。コンテキストが指定されていない場合、グローバルコンテキストが使用されます。変数コンテキストに関する詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。 グローバルコンテキストが使用され、BASE、DEST、HERE または TOOL関数が参照されている場合、この関数は eV+ モニタが選択したロボットに関する情報を返します(詳細については SELECT モニタコマンドを参照してください) 。
expression	オプションの実数型の定数、関数、変数または式であり、その値が表示されます。

詳細

パラメータが指定されていない場合、すべてのグローバル実数値変数の値が表示されます。プログラムコンテキストを指定し、式がリストにない場合、対象のプログラムにローカルなすべての実数値変数が表示されます。

式の各パラメータには、ワイルドカード文字をいくつでも使用できます。各ワイルドカードは、0、1、または複数の文字に対応できます。

配列要素を指定した場合、その要素が表示されます。配列名を明示的インデックスなしに指定した場合、配列全体が表示されます。多次元配列で最も右のインデックスを 1 つ以上省略した場合、これらのインデックスで定義されたすべての要素が表示されます。例えば、ステートメント「LISTR b[3,2,]」は、要素b[3,2,0]からb[3,2,last]を表示します。ステートメント LISTR b[, ,]は配列全体を表示します。

> **注:** 一部の関数は特定の eV+ プログラムに関連する情報を返します（IOSTAT および PRIORITY など）。LISTR コマンドで参照されている場合、これらの関数は、コンテキストパラメータ（@task）で指定したプログラムタスクの値を返します。タスクのコンテキストが指定されていない場合、これらの関数はタスク 0 を返します。

例

次の例では、実数値変数「loop.count」の値とシステムタイマー 2 の現在の値をモニタウィンドウに表示します。

```
LISTR loop.count, TIMER(2)
```

関連キーワード

LIST（モニタコマンド）

LISTL（モニタコマンド）

LISTP

LISTS（モニタコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

LISTS

指定の文字列の値を表示するモニタコマンドです。

構文

LISTS /N @task:program string, ..., string

パラメータ

/N	/N が指定されている場合、変数名のみ表示され、値は表示されません。配列の次元を 0～2 個のコンマで示した、インデックスがない括弧付きの各配列が 1 行で表示されます。 /N オプションは、文字列定数、関数、式として指定された文字列引数には効果がありません。
@task:program	これらのオプションのパラメータは、指定された文字列変数のコンテキストを指定します。文字列変数は、指定のコンテキストから参照されるかのように処理されます。コンテキストが指定されていない場合、その文字列変数はグローバル変数とみなされます。変数コンテキストに関する詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。
string	値が表示されるオプションの文字列定数、関数、変数、式です。

詳細

パラメータが指定されていない場合、すべてのグローバル文字列変数の値がアルファベット順に表示されます。プログラムコンテキストを指定し、変数がリストにない場合、対象のプログラムにローカルなすべての文字列変数が表示されます。

各文字列パラメータには、ワイルドカード文字をいくつでも使用できます。各ワイルドカードは、0、1、または複数の文字に対応できます。

配列要素を指定した場合、その要素が表示されます。配列名を明示的インデックスなしに指定した場合（$line[ ]など）、配列全体が表示されます。多次元配列で最も右のインデックスを 1 つ以上省略した場合、これらのインデックスで定義されたすべての要素が表示されます。例えば、ステートメント「LISTS $c[3,2,]」は、要素$c[3,2,0]から$c[3,2,last]を表示します。

LISTS コマンドは、次の特殊な方法を使用して、文字列の値で特定の文字を表示します。

- ASCII 制御文字（10 進数の 0～31）は、サーカムフレックス（「^」、10 進数の 94）に続き、実際の制御文字に 96（10 進数）を加算した値と等しい ASCII 文字で構成される、2 文字シーケンスとして表示されます。例えば、キャリッジリターン（10 進数の 13）は、「^m」（13 + 96 = 109：「m」の ASCII 値）に変換されます。
- ダブルクォーテーション（「""」：10 進数の 34）は「^」として表示されます。
- サーカムフレックス（「^」：10 進数の 94）は「^^」として表示されま

す。

- LISTS コマンドは、パリティビットを持つバイト（8 ビットバイトの上位ビット）を区別しません。LISTS は、$CHR(^B11000001) と $CHR (^B01000001) の両方とも「A」として表示します。

例

次の例では、文字列変数「$message」の値と最新のエラーメッセージの文字をモニタウィンドウに表示します。

```
LISTS $message, $ERROR(ERROR(0))
```

関連キーワード

LIST（モニタコマンド）

LISTL（モニタコマンド）

LISTP

LISTR（モニタコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

LOAD

指定のディスクファイルの内容をシステムメモリにロードするモニタコマンドです。

構文

LOAD /qualifier **file_spec**

使用上の留意事項

読み取り専用モードでファイルを読み込んだ場合、ファイルにある最初のプログラムのみがモニタウィンドウにリスト表示されます（すべてのプログラム名を非表示にする /Q を指定していない場合）。

パラメータ

/qualifier　オプションの修飾子です。取り得る値を次に示します。

Switchの値	目的
/Q	ファイルを読み込んだとき、モニタ画面にプログラム名のリストを非表示にします。
/S	SQUEEZE ユーティリティプログラムがプログラムファイルで動作するときと同じ方法でメモリにロードされる圧縮プログラムです。 文字シーケンス「;*」で始まるフルラインコメントを除き、すべてのインラインコメントとフルラインコメントは削除されます。
/R	他の留意事項でより安全なモードの使用を規定している場合を除き、読み込まれるすべてのプログラムは強制的に読み取り専用モードになります。



注意: /S スイッチでファイルを読み込む場合、誰かが誤って圧縮ファイルを編集したり、変更を保存するときに非圧縮ファイルを上書きすることを防ぐために、/R スイッチも使用してください。

file_spec　読み込むプログラムおよび変数があるディスクファイルのファイル指定です。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。

必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。

ファイルの拡張子を指定せず、ローカルディスクにアクセスに

する場合、指定のファイル名に拡張子「.V2」が付加されます。

詳細

このコマンドは、指定のディスクファイルの内容をシステムメモリにロードします。ディスクファイルにはプログラムと変数を含めることができますが、そのフォーマットは eV+ の STORE_ コマンドで作成したものである必要があります。

すでに同じ名前のプログラムがメモリに存在する場合、プログラムの読み込みを試みると、エラーメッセージが表示され、新しいプログラムは読み込まれません。LOAD コマンドを使用して同じ名前の新しいプログラムを読み込むためには、事前に現在読み込まれているプログラムを DELETE コマンドを使用して削除するか、メモリを ZERO コマンドを使用してゼロ化する必要があります。

ディスクファイルの変数と同じ名前のロケーション変数、実数値変数、または文字列変数がすでにメモリにある場合、以前の変数は警告なしにディスクの情報に従って削除され、置き換えられます。

プログラムがシステムに読み込まれている際に eV+ が処理できないプログラム行があると、モニタ画面にエラーメッセージが表示されて、その行にクエスチョンマークが付けられます。プログラムがメモリに読み取られた後は eV+ エディタを使用してプログラムを修正できます。例えば、オフラインで作成されたプログラムをロードする場合に役立ちます。

LOAD コマンドによりプログラムをメモリに読み込む場合、すべてのプログラムは、ディスクファイルから最初に読み取られたプログラムと同じ名前のプログラムモジュールに入力されます。まだ存在しない場合は、プログラムモジュールが作成されます。読み込まれたプログラムは、ディスクから読み取られた順番でモジュールに入力されます。プログラムモジュールについては MODULE コマンドの説明を参照してください。

eV+ システムで利用可能なオートスタート機能によって、ロボットシステムの電源を入れたときや eV+ をディスクから読み込んだときに LOAD コマンドを自動的に発行することができます。詳細については、『eV+ 3 ユーザーズマニュアル (SBCE-486)』を参照してください。

ファイルが独自のバイナリプログラムフォーマットである場合、LOAD コマンドはそのファイルが要求する独自の操作を適用します。

例

次の例では、ディスク「D」（SD カード）からファイル「PALLET.V2」の内容をロードします。

```
LOAD D:pallet
```

次の例では、ファイル「F3.PG」のすべてのプログラムをシステムメモリにロードします。

```
LOAD f3.pg
```

関連キーワード

DEFAULT

MODULE

STORE
STOREL
STOREM
STOREP
STORER
STORES

MDIRECTORY

システムメモリにあるすべてのプログラムモジュール、または指定のプログラムモジュールにあるプログラムの名前を表示するモニタコマンドです。

構文

MDIRECTORY /M module

パラメータ

/M	モジュール内の修正済みモジュールまたはプログラムのみをリスト表示するように指定するオプションの修飾子です。
module	オプションのメモリ内のプログラムモジュール名です。このパラメータを省略した場合、メモリのすべてのモジュールがリスト表示されます。このパラメータが指定されている場合、名前付きモジュールのすべてのプログラムがリスト表示されます。

詳細

このコマンドを使用して、システムメモリに現在定義されているプログラムモジュールの情報を取得できます。プログラムモジュールは、LOAD コマンドによって自動的に作成されます。MODULE コマンドを使用してプログラムモジュールを作成、拡張、再配置できます。

コマンドパラメータを省略した場合、MDIRECTORY コマンドは、メモリに現在あるすべてのプログラムモジュールの名前をリスト表示します。

どのモジュールまたはプログラム名についても、メモリ内のコピーが、最後にロードまたは保存されてから修正されている場合、モジュールまたはプログラム名の前に「M」が表示されます。

module パラメータを指定している場合、MDIRECTORY コマンドは DIRECTORY コマンドと同様ですが、指定されたモジュール内のプログラムだけが、そのモジュールでの順番でリストされる点は異なります。表示情報の詳細については、DIRECTORY コマンドを参照してください。

MODULE モニタコマンドを使用すると、モジュール内のプログラムの順番を変更できます。

例

次の例では、メモリにあるすべてのプログラムモジュールの名前を表示します。

```
MDIRECTORY
```

次の例では、プログラムモジュール「main.package」にあるすべてのプログラムの名前を表示します。

```
MDIRECTORY main.package
```

関連キーワード

DELETEM

DIRECTORY

LOAD

MODULE

STOREM

MODULE

新しいプログラムモジュールを作成する、または既存のモジュールの内容を修正するモニタコマンドです。

構文

MODULE module = program, ..., program

パラメータ

module プログラムモジュール名です。

program メモリ内のプログラム名です。

詳細

プログラムモジュールは、単一の名前で参照できるプログラムグループです。次のモニタコマンドを使用して、プログラムモジュールにアクセスできます。

- LOAD は必要に応じて新しいモジュールを作成し、ディスクファイルから読み込んでモジュールにプログラムを入力します。
- DELETEM はモジュールにあるすべてのプログラムを削除してから、モジュールを削除します。
- MDIRECTORY は現在メモリにあるすべてのモジュール、または名前付きモジュールのすべてのプログラムをリスト表示します。
- MODULE は、新しいモジュールを作成する、または既存のモジュールの内容を修正します。
- STOREM は、モジュールにあるすべてのプログラムをディスクファイルに保存します。

プログラムファイルをディスクから読み取ると、LOAD モニタコマンドによってプログラムモジュールが自動的に作成されます。MODULE コマンドを使用すると、新しいプログラムモジュールを作成したり、システムメモリですでに定義されているモジュールの内容を拡張または再配置することができます。

指定したプログラムモジュールがまだ存在していない場合、MODULE コマンドによってそのモジュールが作成されます。この場合、リストされているすべてのプログラムは、新しいモジュールに配置されます。

指定したモジュールが存在している場合、MODULE コマンドによって、リストされているプログラムがモジュールの末尾に追加されます。指定されたモジュールに任意のプログラムが既に存在している場合、それらのプログラムはモジュールの最後に移動されます。

メモリにある各プログラムは、2 つ以上のモジュールに属することはできません。別のモジュールにすでに存在するプログラムをモジュールに追加した場合、そのプログラムは元のモジュールから削除されます。

例

以下の例では、「system.1」という名前のプログラムモジュールがメモリに存在しない場合に、そのモジュールを作成して、3 つのプログラムをモジュールに格納し

ます。「system.1」という名前のプログラムモジュールがメモリに存在している場合、このコマンドは、そのモジュールに 3 つのプログラムを追加します。

```
MODULE system.1 = main.program, subroutine.1, subroutine.2
```

関連キーワード

DELETEM

LOAD

MDIRECTORY

MODULE

STOREM

NET

ネットワーク状態の情報を表示するモニタコマンドです。また、eV+ システムで現在定義されているリモートマウントの詳細情報も表示します。

構文

NET mode

パラメータ

mode パラメータは、どの状態の情報を表示するかを指定するオプションの値です。mode パラメータの値を次の表に示します。

> **注:** オプションの mode パラメータを省略すると、値 0 を指定したのと同じ機能があります。

値	説明

0	次に示すようにしてネットワークステータス情報を表示します。
	
	`.NET 0`
	`  TCP/IP:  Up`
	`  FTP:  Option not installed`
	`  Local IP address:  192.9.222.252`
	`  Packets transmitted:  105007`
	`  Packets received:  577717`
	
	`  Transmission errors:  0`
	`  Reception errors:  6`
	`  Missed packets:  0`
	`  Available TCP connections:  32`
	
	`  Mount  Node  Path`
	`  XC  ASERVER  C:/ADEPT/DISKS/DISK_C`

1	次に示すようにして出力に対するエラーに関する情報を表示します。
	
	`.NET 1`
	`  TCP/IP:  Up`
	`  FTP:  Option not installed`
	
	`  Local IP address:  192.9.222.252`
	
	`  Packets transmitted:  105007`
	`  Packets received:  577875`
	`  Transmission errors:  0`
	`  Reception errors:  6`
	`  Missed packets:  0`
	
	`  Reception framing error:  0`
	`  Reception CRC error:  3`
	`  Reception overflow:  3`
	`  Reception buffer error:  0`
	`  Transmission late collision:  0`

値	説明
	Transmission loss of carrier: 0 Transmission retry error: 0 Transmission buffer error/underflow: 0 Available TCP connections: 32 Mount Node Path XC ASERVER C:/ADEPT/DISKS/DISK_C
2	次に示すようにして、定義されたノードのネットワーク状態と IP アドレスを表示します。 .NET 2 TCP/IP: Up FTP: Option not installed Name IP address ASERVER 192.9.222.252

詳細

このコマンドは TCP/IP プロトコルに関する情報を表示します。

次の表に示すように、3 つのネットワーク状態のうちの 1 つが表示されます。

ネットワーク状態	説明
Up（アップ）	ネットワークが無事初期化され、作動していることを示します。
Option installed（オプションインストール済）	ソフトウェアのオプションはインストールされているが、プロトコルが現在有効ではありません。
Hardware not installed（ハードウェア未インストール）	イーサネットケーブルが接続されていません。

次の例に示すように、TCP/IP がアップ状態の場合、追加の情報も表示されます。

例

TCP/IP ネットワークが Up 状態の場合、NET コマンドを発行すると次の例に示すように情報が返されます（値は異なる可能性があります）。

```
Local IP address                    192.9.222.252
Packets transmitted                 105007
Packets received                    577717
Transmission errors                 0
Reception errors                    0
```

```
Missed packets                          0
Available TCP Connections               32
```

Mode が 0 以外の値で指定されている場合に限り、次の追加の出力が表示されます。

```
Reception framing error                  0
Reception CRC error                      0
Reception overflow                       0
Reception buffer error                   0
Transmission late collision              0
Transmission loss of carrier             0
Transmission retry error                 0
Transmission buffer error/underflow      0
```

関連キーワード

NETWORK

PANIC

外部 E-stop ボタン押下をシミュレートし、すべてのロボットをただちに停止するが、ロボットのハイパワーを OFF にしないモニタコマンドです。

構文

PANIC robot_num

パラメータ設定

robot_num	影響を受けるロボットの番号を示す整数値と解釈されるオプションの実数値、変数、または式です。このパラメータが省略されているか 0 の場合、すべてのロボットに対する操作は変更されます。それ以外の場合、指定されたロボットに対する操作だけが影響を受けます。

詳細

このコマンドは次の動作を実行します。

- ロボットの動作を即座に停止する
- ロボットが割り付けられていて REACTE 操作が実行されていない場合に、ロボット制御プログラムの実行を停止して、プログラムのエラー処理を可能にします。

手動制御ペンダントの非常停止ボタンを押した場合と異なり、PANIC 操作を処理した後も高電圧は通電したままになります。

例

次のモニタコマンドの例では、外部 E-stop ボタン押下をシミュレートし、ロボット 2 をただちに停止しますが、ロボットのハイパワーは OFF にしません。

```
PANIC 2
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

ESTOP(モニタコマンド)

ESTOP プログラムコマンド）

PANIC （プログラムコマンド）

PARAMETER

システムパラメータの値を設定、または表示するモニタコマンドです。

構文

PARAMETER parameter[index] = value

使用上の留意事項

等号と値が省略されている場合、システムパラメータは変更されず、モニタウィンドウに現在の値が表示されます。

システムパラメータが指定されていない場合、モニタウィンドウにすべてのシステムパラメータの現在の値が表示されます。

システムパラメータに関する詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

parameter	値を表示または修正するシステムパラメータの名前です。システムパラメータの値を修正するコマンドに等号と値が含まれている場合、名前を指定する必要があります。システムパラメータの値の表示にコマンドを使用している場合、名前を省略できます。指定されている場合、システムパラメータの名前を短縮できます。
index	インデックスで条件を付けることができるシステムパラメータの場合、index は、特定のパラメータ要素を対象として指定するオプションの実数値、変数、式です。
value	システムパラメータに割り当てる値を定義するオプションの実数値、変数、式です。値を指定しない場合、等号を省略する必要があります。

詳細

値が指定されている場合、指定されたシステムパラメータは、等号の右側にある値に設定されます。パラメータの名称は一意に特定する最小長さに短縮することができます。

引数なしで PARAMETER コマンドを使用することで、eV+ システムで使用できるすべてのシステムパラメータのリストを表示することができます。システムパラメータ名を短縮し、値を入力しない場合、完全なリストのサブセットをリクエストできます。指定のルートで始まる名前のすべてのシステムパラメータと現在の値が表示されます（以下の例を参照）。

指定のパラメータがインデックスの修飾子を受け付け、インデックスが 0 または括弧の有無を問わず省略されている場合、パラメータ配列のすべての要素が修正または表示されます。

システムパラメータの名前が省略されているが、インデックスが指定されている場合、インデックスのないすべてのパラメータの値が、すべてのパラメータ配列の指定要素とともに表示されます。

例

次の例では、BELT.MODE システムパラメータを 4 に設定します。

```
PARAMETER BELT.MODE = 4
```

以下の例では、「B」で始まるシステムパラメータ名の現在の設定値を表示します。

```
PARAMETER B
```

関連キーワード

BELT.MODE

NOT.CALIBRATED

PARAMETER（プログラムコマンド）

PARAMETER（実数型関数）

PING

ノードに対してネットワーク接続をテストするモニタコマンドです。

構文

PING node

パラメータ

node 通信を試みるネットワークノード名または IP アドレスです。ノード名を使用する場合、eV+ 構成ファイルにおいて定義しているか、FSET モニタコマンドキーワードまたは FSET プログラム命令キーワードを使用して定義している必要があります。

詳細

このコマンドは、アドレスが割り当てられているノードに対してネットワーク接続をテストします。ノードが応答する場合、コマンドは「Success」を表示します。ノードが5秒以内に応答しない場合、コマンドは「Node not reachable」を表示します。

例

次の例では、ノード名「server2」との接続が成功した場合、メッセージ「Success」を返します。接続が失敗した場合、メッセージ「Node not reachable」を返します。

```
ping server2
```

次の例では、IP アドレス 172.16.200.1 のノードとの接続が成功した場合、メッセージ「Success」を返します。接続が失敗した場合、メッセージ「Node not reachable」を返します。

```
ping 172.16.200.1
```

関連キーワード

FSET（モニタコマンド）

FSET（プログラムコマンド）

NET

PRIME

実行するプログラムの準備をしますが、実行は開始しないモニタコマンドです。

構文

PRIME task program(param_list), cycles, step

使用上の留意事項

PRIME は、選択したプログラムの実行タスクの実行スタックをリセットし、そのタスクのために一時停止しているプログラムのコンテキストを取り消します。

選択したプログラムタスクがアクティブの間、PRIME コマンドの処理は実行できません。

外部フロントパネルキースイッチが AUTOMATIC モードに設定されている場合に限って、このコマンドを使用できます。

パラメータ

task	どのシステムプログラムタスクを起動するかを指定するオプションの整数値です。
program	実行対象のプログラム名です。名前が省略されている場合、選択されたタスクに対してEXECUTE モニタコマンド、プログラムコマンド、または PRIME モニタコマンドで指定された最後のプログラム名が使用されます。
param_list	コンマで区切られた、定数、変数または式のオプションのリストであり、指定したプログラムの .PROGRAM ステートメント内の引数のタイプおよび数に対応する必要があります。プログラムが引数を必要としない場合、リストを空白にし、括弧を省略できます。 省略されたパラメータをスキップする場合、コンマを使用してプログラムパラメータを省略できます。リストの最後のパラメータが省略されている場合、コンマは必要ありません。省略されたパラメータは呼び出されたプログラムに未定義として引き渡され、DEFINED 関数で検出することができます。
cycles	実行されるプログラム実行サイクルの数を指定する、整数と解釈されるオプションの実数値、変数または式です。省略した場合、サイクルカウントは 1 と仮定されます。無制限のサイクルについては任意のマイナス値を指定します。最大ループカウント値は、32767です。
step	実行プログラムを開始するステップを指定する、整数と解釈されるオプションの実数値、変数または式です。省略されている場合、プログラムの実行は、最初の空白とコメントの行およびすべての AUTO、GLOBAL、LOCAL ステートメントの後にあるプログラムの最初の実行可能なステートメントで開始します。

詳細

このコマンドは実行するプログラムの準備をします。実際に PRIME がプログラムの実行を開始していない場合を除き、EXECUTE モニタコマンドと同等とみなすことができます。

プログラムの準備が整った後、PROCEED コマンドで実行を開始できます。

例

次の例では、タスク 1 で「rob.move」プログラムを準備します。

```
PRIME 1 rob.move()
```

関連キーワード

EXECUTE （モニタコマンド）

EXECUTE（プログラムコマンド）

STOREP

XSTEP

PROCEED

アプリケーションプログラムの実行を再開するモニタコマンドです。

構文

PROCEED task

使用上の留意事項

実行プログラムが正常に終了した場合または HALT プログラムコマンドによって停止した場合、プログラムを再開することはできません。

パラメータ

task	実行するプログラムタスクを指定する、整数として解釈される実数値、変数、または式です。タスク番号を指定していない場合、タスク番号 0 とみなされます。

詳細

このコマンドは、PAUSE プログラムコマンド、ABORTモニタコマンド、ブレークポイント、シングルステップ実行またはランタイムエラーにより実行停止したステップの次のステップから、指定したプログラムタスクの実行を再開します。

中断されたプログラムの実行を継続する他に、このコマンドを使用して PRIME モニタコマンドで実行されるように作成されたプログラムの実行を開始することができます。

指定されたタスクが実行中で、そのプログラムが WAIT または WAIT.EVENT ステートメントにある（例えば、外部信号条件が満たされるのを待っている）場合、PROCEED コマンドを発行すると、その WAIT または WAIT.EVENT ステートメントをスキップします。

指定されたタスクが実行中で、プログラムが WAIT または WAIT.EVENT プログラムコマンドにない場合、このコマンドには効果がありません。

> **注:** PROCEED は RETRY とは異なります。プログラムステートメントがエラーを発生した場合、RETRY は、そのステートメントの再実行を試みますが、PROCEED は、次のステートメントから実行を再開します。プログラムの停止中にロボットのモーションが進行している場合、PROCEED が次のモーションに進めるように、RETRY はそのモーションを完了を試みます。

例

次の例では、タスク2 で一時停止しているプログラムの実行を再開します。

```
PROCEED 2
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

RETRY

SSTEP

STATUS

XSTEP

RENAME

メモリにあるユーザープログラムの名前を指定された新しい名前に変更するモニタコマンドです。

構文

RENAME new_program = old_program

使用上の留意事項

RENAME はメモリにあるプログラムで使用できます。プログラムが現在実行中の場合、エラー"Program Interlocked"が返されます。プログラムが停止しているが、スタックにある場合、そのプログラムはスタックから削除され（KILL と同様）、メモリ内で名前が変更されます。RENAME は、ディスクファイル名は変更せず、システムメモリに存在するプログラムの名前を変更します。

パラメータ

new_program プログラムの新しい名前です。

old_program プログラムの現在の名前です。

詳細

すでにシステムメモリ内に指定された新しい名前と同じ名前のプログラムが存在する場合、RENAME 命令は実行されず、エラーメッセージが表示されます。この場合、まず新しい名前と同じ名前の既存のプログラムを削除してから、RENAME 命令を実行する必要があります。

名前を変更するユーザープログラムがプログラムモジュールに現在割り当てられている場合、プログラムはモジュールから削除され、新しい名前でグローバルモジュールに割り当てられます。

プログラムモジュールに関する情報については、 MODULE コマンドを参照してください。

例

次の例では、プログラムの名前を「oldprog」から「newprog」に変更します。

```
RENAME newprog=oldprog
```

関連キーワード

COPY

FRENAME

RESET

すべてのデジタル出力信号をOFF するモニタコマンドです。

構文

RESET

使用上の留意事項

このモニタコマンドは、ホストI/O(外部)信号番号 4001~4999には影響しません。RESET プログラムコマンドは、プログラムの初期化の一部として、すべての外部出力信号を既知の状態する場合に役立ちます。



警告: すべての出力信号を安全に OFF できない限り、このコマンドを発行しないでください。信号がOFF のときにデバイスが起動する場合、特に注意が必要です。

関連キーワード

BITS(モニタコマンド)

BITS (プログラムコマンド)

BITS(実数値関数)

IO

SIG

SIG.INS

SIGNAL（モニタコマンド）

SIGNAL (プログラムコマンド)

RESET.LOCK

アプリケーションプログラムからロボットをデタッチするモニタコマンドです。

構文

RESET.LOCK

使用上の留意事項

このコマンドは、現在アタッチされているすべてのロボットをデタッチします。

関連キーワード

ATTACH

DETACH

RETRY

最後の割り込みステートメントの実行とプログラムの連続実行を繰り返すモニタコマンドです。

構文

RETRY task

使用上の留意事項

指定のタスクが実行中の場合、RETRY を処理できません。

実行プログラムが正常に終了した場合または HALT プログラムコマンドによって停止した場合、プログラムを再開することはできません。

パラメータ

task	実行するプログラムタスクを指定する、整数として解釈される実数値、変数、または式です。タスク番号を指定していない場合、タスク番号 0 とみなされます。

詳細

このコマンドは、PROCEED モニタコマンド同様、指定したタスクの実行を再始動します。RETRY コマンドの後、どの地点から実行が再開されるかは、その実行が中断されたときの状況によって変わります。プログラムステップまたはロボット動作が完了前に中断された場合、RETRY コマンドを使用すると、中断されたオペレーションを完了してから、実行が通常通り継続します。そのため、中断されたステップまたはエラーの原因となったステップを再試行することができます。

プログラムステップやロボット動作を中断していない場合、RETRY コマンドの効果は PROCEED コマンドのものと同じになります。

> **注:** RETRY コマンドで割り込まれたモーションを再開する場合、すべてのモーションパラメータは、モーションが割り込まれる前の設定で保存されます。

例

次の例では、（タスクが実行中ではない、または HALT 操作で停止されていない場合）「task.num」でタスク番号が定義されたタスクを、中断されたところから再開します。

```
RETRY task.num
```

関連キーワード

PROCEED

SSTEP

STATUS

XSTEP

SELECT

以降のモニタウィンドウでの操作のために、ロボットを選択するモニタコマンドです。

構文

SELECT device_type = unit

使用上の留意事項

同じ種類の複数のデバイスがシステムコントローラーに接続されている場合、SELECT コマンドを使用する必要があります。

SELECT コマンドが発行されていない場合、システムの最初のロボットを自動的に選択し、最小のロボット識別子（1）から開始します。

SELECT コマンドはモニタウィンドウのロボット選択だけに影響を与えます（HERE やWHERE など以降のモニタコマンド）。プログラムタスクでの選択を変更する場合は、そのプログラムタスクやプログラム内、または DO モニタコマンドを使用して SELECT プログラムコマンドを実行する必要があります。例えば、DO @1 SELECT ROBOT = 2 は、プログラムタスク 1 用にロボットの選択を変更します。

パラメータ

device_type	選択されるデバイスの種類を識別するキーワードです。唯一有効なデバイスの種類は、ROBOTです。
unit	選択される特定のロボットを指定し、整数値と解釈される実数値、変数、または式です。使用できる値は、システムの設定によって異なります。

詳細

複数のロボットシステムの場合、SELECT モニタコマンドはモニタウィンドウがアクセスするロボットを指定します。

unit パラメータに指定されたロボット番号が無効な場合、エラー「Invalid argument」（-407、無効な引数）が発生します。

unit パラメータに対して指定されたロボット番号は有効であるが、ロボットがシステム内で設定されていない場合、エラー「No robot connected to system」（-622、システムに接続しているロボットがありません）が発生します。

unit パラメータに対して指定されたロボット番号が有効で、ロボットがシステム内で設定されている場合、「Success」（成功しました）情報メッセージが発生します。

例

次の例では、以降のモニタウィンドウでの操作のためにロボット2 を選択します。

```
SELECT ROBOT = 2
```

関連キーワード

ATTACH

SELECT（プログラムコマンド）

SELECT

SIGNAL

デジタル出力信号、内部ソフトウェア信号またはホスト信号をオンまたはオフにするモニタコマンドです。

構文

SIGNAL signal, ..., signal

使用上の留意事項

SIGNAL コマンドを使用してホスト信号を制御するためには、Sysmac Studio を使用して NJ シリーズロボット統合CPU ユニット 内でホスト信号を 4001～4999 の範囲に割り当てる必要があります。詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

パラメータ

signal	デジタル出力信号、または内部信号の番号を評価する実数式です。正の値はオン、負の値はオフを示します。 SIGNAL は、値 0 のパラメータは無視します。

詳細

signal パラメータの値は、どの信号を制御するのかを決定しています。signal パラメータには次の値を使用できます。

- それぞれのロボットの XIO および IO Blox に接続するデジタル出力: 1～999
- グリッパなどエンドエフェクタのためのロボット出力: 3001～3004
- 内部ソフトウェア信号: 2001～2999
- ホスト信号: 4001～4999

複数のロボットが存在する場合のデフォルトの信号割り当てに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

エミュレーションモードの場合、SIGNAL コマンドでデジタル入力（1001～1999）も制御できます。

例

次の例では、（「reset」の値がプラスであるとすると）変数「reset」の値によって指定される外部出力信号をオフにして外部出力信号 4 をオンにします。

```
SIGNAL -reset, 4
```

次の例では外部出力信号 1 をオフにし、外部出力信号 4 をオンにし、内部ソフトウェア信号 2010 をオンにします。

```
SIGNAL -1, 4, 2010
```

関連キーワード

BITS（モニタコマンド）

BITS（プログラムコマンド）

BITS（実数値関数）

IO

RESET

RUNSIG

SIG

SIG.INS

SPEED

モニタ速度を指定するモニタコマンドです。

構文

SPEED speed_factor

使用上の留意事項

モニタ速度は、最大で 100 以下です。この制限以上の速度を設定した場合、100 とみなされます。値 100 は通常の全速力、50は全速力の半分とみなされます。

- ジョイント内挿モーションと直線モーションでは、モーション速度は異なる影響を示します。詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。
- SPEED 設定は、現在実行中の動作速度を含め、ただちに有効になります。

ロボットの動作速度は、SPEED モニタコマンドで設定されたモニタ速度とSPEED プログラムコマンドを使用した実行プログラムで設定されたプログラム速度の組み合わせで決定します。

eV+ システムがロボットを制御するように構成されていない場合、SPEED コマンドを実行するとエラーになります。

パラメータ

speed_factor 新しい速度係数として使用する値の実数式です。

詳細

ロボットの動作速度は、このコマンドで設定された速度と SPEED プログラムコマンドで設定された速度から決定します。連続パスモーションの間にプログラム速度が変更された場合、その後のパスは変更され、指定の速度と加速度を維持します。しかし、モニタ速度が変更されると、パスには影響せず、加速度が修正されます。

モニタ速度は設定されたプログラム速度の割合（%）として適用されます。プログラムや変数を修正することなく、ロボットの動作速度を調整できるため、非常に有用です。モーション速度、プログラム速度、加速度の関係を次に示します。

モニタ速度が 100% の場合、eV+ は指定のプログラム速度と加速度に達するための動作を生成します。連続パスモーションの間、加速度が過剰にならないように、パスは目的地の中間点近くて円状になります。プログラム速度が増加し、加速度が一定の状態の場合、中間地点での回転が増加し、指定の加速度を維持します。

モニタ速度が 100% 未満に設定されている場合、eV+ は 100% のモニタ速度に対して計画されたものと同じ経路を生成し、その丸めは変わりません。動作全体の速度を下げるために、動作の各部分の区間（加速区間、一定速度区間および減速区間）は比例的に加減されます。

> **注:** eV+ が初期化されると、モニタ速度は 50 に設定されます。速度は、0.000001 [1.0E-6] 未満に設定できません。

例

次の例では、モニタ速度を 30% に設定します。

```
SPEED 30
```

関連キーワード

SCALE.ACCEL

SPEED（プログラムコマンド）

SPEED （実数値関数）

SRV.RESET

eV+ を再起動またはリセットするモニタコマンドです。

構文

SRV.RESET

使用上の留意事項

このコマンドによってすべてのスイッチおよびパラメータ設定はデフォルト値にリセットされます。

> **注:** ロボットの EtherCAT 設定はリセットされません。

詳細

SRV.RESET コマンドを発行すると、eV+ がリセットされます。すべてのロボットハイパワーは無効になります。

次の状態ではコマンドの実行が防止されます。

- システム内のロボット数が現在の設定と異なる。
- 任意のロボットがタスクに割り付けられているか、任意のタスクが実行中である。
- 任意のロボットがエラー状態である。

設定済みのそれぞれのロボットの名前が、システムのリセット中に発生した可能性のあるエラーと共にモニタウィンドウに表示されます。

SSTEP

シングスステップ、または制御プログラムのサブルーチン全体を実行するモニタコマンドです。

構文

SSTEP task

使用上の留意事項

SSTEP（サブルーチンステップ）を使用して、他のシステムタスクの実行状態と独立した、使用可能なシステムプログラムタスクをシングルステップで実行できます。

パラメータ

task	どのシステムプログラムタスクを実行するかを指定するオプションの整数値です。タスク番号を指定していない場合、タスク番号 0 とみなされます。

詳細

実行予定の次のプログラムステップが CALL、または CALLS プログラムコマンドでない場合、SSTEP コマンドはプログラム引数がない XSTEP コマンドと同じように動作します。

実行予定のプログラムステップがCALL、またはCALLS プログラムコマンドの場合、SSTEP コマンドはCALL、またはCALLS プログラムコマンドの後のステップでプログラムの実行を停止する前に呼び出されたサブルーチン全体を実行します。

PROCEEDおよびRETRYコマンドと同様に、SSTEP コマンドは先行のプログラムステートメントのシングルステップ実行、PAUSE プログラムコマンド、ブレークポイント、またはプログラム実行中の致命的でないエラーの後でのみ実行できます。

シングルステップ実行の間、次に実行するステートメントは、モニタウィンドウと手動制御ペンダントに表示されます。

プログラムが呼び出されたルーチン内で停止した場合、および追加のXSTEP、SSTEPまたは PROCEED コマンドによってその地点で再始動した場合でも、SSTEP ステータスは eV+ によって保持されます。最終的に元のサブルーチンが RETURN プログラムコマンドを実行するときにプログラムの実行は停止します。

例

次の例では、タスク 0 として実行していたプログラムの次のステップを実行します。

```
SSTEP
```

次の例では、プログラムタスク 1 の次のステップを実行します。

```
SSTEP 1
```

関連キーワード

CALL

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

PROCEED

RETRY

STATUS

XSTEP

STACK

サブルーチンコールと自動変数に使用するためにプログラムタスク用に予約するシステムメモリの量を指定するモニタコマンドです。

構文

STACK task = size

使用上の留意事項

何らかのプログラムタスクがアクティブの場合、このコマンドは実行できません。

Sysmac Studioが接続されている間、モニタウィンドウを使用してこのコマンドを発行することはできません。通常、このコマンドはモニタコマンドプログラムのMC キーワードで使用されます。

エミュレーションモードで動作している間はこのコマンドを実行できません。

RETRY モニタコマンドを使用して、タスクの実行を続行できます。

タスクで現在使用しているスタックメモリの量より小さいスタックサイズを設定する場合、スタックサイズは、現在使用しているサイズに設定されます。

デフォルトのスタックサイズ（128 kB）を使用しないのであれば、eV+ システムを ディスクから起動するたびに、この値を設定する必要があります。この場合、スタックサイズの設定に、初期化コマンドプログラムを使用できます。

パラメータ

task	変更するスタックサイズのプログラムタスクを指定する、整数として解釈される実定数、変数、または式です。
size	予約するスタックサイズの量(キロバイト) を指定する実定数、変数、または式です。予約のバイト数は、サイズパラメータの値に 1024 を乗算して算出されます。

詳細

サブルーチンの呼び出しを行うと、eV+ はスタックと呼ばれる内部保存領域を使用して、実行を開始するサブルーチンが必要とする情報を保存します。この情報には、次の項目が含まれます。

- コールする側のプログラムの名前とステップ番号。
- サブルーチンの引数のアクセスに必要なデータ。
- コールされるプログラムで指定された自動変数の値。

STACK コマンドを使用すると、各プログラムタスクが使用するスタックの保存スペースを明示的に割り当てることができます。システムメモリの使用を最適化するために、特定のアプリケーション用にスタックスペースの量を調整できます。スタックは任意の大きさに作成できますが、システムで使用できるメモリ量にのみ制限されます。

STACK コマンドで必要なストレージ量を割り当てることができない場合、割り当てが失敗し、エラーメッセージ*Not enough storage area*が表示されます。エラーが発生した場合、次のステップに従います。これらのステップでは、プログラムの保存領域を圧縮するため、割り当てるスタックをより大きくできます。

1. 可能な場合、他のプログラムタスクのスタックサイズを縮小します。
2. ZERO モニタコマンドを発効して、メモリ内のすべてのプログラムを削除します。
2. 目的の STACK コマンドを発行します。
3. プログラムをリロードします。

プログラムタスクのスタックスペースが不足している場合、タスクは終了し、エラーメッセージ*Not enough program stack space*が表示されます。この場合、STACK モニタコマンドを使用してスタックサイズを増やし、RETRY モニタコマンドを発行して実行プログラムを継続します。他のすべてのプログラムタスクを停止する必要があります。

STATUS コマンドを使用して、単一のプログラムタスクのスタック統計を表示することができます。スタックの最大値は、エラーを発生したタスクに必要にスタックスペース量を示します。

例

次のステートメントでは、タスク 0 のスタックに 64 キロバイトのメモリを予約します。

```
STACK 0 = 64
```

関連キーワード

AUTO

STATUS

STATUS

システムと実行するプログラムのステータス情報を返すモニタコマンドです。

構文

STATUS select

使用上の留意事項

STATUS コマンドは、システムのステータスを判断するためにいつでも使用できます。

パラメータ

select	返される情報を選択する整数値として解釈されるオプションの実数値、変数、式です。

詳細

select パラメータが省略されている場合、すべてのプログラムタスクのステータスが返されます。

返される情報は連続的には更新されません。

select パラメータの値が 0 または正の値の場合、プログラムタスクの 1 つに対応する必要があります。

次の例で、select パラメータが省略されている場合に STATUS コマンドが返す情報について説明します。

```
ROBOT:  Robot power off        Monitor speed:  100

TASK    STATE           MAIN            CURRENT PROGRAM   STEP   CYCLES   STACK
0       Program Running ai.monitor.jobs ai.check.job      25          0     2.5
1       Program Running ai.monitor.jobs ai.check.jobs.cmp  7         10     2.0
2       Program Wait    ai.monitor.jobs rn.get.msg.wait   35          0     1.5
3       Not Active      ai.monitor.jobs ai.help.screen    35          0     0.0
```

ROBOT

ロボットが存在するシステムでのみ表示されます。上記に示す「ROBOT:」フィールドの場合、次のメッセージが考えられます。

- **致命的なエラー**

  致命的なハードウェアエラーが発生。ロボットの電源が OFF になり、エラーが解決するまで ON にできません。

- **Robot power off**

  ロボットの電源が OFF になり、必要な場合は ON にできます。

- **Not calibrated**

  ロボットの電源が ON なりますが、ロボットはキャリブレーションされず、CALIBRATE モニタコマンドまたはプログラムコマンドが発行されるまでロボットを移動できません。

- **COMP モード**

  ロボットの電源が ON なり、アプリケーションプログラムでの制御が有効になります。

- **手動モード**

  ロボットの電源が ON なり、ロボットは手動制御ペンダントで制御されます。

モニタ速度

返される情報の「モニタ速度：」フィールドに現在のモニタ速度係数が表示されます。

TASK

「TASK」列に実行中のすべてのタスクが一覧表示されます。

STATE

次に示すように、「ステート」フィールドには、各プログラムタスクの現在のステータスを示すメッセージが表示されます。

- **Not Active**

  現在、タスクが非アクティブです。

- **Program running**

  タスクが右側に示すプログラムを実行中です。

- **Program input**

  実行中のプログラムが I/O デバイスからの入力を待機しています。

- **Program WAIT**

  実行中のプログラムが WAIT プログラムコマンドステートメントで待機しています。

MAIN

「メイン」列は、実行中のメインプログラムが、EXECUTE モニタコマンドまたはプログラムコマンド、PRIME、XSTEP モニタコマンドで呼び出されたプログラムであることを示します。

CURRENT PROGRAM

「CURRENT PROGRAM」列のプログラムは、現在スタックの最上位にあるプログラムおよび現在実行中のプログラムです。メインプログラム、または後から

CALL、またはCALLS プログラムコマンド（または応答）で呼び出されたプログラムである可能性があります。

STEP

「STEP」列には、現在のプログラムで次に実行されるステップの番号が示されます。

CYCLES

「CYCLES」列には、STATUS コマンドが発行されたときに完了しているメインプログラムの実行サイクル数が示されます。

STACK

「STACK」列には、実行スタックの現在のサイズ（キロバイト）が示されます。

次の例で、select パラメータにプログラムタスクが指定されている場合に STATUS コマンドが返す情報について説明します。

```
STATE          PROGRAM        STEP        CYCLES       STACK    MAX  LIMIT
Not active     program0       82          0 of 1         0.0    0.0  128.0
```

このフォーマットの「ステート」、「プログラム」、「ステップ」フィールドは、上記説明の「ステート」、「現在のプログラム」、「ステップ」フィールドと類似しています。

選択したプログラムタスクがアクティブの場合、最上位のスタックにあるプログラムのみが表示されます。タスクステートが「非アクティブ」の場合、メインプログラムで始まり、最上位スタックで終わる、実行スタック全体が表示されます。

CYCLES

「CYCLES」列には、完了している総サイクル数に続いて完了予定の総サイクル数が示されます。値 -1 は、無限に実行されるサイクルを示します。

STACK

「STACK」列には、現在使用しているスタックサイズ（キロバイト）が示されます。

MAX

「MAX」列には、プログラムが最後に実行されてから使用されたスタックの最大量が示されます。プログラムが失敗し、エラーメッセージ「*Not enough program stack space*」（プログラムスタックスペース不足）が表示された場合、「MAX」フィールドは、オペレーティングシステムが必要とするスタックスペースの量を示します。タスクにスタックスペースを再割り当てするために使用する値を返します。

LIMIT

「LIMIT」フィールドは、スタックサイズの制限値を示します。この制限値は STACK モニタコマンドを使用して変更することができます。

例

次の例では、タスク 0 とタスク 1 が実行中、タスク 2 は 1 つのサイクルを完了し、実行していない、タスク 3 は非アクティブでスタックが空であることを示しています。

```
ROBOT:  Robot power off        Monitor speed:  100

TASK    STATE            MAIN              CURRENT PROGRAM    STEP   CYCLES   STACK
0       Program Running  ai.monitor.jobs   ai.check.job         25        0     2.5
1       Program Running  ai.monitor.jobs   ai.check.jobs.cmp     7       10     2.0
2       Program Wait     ai.monitor.jobs   rn.get.msg.wait      35        0     1.5
3       Not Active       ai.monitor.jobs   ai.help.screen       35        0     0.0
```

タスク 1 はリクエスト「EXECUTE 1 ai.monitor.jobs」によって開始されました（「メイン」列を参照）。次の実行ステップは、メインプログラム「ai.-monitor.jobs」から直接または間接的にコールされた、プログラム「ai.check-.job」のステップ 25 です。タスクはどのサイクルも完了しておらず、現在 2.5 キロバイトのスタックを使用しています。

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

KILL（モニタコマンド）

KILL （プログラムコマンド）

PROCEED（モニタコマンド）

RETRY

STACK

STATUS（実数値関数）

STORE

ディスクファイルのプログラムと変数を保存するモニタコマンドです。

構文

STORE /levels **file_spec** = program, ..., program

使用上の留意事項

プログラムの実行中、STORE コマンドを使用できます。実行中のプログラムを保存できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

プレシジョンポイントを定義したシステムよりも軸の数が少ないシステムで、プレシジョンポイントを読み込みおよび保存した場合、成分が損失します。

メモリの保護プログラムと読み取り専用プログラムは、保存できません。

このモニタコマンドは外部変数を保存しません。

パラメータ

/levels	プログラムパラメータが指定されている場合、プログラム参照レベルを決定するオプションの修飾子です。/level が省略されている場合、後述するように、すべてのプログラム参照が処理されます。また修飾子を例えば「/2」として指定した場合、プログラム参照の最初の2 つのレベルのみが処理されます。
file_spec	プログラムと変数を保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。 ファイルの拡張子が指定されていない場合、指定のファイル名に拡張子「.V2」が付加されます（ディスクファイルのみ）。
program	メモリ内のオプションのプログラム名です。

詳細

このコマンドは、指定のディスクファイルを作成し、ファイルに次の情報を保存します。

- 指定のプログラム。
- 指定のプログラムが直接または間接的に参照するすべてのサブルーチン（/levels 修飾子で制限されない限り）。

- 保存されているプログラムとサブルーチンが参照するすべてのグローバル（ロケーション、実数値、文字列）変数です。

プログラム名が指定されていない場合、メモリにあるすべてのプログラム、サブルーチン、グローバル変数がディスクファイルに保存されます。

このコマンドは個別のコマンドである STOREP、STOREL、STORER および STORES と同じ情報を保存しますが、STORE コマンドは他の 4 つと異なり 1 つのファイルしか作成しません。

ディスクに保存されているプログラム名がモニタウィンドウに表示されます。参照されるサブルーチンは自動的に保存されるため、コマンドラインで指定した以外の名前も表示されます。コマンドで使用した順番にかかわらず、プログラムはアルファベット順で保存されます。

例

次の例では、デフォルトのディスクユニットにファイル「F3.V2」を作成し、プログラムが参照するすべてのサブルーチンとグローバル変数とともに、2 つのプログラム「cycle」と「motor」を保存します。

```
STORE f3=cycle,motor
```

次の例では、デフォルトのディスクユニットにファイル「F3.V2」を作成し、プログラムが参照するすべてのサブルーチンとグローバル変数とともに、プログラム「cycle」とこのプログラムが直接コールするサブルーチン（これらのサブルーチンがコールするサブルーチンは含まず）のみを保存します。

```
STORE /2 f3=cycle
```

次の例では、ディスクユニット「D」にファイル「DEMO.V2」を作成し、メモリにあるすべてのプログラムとグローバル変数を保存します。

```
STORE D:demo
```

関連キーワード

DEFAULT

FCOPY

LOAD

STOREL

STOREM

STOREP

STORER

STORES

STOREL

ディスクファイルのロケーション変数を保存するモニタコマンドです。

構文

STOREL /levels **file_spec** = program, ..., program

使用上の留意事項

プログラムの実行中、STOREL コマンドを使用できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

プレシジョンポイントを定義したシステムよりも軸の数が少ないシステムで、プレシジョンポイントをロード、保存した場合、成分が損失します。

パラメータ

/levels	プログラムパラメータが指定されている場合、プログラム参照レベルを決定するオプションの修飾子です。/level が省略されている場合、後述するように、すべてのプログラム参照が処理されます。また修飾子を例えば「/2」として指定した場合、プログラム参照の最初の2 つのレベルのみが処理されます。
file_spec	プログラムと変数を保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。 ファイルの拡張子が指定されていない場合、指定されたファイル名に拡張子「.LC」が付加されます（ディスクファイルのみ）。
program	メモリ内のオプションのプログラム名です。

詳細

このコマンドは、/levels パラメータで制限されていない限り、指定されたプログラムとこれらのプログラムが参照するサブルーチンが参照する、すべてのグローバルロケーション変数の名前と値を保存します。プログラムが指定されていない場合、値が定義されたすべてのグローバルロケーション変数がディスクファイルに保存されます。

例

次の例では、プログラム「motor」とこのプログラムが参照するすべてのサブルーチンが参照する、すべてのグローバルロケーション変数をディスクファイル「F2.LC」に保存します。

```
STOREL f2=motor
```

関連キーワード

DEFAULT

FCOPY

LOAD

STORE

STOREM

STOREP

STORER

STORES

STOREM

指定されたプログラムモジュールをディスクファイルに保存するモニタコマンドです。

構文

STOREM /qualifiers **file_spec = module**

使用上の留意事項

プログラムの実行中、STOREM コマンドを使用できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

メモリの保護プログラムと読み取り専用プログラムは、保存できません。

パラメータ

/qualifiers

モジュールプログラムに加えて、複数のデータタイプを保存するために、3 つの修飾子まで組み合わせを指定できます。すべての修飾子が省略されている場合、モジュールプログラムのみが指定されたファイルに保存されます。

/L が指定されている場合、モジュールプログラムが直接参照するグローバルロケーション変数とプレシジョンポイント変数が、ファイルの最後にある .LOCATIONS セクションに保存されます。

/R が指定されている場合、モジュールプログラムが直接参照するグローバル実数値変数とダブル型変数が、ファイルの最後にある .REAL セクションと .DOUBLE セクションに保存されます。

/S が指定されている場合、モジュールプログラムが直接参照するグローバル文字列変数が、ファイルの最後にある .STRINGS セクションに保存されます。

file_spec

プログラムを保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。

必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。

ファイルの拡張子が指定されていない場合、指定されたファイル名に拡張子「.PG」が付加されます(ディスクファイルのみ)。

module

メモリ内のプログラムモジュール名です。

詳細

このコマンドは、指定されたプログラムモジュールにあるすべての非制限プログラムを指定されたディスクファイルに保存します。ディスクに保存されているプログラム名がモニタウィンドウに表示されます。プログラムは、モジュール内での順番に従って保存されます。

例

次の例では、プログラムモジュール「main」にあるすべてのプログラムをディスクファイル「LINE23.PG」に保存します。

```
STOREM line23=main
```

関連キーワード

DEFAULT

FCOPY

LOAD

MDIRECTORY

MODULE

STORE

STOREP

STOREP

プログラムファイルをディスクファイルに保存するモニタコマンドです。

構文

STOREP /levels **file_spec** = program, ..., program

使用上の留意事項

プログラムの実行中、STOREP コマンドを使用できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

メモリの保護プログラムと読み取り専用プログラムは、保存できません。

> **注:** 一般的に、プログラムをモジュールにグループ化することは、優れたプログラミング習慣なので、STOREP コマンドの代わりに STOREM を一般的に使用すべきです。

パラメータ

/levels	プログラムパラメータが指定されている場合、プログラム参照レベルを決定するオプションの修飾子です。/level が省略されている場合、後述するように、すべてのプログラム参照が処理されます。また修飾子を例えば「/2」として指定した場合、プログラム参照の最初の2 つのレベルのみが処理されます。
file_spec	プログラムを保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。 ファイルの拡張子が指定されていない場合、指定されたファイル名に拡張子「.PG」が付加されます（ディスクファイルのみ）。
program	メモリ内 のオプション のプログラム名 です。

詳細

このコマンドは、指定されたプログラムを指定されたディスクファイルに保存します。指定されたプログラムに加えて、/levels パラメータで制限されていない限り、これらのプログラムが参照するサブルーチンとサブルーチンが参照するサブルーチンも自動的に保存されます。プログラム名が指定されていない場合、メモリにあるすべてのプログラムがディスクファイルに保存されます。

ディスクに保存されているプログラム名がモニタウィンドウに表示されます。参照されるサブルーチンは自動的に保存されるため、コマンドラインで指定した以外の名前も表示されます。コマンドで使用した順番にかかわらず、プログラムはアルファベット順で保存されます。

例

次の例では、プログラム「test」とこのプログラムが参照するすべてのサブルーチンをディスクファイル「F1.NEW」に保存します。

```
STOREP f1.new=test
```

関連キーワード

DEFAULT

FCOPY

LOAD

STORE

STOREL

STOREM

STORES

STORER

実数値変数をディスクファイルに保存するモニタコマンドです。

構文

STORER /levels **file_spec** = program, ..., program

使用上の留意事項

プログラムの実行中、STORER コマンドを使用できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

パラメータ

/levels	プログラムパラメータが指定されている場合、プログラム参照レベルを決定するオプションの修飾子です。/level が省略されている場合、後述するように、すべてのプログラム参照が処理されます。また修飾子を例えば「/2」として指定した場合、プログラム参照の最初の2 つのレベルのみが処理されます。
file_spec	変数を保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。 ファイルの拡張子が指定されていない場合、指定されたファイル名に拡張子「.RV」が付加されます（ディスクファイルのみ）。
program	メモリ内 のオプション のプログラム名 です。

詳細

このコマンドは、/levels パラメータで制限されていない限り、指定されたプログラムとこれらのプログラムが参照するサブルーチンが参照する、すべてのグローバル実数値変数の名前と値を保存します。プログラムが指定されていない場合、定義されたすべてのグローバル実数値変数がディスクファイルに保存されます。

> **注:** システムパラメータは実数値を持つことができますが、STORER コマンドで保存されません。

例

次の例では、プログラム「cycle」とこのプログラムが参照するすべてのサブルーチンが参照する、すべてのグローバル実数値変数をディスクファイル「F2.RV」に

保存します。

```
STORER f2=cycle
```

関連キーワード

DEFAULT

FCOPY

LOAD

STORE

STOREL

STOREM

STOREP

STORES

STORES

文字列変数をディスクファイルに保存するモニタコマンドです。

構文

STORES /levels **file_spec** = program, ..., program

使用上の留意事項

プログラムの実行中、STORES コマンドを使用できます。

新しいディスクファイルを保存するために、ディスクに十分な空き領域が必要です。空き領域がない場合、保存動作は失敗します。

パラメータ

/levels	プログラムパラメータが指定されている場合、プログラム参照レベルを決定するオプションの修飾子です。/level が省略されている場合、後述するように、すべてのプログラム参照が処理されます。また修飾子を例えば「/2」として指定した場合、プログラム参照の最初の2 つのレベルのみが処理されます。
file_spec	変数を保存するディスクファイルを指定します。file_spec は、オプションの物理デバイス、ディスクユニット、ディレクトリパス、ファイル名、オプションの拡張子で構成されます。 必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。 ファイルの拡張子が指定されていない場合、指定されたファイル名に拡張子「.ST」が付加されます（ディスクファイルのみ）。
program	メモリ内のオプションのプログラム名です。

詳細

このコマンドは、/levels パラメータで制限されていない限り、指定されたプログラムとこれらのプログラムが参照するサブルーチンが参照する、すべてのグローバル文字列変数の名前と値を保存します。プログラムが指定されていない場合、定義されたすべてのグローバル文字列変数がディスクファイルに保存されます。

特定の特殊文字を保存するために使用されるフォーマットに関しては LISTS コマンドを参照してください。

例

次の例では、システムメモリにあるすべてのグローバル文字列変数をディスクファイル「F3.ST」に保存します。

```
STORES f3
```

関連キーワード

DEFAULT

FCOPY

LOAD

STORE

STOREL

STOREM

STOREP

STORER

SWITCH

モニタウィンドウにシステムスイッチの設定値を表示するモニタコマンドです。

構文

SWITCH switch[index]

使用上の留意事項

switch パラメータが指定されていない場合、すべてのスイッチの現在の設定値を表示します。

システムスイッチに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

switch	表示対象のスイッチ名です。次に示すように、スイッチ名を短縮できます。
index	インデックスで条件を付けることができるスイッチの場合、index は、特定のスイッチ要素を対象として指定するオプションの実数値、変数、式です。

詳細

このコマンドは指定されたスイッチの設定値を「On」（有効）または「Off」（無効）で示します。switch 名が指定されていない場合、すべてのスイッチの状態を表示します。

スイッチ名を短縮することで、完全なリストのサブセットをリクエストできます。指定されたルートで始まる名前のすべてのスイッチと現在の値が表示されます。

指定されたスイッチがインデックスの修飾子を受け付け、インデックスが 0 または（括弧の有無を問わず）省略されている場合、スイッチ配列のすべての要素が表示されます。

スイッチ名が省略されているが、インデックスが指定されている場合、インデックスのないすべてのスイッチの値が、すべてのスイッチ配列の指定要素とともに表示されます。

例

次の例では、CP スイッチの現在の設定値を表示します。

```
SWITCH CP
```

関連キーワード

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

ENABLE

ENABLE（プログラムコマンド）

SWITCH（プログラムコマンド）

SWITCH（実数値関数）

TESTP

システムメモリに名前付きプログラムが存在するかをテストするモニタコマンドです。

構文

TESTP program

パラメータ

program 検索対象のプログラム名です。

詳細

このコマンドは基本的に モニタコマンドプログラム において有効です。プログラムが見つかった場合、成功メッセージが出力されます。それ以外の場合は、エラーが返されます。

例

次の例では、メモリにプログラム「move」がない場合、エラーメッセージを表示します。

```
TESTP move
```

関連キーワード

DIRECTORY

STATUS

TOOL

ロボットのツール取付フランジに対するツール先端のロケーションと方向を示すために使用される内部変換を設定するモニタコマンドです。

構文

TOOL @task:program transform_value

使用上の留意事項

TOOL コマンドは、eV+ モニタが SELECT モニタコマンドを使用して選択したロボットに適用されます。

プログラムの実行中、このコマンドを使用できます。しかし、ロボットが実行中のプログラムによりアタッチされている場合、エラーが発生します。

eV+ システムがロボットを制御するように構成されていない場合、TOOL コマンドを実行するとエラーになります。

パラメータ

@task:program	これらのオプションのパラメータは、コマンドが参照する変数のコンテキストを指定します。変数は、指定されたコンテキストから参照されるかのように処理されます。コンテキストが指定されていない場合、変数はグローバル変数とみなされます。変数コンテキストに関する詳細情報については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。
transform_value	新しいツール変換になるオプションの変換変数、関数、複合変換式です。変換値が省略されている場合、ツールは NULL に設定されます。

詳細

変換値が指定されていない場合、ツール変換は null ツールと等しい値に設定されます。Null ツールは、ツール取付フランジ表面の中心にあり、座標軸は、変換 [0,0,0,0,0,0] で表されるロボットの最後のジョイント軸と平行になります。システムを ON にして、ZERO モニタコマンドの後、ツール変換は自動的に null ツールと等しい値に設定されます。

変換が定義する位置にロボットを移動するためにコマンドが発行されるときやワールドモードやツールモードで手動制御動作が実行されるときに、ロボットの位置が要求されるたびに相対ツール変換が自動的に考慮されます。ツール変換の定義方法については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

TOOL コマンドが発行された後、このコマンドの引数として指定された変換値が変更された場合、この変更は、別の TOOL コマンドが発行されるまでロボットの動作に影響しません。例えば、変換値が変換変数で指定された場合、その変数値の変更は、別の TOOL コマンドがその変数で発行されるまで、ツール変換に影響しません。

> **注:** 「LISTL TOOL」モニタコマンドステートメントを使用して、現在のツール設定を表示できます。

例

次の例では、現在のツール変換を複合変換「grip:extension」の値に置き換えます。

```
TOOL grip:extension
```

次の例では、有効なツール変換を取り消します。

```
TOOL
```

関連キーワード

SELECT（モニタコマンド）

SELECT（実数値関数）

TOOL（プログラムコマンド）

TOOL（変換関数）

WAIT.START

条件が満たされるまで モニタコマンドプログラム を待機状態にするモニタコマンドです。

構文

WAIT.START condition

使用上の留意事項

このコマンドは、モニタウィンドウ用ではありません。これは通常、モニタコマンドプログラム におけるステップとして使用されます。

外部フロントパネルの非常停止ボタン、またはペンダントの非常停止ボタンを押して、起動した WAIT.START コマンドを取り消すことができます。

起動した WAIT.START コマンドを停止すると、このコマンドを含むモニタコマンドプログラムは終了します。

モニタコマンドプログラム に関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

condition	TRUE(0 以外) または FALSE(0) として評価されるオプションの実数値、変数、式です。

詳細

このコマンドを使用すると、目的の条件になるまで モニタコマンドプログラム の処理を中断することができます。例えば、継続状態として複数の信号の状態を使用できます。

WAIT.START コマンドに condition パラメータが含まれている場合、その condition の値が FALSE（0）から TRUE（0 以外）に移行するまで モニタコマンドプログラム は中断します。

> **注:** WAIT.START コマンドを実行するとき、テスト条件がすでに TRUE の場合、モニタコマンドプログラム は中断します。FALSE から TRUE に移行すると、再開します。

WAIT.START コマンドは、すべてのシステムサイクルに対して一度だけ、指定された条件を確認します。条件が満たされてからプログラムの実行を再開するまで、最大で 16 ms 遅れる可能性があります。

例

次の例では、デジタル入力信号 1001 が OFF から ON に変化するまで、モニタコマンドプログラムの処理を停止します。

```
WAIT.START SIG(1001)
```

関連キーワード

WAIT

WHERE

ロボットの現在のロケーションとハンドオープニングを表示するモニタコマンドです。

構文

WHERE

使用上の留意事項

WHERE コマンドは、eV+ モニタが SELECT コマンドを使用して選択したロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、WHERE コマンドを実行するとエラーになります。

詳細

ロボットツールポイントのロケーションが、デカルトワールド座標で現在のハンドオープニングとともにジョイントのポジションとして表示されます。

例

次の例に、4 軸ロボットに対して WHERE コマンドを発行したときに表示される出力を示します。

```
.WHERE
Robot 1:
      X mm        Y mm        Z mm        y           p           r          Hand
   550.001       0.003     384.000       0.000     180.000     180.000     1.00
0
      J1          J2          J3          J4          J5          J6
  -42.893      93.372      10.000      50.479
```

関連キーワード

HERE

SELECT（モニタコマンド）

SELECT（実数値関数）

XSTEP

プログラムをシングルステップで実行するモニタコマンドです。

構文

XSTEP task program (param_list), cycles, step

使用上の留意事項

XSTEP を使用して、他のシステムタスクの実行状態と独立した、システムプログラムタスクをシングルステップで実行できます。

パラメータ

task	どのシステムプログラムタスクを実行するかを指定するオプションの整数値です。タスク番号が指定されていない場合、タスク 0 とみなされます。
program	実行対象のアプリケーションプログラム名です。 **注:** XSTEP コマンドにプログラム名が含まれていない場合、一時的なタイムスライスと優先パラメータには影響しません。
param_list	プログラムのオプションのパラメータリストです。EXECUTE モニタコマンドとプログラムコマンドの詳細を参照してください。
cycles	実行されるプログラム実行サイクルの数を指定する、整数と解釈されるオプションの実数値、変数または式です。EXECUTE モニタコマンドとプログラムコマンドの詳細を参照してください。
step	実行プログラムを開始するステップを指定する、整数と解釈されるオプションの実数値、変数または式です。

詳細

XSTEP コマンドを使用して、アプリケーションプログラムを一度に 1 ステップごと実行できます。プログラムの開発中やプログラムエラーの検出と修正に大いに役立ちます。

このコマンドを使用する場合、プログラム実行について次の側面を考慮する必要があります。

- 使用するシステムプログラムタスク
- 実行するプログラム
- 1 つのコマンドの後でプログラムの実行が停止します。

どのシステムプログラムタスクを起動するかを、オプションのタスクパラメータで指定します。

プログラムの引数（program、param_list、cycles、step）が指定されている場合、プログラムの実行は EXECUTE モニタコマンドと同様に開始されます。EXECUTE モニタコマンドとは異なり、モニタウィンドウに最初の実行可能なプログラムステートメントが表示されますが、実行されません。そのステートメントを実行するために、プログラムの引数を指定せずに再度、XSTEP を入力する必要があります。

すべての引数が省略されている場合、次の動作が実行されます。

1. 表示されたプログラムステートメントを実行。
2. 実行予定の次のステートメントをモニタウィンドウに表示。
3. プログラムは、再度停止します。

PROCEED および RETRY モニタコマンド同様、次のいずれかのイベントのため実行が停止した後に限って、引数なしで XSTEP コマンドを実行できます。

- ABORT キーワード動作が処理された。
- 以降のプログラムステートメントを単ステップで実行した。
- PAUSE コマンドを実行した。
- ブレークポイントに到達した。
- プログラムの実行中に致命的でないエラーが発生した。

例

次の例では、プログラム「pack」3 サイクルの実行をタスク0 として開始します。パラメータ「p2」と「17」がプログラムに渡されます。以降の XSTEP コマンド（パラメータなし）の実行を予測し、「pack」の最初の実行可能ステップが表示されます。

```
XSTEP pack(p2,17),3
```

次の例では、プログラムタスク 0、またはステップ番号 23 から開始する現在のデバッグタスクとして実行されるプログラム「assembly」を作成します。「XSTEP」を次のように入力した場合、ステップ 23 が実行されます。

```
XSTEP assembly,,23
```

次の例では、プログラムタスク 2 として実行するプログラムの次のステップを実行します。

```
XSTEP 2
```

次の例では、タスク 0（または現在のデバッグタスク）として実行したプログラムの次のステップを実行します。

```
XSTEP
```

次の例では、タスク 0（または現在のデバッグタスク）として実行したプログラムのステップ番号 45 に実行ポインターを移動します。

```
XSTEP ,,45
```

関連キーワード

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

PROCEED

SSTEP

STATUS

ZERO

eV+ システムを初期化して、システムメモリにあるすべてのプログラムとデータを削除するモニタコマンドです。

構文

ZERO

使用上の留意事項

このコマンドは、プログラムタスクが実行している場合、またはロボットがタスクに割り付けられている場合は使用できません。

現在選択されているロボットはこのコマンドの影響を受けません。

コントローラーがオンラインの間、このコマンドは使用できません。詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

詳細

このコマンドは eV+ システムを初期化して、システムメモリにあるすべてのプログラムとデータを削除します。

コマンドの処理中に、次の変化が発生します。

- メモリにあるすべてのプログラムと変数は削除されます。
- プログラム実行タスクはクリアされます。
- ステータラインはクリアされます。
- すべてのラッチデータバッファはクリアされます。
- ZERO オペレーションの後にシステムを ON にすると、ツール変換は自動的にヌルツールと等しい値に設定されます。
- ZERO オペレーションの後にシステムを ON にすると、BASE 変換は自動的にヌル変換に等しく設定されます。
- すべての論理ユニット（ロボットを除く）は切り離されます。
- SPEED、FINE などの動作パラメータはデフォルトにリセットされます。その他のパラメータおよびスイッチは影響を受けません。

例

次の例では、システムメモリにあるすべてのプログラムとデータを削除します。

```
ZERO
```

関連キーワード

DELETE

DELETEL

DELETEM

DELETEP

DELETER

DELETES

3.3 その他のキーワード

本項の情報を参考にして、その他のキーワードと eV+ システムでのそれらの使用方法について理解してください。

.END

eV+ プログラムの終了を示すためのキーワードです。

構文

.END

使用上の留意事項

eV+ エディタは、すべてのプログラムの末尾にこの行を自動的に追加します。

このキーワードはすべての eV+ プログラムの末尾に存在する必要があります。.END 命令がないプログラムは、eV+ システムに正しく読み込まれません。

例

次の例では、モニタウィンドウに文字列「Valid Program」を表示します。

```
.PROGRAM program1()
    TYPE "Valid Program"
.END
```

関連キーワード

.PROGRAM

IPS

SPEED プログラムコマンドの単位を inch/s として指定します。

構文

SPEED value IPS ALWAYS

使用上の留意事項

IPS は、SPEED プログラムコマンドのパラメータとしてのみ使用できます。

指定された速度設定は、ロボットモーションが発生したときに有効となるモニタ速度でスケーリングされます。

IPS キーワードで指定された速度は、直線モーションに適用されます。ジョイント補間モーションでは、指定されたツール速度は維持されません。

速度を mm/s で指定するには、MMPSキーワードを使用します。

詳細

IPS は、速度値に使用する単位を指定するSPEEDプログラムコマンドのオプションのパラメータです。IPS が、SPEEDプログラムコマンドで指定された場合、速度値は、直線モーションの inch/sとして解釈されます。

IPS キーワードを使用したモーション速度設定の詳細については、SPEEDプログラムコマンドを参照してください。

例

次の例では、次のロボットの直線モーションに対するロボットツールの先端速度を 20 inch/s に設定しています（モニタ速度設定 を 100 と仮定）。

```
SPEED 20 IPS
```

関連キーワード

MMPS

SPEED（モニタコマンド）

SPEED

SPEED （実数値関数）

MMPS

SPEED プログラムコマンドの単位を mm/s として指定します。

構文

SPEED value MMPS ALWAYS

使用上の留意事項

MMPS は、SPEED プログラムコマンドのパラメータとしてのみ使用できます。

指定された速度設定は、ロボットモーションが発生したときに有効となるモニタ速度でスケーリングされます。

MMPS キーワードで指定された速度は、直線モーションに適用されます。ジョイント補間モーションでは、指定されたツール速度は維持されません。

速度を inch/s で指定するには、IPSキーワードを使用します。

詳細

MMPS は、速度値に使用する単位を指定するSPEEDプログラムコマンドのオプションのパラメータです。MMPS が、SPEEDプログラムコマンドで指定された場合、速度値は、直線モーションの mm/s として解釈されます。

MMPS キーワードを使用したモーション速度設定の詳細については、SPEEDプログラムコマンドを参照してください。

例

次の例では、直線モーションのデフォルトプログラム速度を 10mm/s に設定しています（モニタ速度設定を 100 と仮定）。

```
SPEED 10 MMPS ALWAYS
```

関連キーワード

IPS

SPEED（モニタコマンド）

SPEED

SPEED （実数値関数）

3.4 プログラムコマンドキーワード

このセクションの情報を使用して、プログラムコマンドキーワードとその eV+ システム内での使用について理解してください。

ABORT

実行可能なプログラムタスクの実行を終了します。

構文

ABORT task_num

使用上の留意事項

指定タスクとしてプログラムが実行されていない場合、ABORT は無視されます。

ABORT はディスク論理ユニット上で DETACH または FCLOSE オペレーションを強制するものではありません。このプログラムに 1 つ以上の開かれたファイルがあり、あなたが実行プログラムを再開しないことを決断した場合、KILL キーワードを使用してすべてのファイルを閉じ、論理ユニットを外してください。

パラメータ

task_num	終了するプログラムタスクを指定するオプションの実数値、変数または式です。整数として解釈されます。デフォルトタスクは 0 です。

詳細

ABORT プログラムコマンドは現在実行中のステップを終了したら、指定された有効実行可能プログラムの実行を終了します。このタスクがロボットを制御している場合、ロボットの動作は現在の動作が完了すると終了します。PROCEED キーワードを使用すると実行プログラムを再開することができます。

例

次の例では、位置「safe.loc」に移動した後、タスク 1 のプログラムの実行を終了します

```
MOVE safe.loc
BREAK
ABORT 1
```

関連キーワード

ABORT（モニタコマンド）

CYCLE.END（モニタコマンド）

CYCLE.END （プログラムコマンド）

ESTOP(モニタコマンド)

ESTOP （プログラムコマンド）

EXECUTE

KILL（モニタコマンド）

KILL （プログラムコマンド）
PANIC（モニタコマンド）
PANIC （プログラムコマンド）
PROCEED
STATUS（モニタコマンド）
STATUS（実数値関数）

ABOVE

次の動作に関するロボットの設定変更を要求して、肘が肩から手首のラインより上になるようにします。

構文

ABOVE

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがABOVE 設定をサポートしていない場合、ロボットはこのキーワードを無視します。

> **追加情報:** スカラおよびパラレルロボットにはABOVE設定することはできません。

ABOVE プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。ロボットが割り付けられていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、ABOVE プログラムコマンドを実行するとエラーになります。

以下の図にABOVE およびBELOW 設定を示します。

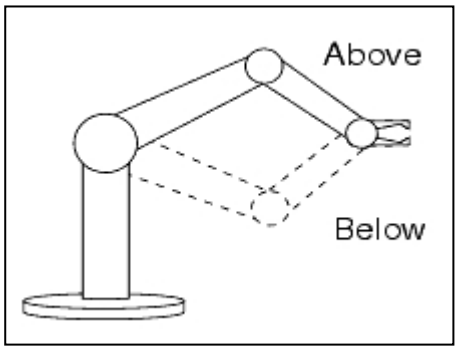


図3-5. ABOVE およびBELOW 設定

例

次の例では、ABOVE 設定で「point1」にロボットを移動します。

```
ABOVE
MOVE point1
BREAK
```

関連キーワード

BELOW

CONFIG

SELECT（プログラムコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

ACCEL

ロボット動作の加減速度を設定し、オプションとして定義された加速プロファイルを指定します。

構文

ACCEL (profile) acceleration, deceleration

使用上の留意事項

ACCEL プログラムコマンドは、そのタスクで選択したロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行するとエラー *Robot not attached to this program* が発生します。

加減速プロファイルを使用できるようにするためには、選択したロボットに対してプロファイルを定義する必要があります（プロファイル 0 が常に定義されています）。

定義されていないの加速度プロファイルがプログラムで当初に指定された場合、eCobraおよびViperロボットにシステムはデフォルトの加速度プロファイル2になります。

定義されていないの加速度プロファイルがプログラムで当初に指定された場合、iX3およびiX4ロボットにシステムはデフォルトの加速度プロファイル2になります。

定義されている加速度プロファイル（例えば 2）をプログラムにおいて指定した後、同じプログラムにおいて存在していない加速度プロファイル（例えば、7）を指定すると、そのプログラムは前回指定したプロファイル（この場合は2）を引き続き使用します。

パラメータを省略すると、現在の設定が有効なままになります。

パラメータ

profile	使用する加速プロファイルを指定するオプションの整数値です。受入可能な値は、定義されたプロファイル数に応じて 0～8 です。デフォルトは最後に指定したプロファイルになります。
acceleration	最大可能加速度のパーセンテージとみなされるオプションの実数値、変数、または式です。
deceleration	最大可能減速度のパーセンテージとみなされるオプションの実数値、変数、または式です。 この値は通常 1～100 の範囲にする必要があります。範囲外の値を指定すると、最も近い限度値が使用されます。

詳細

プロファイル 0 を使用すると、その動作の開始時と終了時に方形波加速プロファイルが生成されます。

プロファイルを指定している場合、その後のロボット動作にはそのプロファイルが呼び出されます。定義済みのプロファイルは、加減速度の最大変化率を設定します。このコマンドで設定される値は、到達する加減速の最大規模を定義します。

eV+ システムを初期化すると、プロファイル、加速度、減速度の値は初期値に設定されます。最初に選択されているプロファイルは、ロボットのタイプに応じて、0、1、または 2 です。実行プログラムが開始または終了する場合または ZERO モニタコマンドを実行する場合はこの設定に影響はありません。

一般的なロボットのペイロードおよびリンクイナーシャの使用に対しては、デフォルトの加減速値である 100% が設定されます。しかし、実際に到達可能な値は、エンドエフェクタ、ペイロード、および動作の始点／終点を変数とする関数として大きく変わるので、対象のロボットに対して 100% を超える加速度が許容される可能性があります。許容されるものよりも高い加減速値を指定した場合、デフォルト値が使用されます。

ACCEL(3) および ACCEL(4) 関数ステートメントを使用して最大許容加速度および減速度の設定をすることができます。

所定の動作に対して、最大到達加速度は所要加速度よりも低いことがあります。これは、非ゼロ加速度ランプ時間を持つプロファイルを使用し最大加速度に到達するための時間が不十分である場合に生じます。加速度が 0 から指定された最大値まで変化するまでには所定の時間が経過する必要があります。最大加速度に到達できない場合、台形プロファイルは三角形の形状に減少します。これは以下の状態で発生します。

- 動作が短時間過ぎる。この場合、最大加速度に到達する前に位置変化が完了します。
- 最大動作速度が低すぎる。この場合、最大加速度に到達する前に最大速度に到達します。

これらどちらの状態においても、最大加速度および最大減速度の値を上げても、動作時間に影響はありません。

> **追加情報:** 最大加速度および最大減速度の値を上げたが、動作時間は変更していない場合、プログラム速度を上げ、より早く加速立ち上げ時間を可能にする加速度プロファイルに変更するか、方形波加速プロファイルを指定する加速度プロファイル 0 への変更します。
>
> **注:** 方形波加速では加速度の値が上昇することなく瞬時に変わるので、この種類の加速制限は加速度プロファイル 0 では発生することがありません。

例

以下の例ではデフォルトの加速時間を通常の 50% および 減速時間を通常の 30% に設定しています。

```
ACCEL 50, 30
```

以下の例では減速時間を通常の 60% にし、加速時間は変更していません。

```
ACCEL ,60
```

以下の例では加速および減速を現在の設定の 1/2 に減らしています。

```
ACCEL ACCEL(1)/2, ACCEL(2)/2
```

以下の例ではプロファイル 2 を呼び出し、加速の規模を定義した定格の 80% に設定しています。

```
ACCEL (2) 80
```

関連キーワード

ACCEL（実数値関数）

DURATION

SCALE.ACCEL

SELECT

SELECT（実数値関数）

SELECT（モニタコマンド）

SPEED

ALIGN

ロボットツール Z 軸を最も近いワールド軸に合わせます。

構文

ALIGN

使用上の留意事項

ALIGN プログラムコマンドは、タスクがロボットに割り付けてあれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

システムがロボットを制御するように構成されていない場合、このコマンドを実行するとエラーになります。

詳細

ALIGN プログラムコマンドは、ツールの Z 軸をワールド座標系の最も近い軸に平行に合わせるように、ツールを回転します。このコマンドは、主に一連の位置をティーチングする前にツールの位置を合わせるときに役立ちます。このためには通常DOモニタコマンドと共に使用します。

例

次の例では、ロボットを「point1」に移動し、ツールの Z 軸を最も近いワールド軸に合わせます。

```
MOVE point1
ALIGN
```

関連キーワード

SELECT (プログラムコマンド)

SELECT（モニタコマンド）

SELECT（実数値関数）

ALTER

次の軌道計算の間にロボットのパスに適用されるリアルタイムパス修正の大きさを指定します。

構文

ALTER Dx, Dy, Dz, Rx, Ry, Rz

使用上の留意事項

このプログラムコマンドは、変更モードでロボットを制御しているタスクやそのロボットを選択（SELECT キーワードを使用）した他のタスクによって実行できます。

> **追加情報:** 変更モードの有効化についてはALTON プログラムコマンドを参照してください。

選択したロボットが変更モードではない場合、このコマンドは無視されます。

変更モードを有効にした場合、各軌道サイクル中にこのコマンドを 1 回実行する必要があります。このコマンドをそれ以上頻繁に実行した場合、各サイクル中に指定された最後の値のみ使用されます。

パラメータ

Dx Dy Dz Rx Ry Rz	X、Y、Z 軸に沿った移動距離および回転角度を定義する実数値、変数、または式です。オプションです。 累積モードでは、省略された座標は 0 として解釈されます。非累積モードでは、省略された座標は前回の ALTER コマンドで指定された値に設定されます。 距離はミリメートル、角度は度として解釈されます。

詳細

ALTON プログラムコマンドで変更モードを有効にした後、パスの修正量を指定するために、このコマンドを軌道ジェネレーターサイクルごとに 1 回実行する必要があります。このコマンドによって定義された座標は、変更モードを開始した ALTON プログラムコマンドによって指定されたモードに従って解釈されます。

例

次のステートメントは、センサデータを使用してロボットの動作を制御するプログラムループに組み込むことができます。

```
ALTER 0.1*sx, , 0.2*sz

WAIT
```

関連キーワード

ALTOFF

ALTON

STATE

ALTOFF

リアルタイム経路修正モード（変更モード）を終了します。

構文

ALTOFF

使用上の留意事項

このプログラムコマンドを実行する前に、ロボットを割り付ける必要があります。

変更モードを終了すると、連続経路動作は中断します。

詳細

このプログラムコマンドは、これ以前のロボット動作が完了するまでプログラムの実行を中断し、リアルタイム経路修正モード（変更モード）をオフにします。変更モードがオフになると、ロボット位置は最後のロボット動作の目標位置およびそれまでに適用された変更修正の総計の両方を反映した最終位置になります。

関連キーワード

ALTER

ALTON

STATE

ALTON

リアルタイムパス修正モード（変更モード）を有効にし、変更座標情報を解釈する方法を指定します。

構文

ALTON mode

使用上の留意事項

このプログラムコマンドを実行する前に、ロボットをアタッチする必要があります。

このプログラムコマンドを実行するときに変更モードがアクティブであってはいけません。

変更モードが有効な間に実行される動作は、直線動作タイプにする必要があり、コンベアベルトに対して指定することはできません。

パラメータ

mode — 以降のALTER コマンドによって指定されるパス修正データをどのように解釈するかを定義する実数値、変数、または式です。オプションです。モード値は次に示すように連続するビットフラグと解釈されます。モードパラメータが省略されている場合、ビットはクリアされているとみなされます。

ビット 1 (LSB): 累積補正 (マスク値 = 1)

このビットが ON の場合、以降の ALTER コマンドによって指定される座標値は増分として解釈され、累積されます。このビットが OFF の場合、座標値は適用される総 (非累積的) 補正として解釈されます。

ビット 2: ワールド座標 (マスク値 = 2)

このビットが ON の場合、以降の ALTER コマンドによって指定される座標値はワールド座標系として解釈されます。このビットが OFF の場合、座標はツール座標系として解釈されます。

詳細

このプログラムコマンドは、リアルタイムパス修正（変更）機能を開始します。このコマンドを実行した後、ALTER コマンドによって指定される座標値は、以降のすべてのロボット動作中に eV+ 軌道ジェネレーターによって算出された通常のパスに自動的に重ね合わせられます。 6 自由度（6DoF）すべてに補正を適用できます。補正は、ワールド座標やツール座標の累積または非累積値として指定することができます。

> **追加情報:** RX、RY、RZ 角は座標系基準の回転を表し、変換変数の自転 yaw、pitch、roll 角とは異なります。

変更モードが開始すると、連続パス区間で中断した場合、すべての後続の動作の間および動作間でロボットの位置は修正されます。以下に示すアクセションによって変更モードは終了します。

- ALTOFF プログラムコマンドの実行
- ロボットのデタッチ

- 通常よりも早いロボット動作の終了
- プログラム実行の停止

例

次の例では変更モードを開始し、ロボットの予定のパスに対する総補正（ビット1 をOFF）とワールド座標（ビット2 をON）補正として以降のALTER コマンドを解釈するようにします。

```
ALTON 2
```

関連キーワード

ALTER

ALTOFF

STATE

ANY

CASE 構造の代替命令グループの先頭を知らせます。

構文

ANY

使用上の留意事項

ANY ステートメントは CASE 構造内にある必要があります。

詳細

詳細についてはCASE 417 ページを参照してください。

例

詳細についてはCASE 417 ページを参照してください。

関連キーワード

CASE

VALUE

APPRO

関節補間動作で指定した位置を基準とした位置へ向かうロボット動作を開始します。

構文

APPRO location, distance

使用上の留意事項

APPRO プログラムコマンドは、タスクがロボットに割り付けてあれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、これらのコマンドを実行するとエラーになります。

パラメータ

location	基準となる最終ロケーションを定義する変換値です。
distance	指定したロケーションと実際に希望する目標位置の間の、ロボットツールZ軸に沿った距離を指定する実数値式です。 距離の値が正の場合、指定されたロケーションからツールを後方（ツールZ軸のマイナス方向）にオフセットします。距離の値が負の場合、ツールを前方（ツールZ軸のプラス方向）にオフセットします。

詳細

このコマンドにより、ロボットは指定されたロケーションの値によって示される方向に動作を開始します。目標位置は指定されたロケーションから、指定されたロケーションのZ 軸のマイナス方向に、指定された距離だけオフセットしています。

例

次の例では、ツールは変換値「place」によって定義された位置から「offset」mm 離れた位置に関節補間動作で移動します。

```
APPRO place,offset
```

関連キーワード

DEPART

DEPARTS

MOVE

MOVES

APPROS

直線動作で指定した位置を基準とした位置へ向かうロボット動作を開始します。

構文

APPROS location, distance

機能

使用上の留意事項

APPROS は直線動作を発生させ、その間、設定の変更はできません。

APPROS プログラムコマンドは、タスクがロボットに割り付けてあれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、これらのコマンドを実行するとエラーになります。

パラメータ

location	基準となる最終ロケーションを定義する変換値です。
distance	指定したロケーションと実際に希望する目標位置の間の、ロボットツールZ軸に沿った距離を指定する実数値式です。 距離の値が正の場合、指定されたロケーションからツールを後方(ツールZ軸のマイナス方向)にオフセットします。距離の値が負の場合、ツールを前方(ツールZ軸のプラス方向)にオフセットします。

詳細

このコマンドにより、ロボットは指定されたロケーションの値によって示される方向に動作を開始します。目標位置は指定されたロケーションから、指定されたロケーションのZ 軸のマイナス方向に、指定された距離だけオフセットしています。

例

次の例では、移動後の結果として得られるツールZ 軸に沿って50 mm 、変換値「place」によって定義された位置を超える位置まで、直線補間でツールを移動します。

```
APPROS place,-50
```

関連キーワード

DEPART

DEPARTS

MOVE

MOVES

ATTACH

デバイスをアプリケーションプログラムで使用できるようにします。

構文

ATTACH (lun, mode) $device

使用上の留意事項

DRY.RUN システムスイッチが有効な場合を除き、EXECUTE モニタコマンドまたはプログラムコマンドがタスク 0 に対して実行されている場合、ロボットは自動的に割り付けされます。実行プログラムが開始されると、その他すべての論理ユニットは自動的に割り付け解除されます。

動作が問題なく完了したことをモニタするために、IOSTAT 実数値関数を ATTACH プログラムコマンドと共に使用することを推奨します。詳細については、例を参照してください。

実行プログラムが停止したときに割り付けられていた場合、PROCEED、RETRY、SSTEP または XSTEP で実行プログラムを再開すると、モニタウィンドウまたはペンダントは自動的に再度割り付けられます。

パラメータ

lun

割り付けられているデバイスに関連付けられている論理ユニット番号です。このパラメータの解釈は、以下に説明するモードパラメータの値によって異なります。

モードパラメータのビット 3 が 0 の場合、このパラメータはオプションでありロボットを割り付けるためにデフォルトで 0 になっています。

これは、実数値、変数または式にすることができ、割り付けられる論理ユニットを指定する 0～24 の範囲の整数として解釈されます。指定されている論理ユニットが 0 ではない場合、$device パラメータを使用してその論理ユニットのデフォルトデバイスを無効にすることができます。

モードパラメータのビット 3 が 1 の場合、このパラメータは必須であり実数型変数である必要があります。この場合、eV+ システムは $device パラメータによって指定されたデバイスを割り付け、論理ユニット番号をこのパラメータに自動的に割り当てます。すべての論理ユニットが使用中の場合、このパラメータは -1 に設定されます。ATTACH リクエストが失敗しても eV+ は lun パラメータに値を割り当てます。

mode

ATTACH リスエストの処理方法を定義するオプションの実数値、変数または式です。整数として解釈されます。指定される値は以下に示すように連続するビットフラグと解釈されます。値が指定されていない場合全てのビットはクリアされているとみなされます。

ビット 1 (マスク値 = 1, LSB): キュー (0) 対失敗 (1)

このビットは、制御プログラムタスクからの割り付けリクエストへのデバイスドライバの対応方法を制御します。デバイスドライバは内部システムタスクであり、制御プログラムタスクとは区別されています。ほとんどのアプリケーションにおいては、このビットが ON である必要があります。

このビットが OFF の場合、デバイスは既に他の制御プログラムタスクに割り付けられており、ドライバはこの割り付けリクエストをキューすると共にこの割り付けリクエストにエラーがあることを制御プログラムに知らせます。この場合、このオペレーション IOSTAT(lun) はマイナスの数値を返します。デバイスが使用可能になると割り付けは完了します。

このビットが ON の場合、デバイスは既に他の制御プログラムタスクに割り付けられており、デバイスドライバは即座に割り付けリクエストが失敗したことを知らせます。

ビット2 (マスク値 = 2): 待機 (0) 対非待機 (1)

このビットは制御プログラムタスクがデバイスドライバからの応答を待つか待たないかを制御します。ほとんどのアプリケーションにおいては、このビットがOFFである必要があります。

このビットがOFFの場合、デバイスドライバが割り当てリクエストの結果を送るまで実行プログラムを待機します。

このビットがONの場合、割り当てリクエストの結果を待たずにプログラムを実行します。その場合プログラムは関数ステートメントIOSTAT(lun)を使用して割り当てが成功したのか反転する必要があります(上記参照)。割り当てを保留している間にプログラムが論理ユニットとの間で読み込みまたは書き込みをしようとすると、割り付けが完了するまで実行プログラムを待機します。

ビット3(マスク値 = 4): LUN を指定 (0) 対 LUN 割当済み (1)

このビットはlunパラメータの処理方法を決定しています。

このビットがOFFの場合、lunの値に対応するデバイスは割り付けられています。lunパラメータの値は割り付けられる予定のデバイスを指定ししています。ただし、$deviceパラメータで他のデバイスが指定されている場合はその限りではありません(以下の「詳細」項を参照してください)。

このビットがONの場合、割り付けられるデバイスは$deviceによって指定され、このパラメータを省略することはできません。この場合、ATTACHキーワードによって論理ユニットが自動的に選択され、lunパラメータの値はONに変わります。eV+はATTACHリクエストが失敗してもlunに値を割り当てます。このモードはロボットまたはペンダントを割り付けるために使用することができません。

$device　割り付けるデバイスを特定するオプションの文字列定数、変数または式です。このモードパラメータのビット3が0の場合、このパラメータはlunパラメータの値に関連付けられたデフォルトデバイスを無効にするために使用されます(論理ユニット0は常にロボットです)。

許容されるデバイス名は以下に示す通りです。

- **DISK** - コントローラの物理ドライブ(ディスクまたはSDカード)
- **MONITOR** - モニタウィンドウ
- **SERIAL:n** - 今後の使用のための予備
- **SYSTEM** - CDまたはDEFAULTキーワードを持つディスクデバイスセット
- **TCP** - TCPプロトコルデバイスドライバ
- **TFTP** - 今後の使用のための予備
- **UDP** - UDPプロトコルデバイスドライバ

詳細

ロボットの動作を実行するためには、ロボット制御プログラムによってロボットが割り付けられたままになっている必要があります。ロボットを割り付け解除すると、手動制御ペンダントを使用して、アプリケーションプログラムからの命令でロボットを動かすことができます。これは例えばアプリケーションのセットアップ手順において有用です。

プログラムタスク0は、実行開始時に 自動的にロボット1 を割り付けます。その他のプログラムタスクが実行するロボット制御プログラムではロボットを明確に割り付ける必要があります。

他のタスクでロボットがすでに割り付けられているのでなければ、任意のタスクを任意のロボットに割り付けることができます。ATTACH コマンドで割り付けられているロボットは、現在のタスクで実行した SELECT プログラムが最後に指定したロボットになります。SELECT プログラムコマンドを実行していない場合、ロボット 1 が割り付けられます。ロボットが現在タスクに割

り付けられていない場合に限って、SELECT プログラムコマンドを使用して異なるロボットを選択することができます。

ロボットを支障なく割り付けるためには、システムを COMP モードである必要があります。そうしないと（モードビット 1 = 0）、システムを COMP モードにするまで実行プログラムを告知なく停止します。この状態は次の手段で回避できます。

- ATTACH コマンドを実行する前に、STATE 関数を使用してシステムが COMP モードであるか判定します。
- mode値で ビット 1 を設定し、IOSTAT 関数を利用して ATTACH コマンドの成功を判定します。

モニタウィンドウ（論理ユニット 4）を割り付けている場合、すべてのキーボード入力を、プログラムが要求する入力のためにバッファします。

> **注:** モニタウィンドウを割り付けている場合、ユーザーは実行プログラムを終了するために ABORT とタイプすることができません。このプログラムが終了リクエストに対応するための手段を提供するか、またはユーザーがペンダントまたは非常停止スイッチを使用して実行プログラムを停止できるようにする必要があります。

DISK デバイスが割り付けられている場合、プログラムがファイルとの間でデータの読み書きをすることができます（DISK は SD カードを意味します）。アクセスするファイルを指定するためには FOPEN プログラムコマンドの 1 つを使用する必要があります。これにより WRITE および READ プログラムコマンドを使用してファイルとの間で情報を伝達することができます。FCMND プロファイルコマンドを使用してコマンドをファイルシステムに送ることができます。

モードビット 3 = 0 であり、$device パラメータが省略されている場合、論理ユニット番号によって、該当するデフォルトデバイスが以下の表から指定されます。

3.1LUN が与えるデフォルトデバイス番号

番号	デバイス
0	ロボット（lun を省略している場合のデフォルト）
1	今後の使用のための予備
2、3、4	モニタウィンドウ
5、6、7、8	ディスク
9	デフォルトデバイスなし
10、11、12、13	今後の使用のための予備
14、15、16	デフォルトデバイスなし
17、18、19	ディスク
20、21、22、23、24	今後の使用のための予備
25、26、27、28、29、30、31	デフォルトデバイスなし

例

以下の例ではタイマ3 で指定した時間（5 秒）、ロボットの割り付けを試みます。IOSTAT 実数値関数は、この時間内に ATTACH オペレーションが成功したか失敗したかをモニタするための情報を返します。この割り付けオペレーションが 5 秒以内に成功しない場合、モニタウィンドウにはメッセージが表示されます。

```
start = TIMER(-3)
DO
    ATTACH (0,1)
    UNTIL (IOSTAT(0, 0) == 1) OR ((TIMER(-3) - start) > 5)
IF IOSTAT(0 , 0) < 1 THEN
    TYPE "The following error occurred: ", $ERROR(IOSTAT(0,0))
    TYPE "The program will be interrupted!"
HALT
END
```

関連キーワード

DETACH

FSET

IOSTAT

SELECT（プログラムコマンド）

AUTO

プログラムを入力するときにプログラムスタックに自動的に作成されるテンポラリ変数を宣言します。

構文

AUTO type **variable,** ..., variable

使用上の留意事項

最初にプログラムを入力するときに、AUTO（自動）変数には未決定の値がありますが、それらは未定義であるとはみなされません。プログラムが終了すると値はなくなります。

AUTO コマンドはプログラム内の実行可能なキーワードの前に必ず現れなければなりません。.PROGRAM コマンド、コメント、空白行、GLOBAL および LOCAL コマンド、およびその他の AUTO コマンドだけがこの命令よりも前に置くことができます。

変数が AUTO コマンドに記載されている場合、その変数と同じ名称のグローバル変数にプログラムから直接アクセスすることはできません。

AUTO 変数の値は STORE コマンドで保存されませんし LOAD コマンドで復旧もされません。

パラメータ

type

type パラメータを指定すると、すべての変数がそのタイプ（REAL、DOUBLE または LOC）に一致する必要があります。配列変数には、インデックスを明確に指定してその配列の最高次の有効なインデックスを示す必要があります。

このキーワードを省略すると、それぞれの変数のタイプはプログラム内での使用によって決まります。使用によってタイプを決定できない場合、エラーが発生します。

LOC	ロケーション変数(変換またはプレシジョンポイント)です。
REAL	単精度実数型変数です。
DOUBLE	倍精度実数型変数です。 デフォルトタイプに関する詳細については GLOBAL コマンドを参照してください。

variable

eV+ で使用可能な任意のデータタイプの変数名です（ベルト、プレシジョンポイント、実数値、文字列、変換）。それぞれの変数は単純な変数または配列である場合があります。type パラメータを指定すると、すべての変数がそのタイプに一致する必要があります。配列変数には、インデックスを明確に指定してその配列の最高次の有効なインデックスを示す必要があります。

詳細

このコマンドは現在のプログラム内だけで定義する変数を宣言するために使用されます。AUTO 変数はプログラムの特定の呼び出しインスタンスによってのみ参照できます。他の任意のプログラムで定義されている変数の名前であっても、AUTO 変数の名前を選択することができます。

AUTO 変数はプログラムが呼び出されるたびに割り当てられ、連続するサブルーチンの呼び出しの間でも保存されません。これらの値は、そのプログラムタスクが非アクティブで実行スタックにある場合に限ってモニタコマンドを使って表示できます。プログラムを最初に入力すると、AUTO 変数には任意の未決定値がありますが、それらは未定義とはみなされません。AUTO 変数はプログラムが終了すると消失します。

LOCAL 変数とは異なり、プログラムを呼び出すと、その都度 AUTO 変数の個別のコピーが作成されます。これはいくつかの異なるプログラムタスクによって同時に呼び出された場合や、単一のタスクに繰り返し呼び出される場合も同様です。LOCAL 変数またはグローバルな変数を使用するプログラムが、いくつかの異なるプログラムタスクによって呼び出される場合や、単一タスクに繰り返し呼び出される場合、これらの変数の値は異なるプログラムインスタンスによって変更されたり、予期できないプログラムエラーを生じる可能性があります。そのため、こういったエラーの可能性を最小限に抑えるために、すべての一時的変数に対して AUTO 変数を使用するようにしてください。

変数は GLOBAL、AUTO、または LOCAL として定義することができます。AUTO、GLOBAL、または LOCAL 変数に同じ名称を定義しようとすると、エラーメッセージ「Attempt to redefine variable class」（変数の分類を再定義しようとしました）が表示されます。

変数は同じコンテクストの内で 1 回に限り定義することができます（AUTO、LOCAL または GLOBAL）。異なるタイプで複数回、変数を定義しようとすると、エラーメッセージ「Attempt to redefine variable type」（変数の分類を再定義しようとしました）が表示されます。

AUTO 配列変数は AUTO ステートメントに指定されるそれぞれの次元のサイズを持つ必要があります。指定されるそれぞれのインデックスはその次元で参照される最後の要素を示す必要があります。割り当てられる最初の要素は常にインデックス値が 0 になります。例えば、以下のステートメントでは変換配列に 24 の要素が割り当てられています。左側のインデックスは 0～3 の範囲であり、右側のインデックスは 0～5 の範囲です。

```
AUTO LOC points[3,5]
```

AUTO 変数の保存スペースはプログラム実行スタックに割り当てられています。宣言した AUTO 変数の数に対してスタックが小さすぎる場合、タスクの実行はエラーメッセージ「Too many subroutine call」（サブルーチンの呼び出しが多すぎます）と共に止まります。この場合、STATUS モニタコマンドを使用していくつの追加スタックスペースが必要になるか判断します。次に、STACK モニタコマンドを使用してスタックサイズを増やし、RETRY モニタコマンドを発行して実行プログラムを継続します。

AUTO 変数は DELETE_ コマンドを使用して削除することができません。

次の例に示すように、AUTO 変数はオプションのコンテキスト指定子 @ を使用することで、DELETE_、DO、HERE、LIST_、TEACH、TOOL などのモニタコマンドで参照することができます。

```
command @task:program command_arguments
```

例

以下の例のステートメントでは、変数 loc.a、$ans および i を現在のプログラムにおいて AUTO にすることを宣言します。loc.a および i の変数タイプをプログラム内での使用からクリアである必要があります。

```
AUTO loc.a, $ans, i
```

以下の例では、変数 i、j および tmp[ ] を現在のプログラムにおいて AUTO の実数型変数にすることを宣言します。配列要素tmp[0] からtmp[10] が定義されます。

```
AUTO REAL i, j, tmp[10]
```

以下の例では、変数「loc」を現在のプログラムにおいて AUTO 変数にすることを宣言します。「loc」の変数タイプはプログラム内での使われ方によって決める必要があります。「loc」はそれ単独で現れているので、タイプ指定型キーワードとは解釈されません。

```
AUTO loc
```

関連キーワード

GLOBAL

LOCAL

STACK

DO

HERE

TEACH

TOOL

BASE

ロボットを基準にしてワールド基準フレームを平行移動、回転させます。

構文

BASE X_shift, Y_shift, Z_shift, Z_rotation

使用上の留意事項

BASE プログラムコマンドは連続パスモーションにおいて中断を発生させます。

このコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、BASE コマンドを実行するとエラーになります。

パラメータ

X_shift	新しい座標系に対して、その原点の通常のワールド座標系における X 座標を記述するオプションの実数値式です。値が設定されていない場合、0 とみなされます。
Y_shift	新しい座標系に対して、その原点の通常のワールド座標系における Y 座標を記述するオプションの実数値式です。値が設定されていない場合、0 とみなされます。
Z_shift	新しい座標系に対して、その原点の通常のワールド座標系における Z 座標を記述するオプションの実数値式です。値が設定されていない場合、0 とみなされます。
Z_rotation	新しい座標系に対して、その原点の通常のワールド座標系における Z 軸回りの回転を記述するオプションの実数値式です。値が設定されていない場合、0 とみなされます。

詳細

eV+システムを初期化すると、ロボットの基準フレームの原点は空間内で固定されていると仮定されます。このとき、ロボットの設置面が X-Y 面にあり、X 軸はジョイント 1 によって定義される方向になっていて 0 に等しく、Z 軸はジョイント 1 の軸に一致するようになります。

BASE コマンドはこの基準フレームを前述のようにオフセットして回転します。これは、アプリケーションに対してロケーションを定義した後にロボットを動作させる場合に有用です。

ロボットがこれらのロケーションを基準としてX_shift、Y_shift、Z_shift 分平行移動したポイントへ移動し、Z_shift 軸回りに Z° 回転する場合、BASE コマンドを使用することで、ロボットのロケーションをロボット基準フレームを基準とした変換によって定義した後に、以前に定義したロケーションへのモーションを、要求通りのままにするように補正することができます。

BASE コマンドのもう一つの便利な使い方としては、X および Y 座標軸を再調整することで、SHIFT 関数が希望する非標準の方向へ変位するようにするというものがあります。

> **注:** BASE プログラムコマンドはプレシジョンポイントとして定義されているロケーションに影響を与えません。BASE プログラムコマンドの引数はロボットの通常のロケーションを基準とした変位を記述します。

追加情報: BASE 関数を LISTL モニタコマンドと共に使用することで、モニタウィンドウに現在の BASE 設定を表示することができます。

例

次の例では、ロボットが X 軸の正方向に「xbase」mm、Z 軸の負方向に 50.5 mm 移動し、軸回りに 30° 回転したために、ワールド基準フレームを再定義しています。

```
BASE xbase,, -50.5, 30
```

次の例では、ワールド基準フレームを再定義することで、すべての位置を、その通常の位置から X 軸の負方向に 100 mm、Z 軸の正方向に 50 mm 移動します。このステートメントの引数はロボットを基準としたロボット基準フレームの動きを記述しており、ロボットを基準とするロケーションに逆の効果を及ぼします。

```
BASE 100,, -50
```

関連キーワード

BASE（変換関数）

SELECT（モニタコマンド）

SELECT（プログラムコマンド）

SELECT（実数値関数）

BELOW

次の動作に関するロボットの設定変更を要求して、肘が肩から手首のラインより下になるようにします。

構文

BELOW

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがBELOW設定をサポートしていない場合、ロボットはこの命令を無視します。

> **追加情報:** ：スカラおよびパラレルロボットにはBELOW設定することはできません。

BELOW プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。ロボットが割り付けられていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、BELOWプログラムコマンドを実行するとエラーになります。以下の図にABOVE およびBELOW 設定を示します。

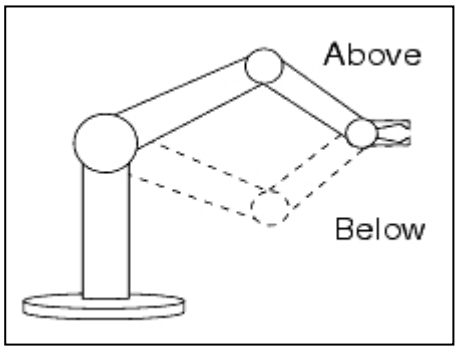


図3-6. ABOVE およびBELOW 設定

例

次の例では、BELOW設定で「point1」にロボットを移動します。

```
BELOW
MOVE point1
BREAK
```

関連キーワード

ABOVE

CONFIG

SELECT（プログラムコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

BITS

値に基づくデジタル信号のグループを設定またはクリアします。

構文

BITS first_sig, num_sigs = **value**

使用上の留意事項

外部デジタル信号と内部ソフトウェア信号の両方を参照することができます。入力信号は参照しないでください。入力信号はモニタコマンド IO 1 によって表示されます。

一度に 32 以上の信号を設定することはできません。

グループ内のすべての信号がシステムでの使用のために設定されているのであれば、最大 32 の信号の任意のグループを設定することができます。

すべての eV+ デジタル出力コマンドが eV+ サイクルを待機するわけではなく、すぐに出力を ON にします。デジタル入力は eV+ オペレーティングシステムによって 2 ms 毎に確認されます。

パラメータ

first_sig	影響を受ける最も少ない番号の信号を定義する実数値式です。
num_sigs	影響を受ける信号の数を定義するオプションの実数値式です。指定しない場合、値 1 が仮定されます。有効な最大値は 32 です。
value	指定した信号に設定される値を定義する実数値式です。この値のバイナリ表記の数が num_sigs よりも多い場合、最も小さい num_sigs 信号だけが影響を受けます。

詳細

このプログラムコマンドは 1 つ以上の出力信号または内部ソフトウェア信号を、等号の右側の値に基づいて ON または OFF にします。このコマンドの効果は、値パラメータを整数に丸めて、その整数のバイナリ表記の個々のビットに基づいて、いくつかの信号を ON または OFF にすることです。

例

次の例では、外部出力信号1～8(8ビット)を現在のモニタ速度設定値のバイナリ表記に設定します。

```
BITS 1,8 = SPEED(1)
```

モニタ速度が 50%（バイナリで 0011 0010）に設定されている場合、上記のステートメントが発行されると、信号 1～8 は次のように設定されます。

信号 1: 0 (OFF)　　信号 5: 1 (ON)

信号 2: 1 (ON)　　信号 6: 1 (ON)

信号 3: 0 (OFF)　　信号 7: 0 (OFF)

信号 4: 0 (OFF)　　信号 8: 0 (OFF)

次の例では、外部出力信号 5～9（4 ビット）を BCD 桁 7 のバイナリ表記に設定します。

```
BITS 5,4 = BCD(7)
```

次の例では、外部出力信号 1～8（8 ビット）を定数 255 のバイナリ表記（11111111）に設定します。信号 1～8 によってすべて ON になります。

```
BITS 1,8 = 255
```

関連キーワード

BITS（実数値関数）

IO

RESET

SIG

SIG.INS

SIGNAL（モニタコマンド）

SIGNAL（プログラムコマンド）

BRAKE

現在のロボットモーションを中止(アボート）します。

構文

BRAKE

使用上の留意事項

BRAKE プログラムコマンドは、ロボットを能動的に制御していないタスクを含む、任意のプログラムタスクによって実行することができます。

このコマンドによって BREAK は発生しません。

複数のロボットがコントローラに接続している場合、このコマンドは現在選択しているロボットに適用されます（SELECT キーワードを参照してください）。

eV+ システムがロボットを制御するように設定されていない場合、ロボットが存在しないことで BRAKE コマンドはエラーを生じません。

詳細

BRAKE プログラムコマンドにより、現在のロボットの動作は直ちにアボートされます。このコマンドに応答して、ロボットは減速して停止し、位置エラーをヌルにするのを待つことなく次のモーションを開始します。

注: ロボットが停止するまで、実行プログラムは中断しません。

例

次の例では、ロボットがモーションを開始すると同時に条件が適合しているかテストします。条件が適合している場合、BRAKE コマンドによってモーションは停止します。そうではない場合、モーションは通常通りに完了します。

```
MOVES step[1]
DO
    IF SIG(1023) THEN
        BRAKE
        EXIT
    END
UNTIL STATE(2) == 2
MOVES step[2]
```

関連キーワード

BREAK

BREAK

現在の動作が完了するまで実行プログラムを中断します。

構文

BREAK

使用上の留意事項

BREAK プログラムコマンドは、現在のタスクに割り付けられているロボットによる動作を待つ場合に限って使用されます。

eV+ システムがロボットを制御するように構成されていない場合、BREAK コマンドを使用するとエラーになります。

詳細

このコマンドには 2 種類の作用があります。

1. ロボットが現在の目的地に到達するまでプログラムの実行が中断します。

   注: ある動作が別のタスクによって完了するまで、あるタスクを待機させるために BREAK を使用することはできません。

2. 現在の動作と次の動作ステートメントが命じる動作の間の連続経路移動はとぎれます。この 2 つの動作を単一の連続する経路に統合することはできません。

BREAK コマンドは、その動作が終了するまで eV+ プログラムをブロックすることで、連続経路処理を終了させます。CPOFF プログラムコマンドでは、軌道ジェネレータが eV+ プログラムの先読み処理に影響を与えることなく、継続経路を終了させます。

例

ブレンディングまたは先読み処理をすることなく、「point1」への移動が完了した後、ロボットを「point2」に移動させます。

```
MOVE point1
BREAK
MOVE point2
```

関連キーワード

BRAKE

CPOFF

CPON

CP

SELECT（プログラムコマンド）

SELECT（モニタコマンド）

SELECT（実数値関数）

BY

SCALE および SHIFT 関数の構文を完了させます。

構文

SCALE(transformation BY *value*)

SHIFT(transformation BY *value*, *value*, *value*)

例

```
SET new.trans = SCALE(old.trans BY scale.factor)

SET new.trans = SHIFT(old.trans BY x,y,z)
```

関連キーワード

SCALE

SHIFT

CALIBRATE

ロボットの位置決めシステムをロボットの現在位置を使用して初期化します。

構文

CALIBRATE mode, status, robot

使用上の留意事項

通常、キーワードが発行されるとモードは 0 になります。

DRY.RUNスイッチが有効な場合このコマンドはロボットに効果はありません。

ロボットを動かす場合、CALIBRATE プログラムコマンドまたはモニタコマンドのキーワードは、システム電源をオンにする、またはシステムをディスクから起動するたびに実行される必要があります。起動を有効にしたときにCALIBRATEを実行するオプションがロボットにある場合、ロボットは起動時にCALIBRATIONオペレーションを実行します。このオプションが有効ではない（またはそのロボットでサポートされていない）場合、ハイパワーが有効になった後で、このキーワードを実行しなければなりません

起動時にCALIBRATE を実行するオプションの有効化に関する詳細については『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。

ロボットがキャリブレーションされていない場合、CALIBRATEプログラムコマンドまたはモニタコマンドが実行されるまでロボットを手動制御ペンダントを使用して、またはプログラム制御の下で動かすことはできません。

CALIBRATEキーワードは、そのそれぞれのロボットタイプで異なるオペレーションをする可能性があります。アブソリュート（すなわちインクリメンタル）エンコーダを装備するロボットの場合、このキーワードによりロボットは動作します。この場合、ロボットを作業範囲区域から遠く離して、キャリブレーションプロセス中に範囲外に移動しないようにする必要があります。ロボット動作の詳細についてはCALIBRATEモニタコマンドキーワードの説明を参照してください。

eV+ システムがロボットを制御するように設定されていない場合、CALIBRATEを実行するとエラーになります。

パラメータ

mode　キャリブレーションのどの部分を実行するかを示すオプションの実数値式です。

モード値	解釈
0（または省略）	通常のキャリブレーションを実行します。 詳しく言うと、次のオペレーションを行います。 (a) メモリにない場合、メインキャリブレーションプログラムを読み込みます。 (b) 読み込み、実行および削除フラグセットを使用してメインキャリブレーションプログラムを実行します。これによりそのロボット専用のキャリブレーションルーチンを読み込み、ロボットをキャリブレーションし、ロボットルーチンを削除します。メインキャリブレーションプログラムはメモリに残ります。
1	メモリにない場合、メインキャリブレーションプログラムを読み込み、読み込みフラグセットを使用してメインキャリブレーションプログラムを実行します。これによりキャリブレーションプログラムは適切なロボット専用キャリブレーションルーチンを読み込みます。キャリブレーションプロセスは実際には実行されないことに注意してください。
2	メインキャリブレーションプログラム（すでにメモリにある）を実行フラグセットを使用して実行します。これにより、システムのロボットをキャリブレーションし、すべてのキャリブレーションプログラムがメモリに残されます。
3	メインキャリブレーションプログラム（すでにメモリにある）を削除フラグセットを使用して実行します。これによりキャリブレーションプログラムはロボット専用キャリブレーションルーチンをメモリから削除します。実際のキャリブレーションプロセスは実行されず、メインキャリブレーションプログラムはメモリに残されることに注意してください。

status　キャリブレーションプログラムから返された、または(mode -1 においては)独自の「キャリブレーション」モードに入ろうとしたときに eV+ から返された、終了ステータスを受けるオプションの実数型変数です。

robot	影響を受けるロボットの番号を示すオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略されているか 0 の場合、すべてのロボットが影響を受けます。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

キャリブレーション作業が開始されると、eV+は、ロボットはキャリブレーションされていないと仮定するので、ペンダントおよびアプリケーションプログラムを使用してロボットを動かす機能が制限されます。

> **注:** ロボットがキャリブレーションされていない場合、ペンダントの COMP モードライトが点灯しません。

システム電源を OFF にすると、インクリメンタルエンコーダを装備したロボットは起動時キャリブレーションを失います。*Encorder quadrature error*（エンコーダ直交エラー）がロボットの関節の一つに発生した場合、安全のためにこういったロボットも起動時キャリブレーションを失います。ロボットがキャリブレーションを失う可能性のあるその他のサーボエラーとしては、*Unexpected zero index*（予期しない 0 インデックス）、*No zero index*（0 インデックスなし）および *RSC Communications Failure*（RSC 通信不良）があります。

> **注:** eCobra 600 および 800 ロボットの場合、このオペレーション命令はジョイント 4（θ）の小さな動作を生じます。

このプログラムコマンドが、メインキャリブレーションプログラムを読み込もうとする場合、同じプログラム、モジュール、ファイル名および検索プログラムがCALIBRATEモニタコマンドに関して採用されます。

タスク0でCALIBRATEオペレーションを実行する場合、プログラム内のEXECUTEプログラムコマンドキーワード命令で/C修飾子を使用してプログラムを完了することができます。指定された修飾子を使用して、DRY.RUNシステムスイッチが無効な場合であってもロボットをキャリブレーションするプログラムをタスク0で実行できます。タスク0以外のタスクで実行されているプログラムは特別な条件なしでCALIBRATEオペレーションを実行することができます。

CALIBRATEオペレーションは一度に1つのタスクからしか実行することができません。複数のタスクから呼び出された場合（例えばREACTEプログラムから）、その呼び出しをTASキーワード関数を使用したミューテックスで保護する必要があります。

例

以下の例では、ロボットに起動時キャリブレーションを実行するために、任意のプログラムタスクが使用することができます（タスク0を使用する場合、プログラムを実行する前にDRY.RUNシステムスイッチを有効にする必要があります。

```
DETACH()
DISABLE DRY.RUN
ENABLE POWER

CALIBRATE
ATTACH()
```

複数のタスクからCALIBRATEが呼び出された場合、以下の例の命令シーケンスを使用する必要があります。

```
GLOBAL en_po_lock


WHILE TAS(en_po_lock,TRUE) DO
    WAIT
END


ENABLE POWER

CALIBRATE
ATTACH()


en_po_lock = FALSE
```

関連キーワード

CALIBRATE

NOT.CALIBRATED

SELECT（プログラムコマンド）

SELECT（実数値関数）

TAS

CALL

現在実行しているプログラムを中断し、新しいサブルーチンプログラムを使用して実行を継続します。

構文

CALL program (arg_list)

使用上の留意事項

外部変数タイプは、arg_list パラメータに使用できません。

パラメータ

program 実行する予定の新しいプログラムの名称です。

arg_list 現在実行中のプログラムと新しいプログラムの間で引き渡されるサブルーチン引数をコンマで区切ったオプションのリストです。引数リストを指定しない場合、program パラメータの後の括弧は省略可能です。

引数を使用すると、次に説明するように、データを呼び出されたプログラムに引き渡すこと、結果を受け取ること、またはこの両者を組み合わせることが可能です。

各引数は eV+ がサポートしている 1 つのデータタイプ（ベルト、精密位置、実数値、文字列、および変換）にすることが可能で、定数、変数、または式として指定することができます（以下の注記を参照してください）。それぞれの引数のタイプは呼び出されたプログラムの引数リストにある相手方のタイプに一致している必要があります。変数として指定されている引数は、単純な変数、配列要素またはその 1 つ以上のインデックスが空白になっている配列とする必要があります（詳細については以下を参照してください）。

> **注:** 値が呼び出しているプログラムに渡し戻されている場合、そのパラメータは変数として指定する必要があります。
>
> 外部変数タイプは使用できません。

任意の引数は省略できますが、呼び出されたプログラムのそれに対応する引数は未定義になります。引数リスト内で引数を省略する場合、区切り用のカンマを含めたままにする必要があります。リストの最後にある引数を省略する場合、その引数の前にあるカンマも省略することができます。

> **注:** 引数を省略することの影響に関する情報については .PROGRAM プログラムコマンドを参照してください。

詳細

CALL コマンドによって現在実行中のプログラムは一時的に停止します。示された新しいプログラムの最初のステップで実行が継続され、これはサブルーチンとみなされます。

サブルーチンで RETURN コマンドを実行すると、実行は自動的に現在のプログラムに戻ります。CALL コマンドの直後のキーワードを使用して実行が継続します。

サブルーチンの引数は値または参照によって引き渡すことができます。引数が値によって引き渡された場合、引数値のコピーがサブルーチンに引き渡されます。サブルーチン内での対応するサブルーチン引数の変更は、呼び出し側プログラムに引き戻されません。式または複合変換として指定されている引数は値によって引き渡されます。

引数がスカラまたは配列変数である場合、参照によって引き渡されます。すなわち、その変数のポインタがサブルーチンに引き渡され、その後呼び出し側プログラムと同じ変数を持って働きます。呼び出されたサブルーチンがこの変数の値を変える場合、この値も呼び出し側プログラムに対して変更されます。これは特に、同じ変数が 1 つのサブルーチン呼び出しの 2 つの引数として引き渡される場合などに重要になります。サブルーチン内の対応するサブルーチン引数のいずれかを変更すると、もう一方の対応するサブルーチン引数を自動的に変更します。

参照によって引き渡される引数は（変数であるために）一般的に値によって引き渡されることを強制したい場合があります。その方法は、次の例に示すように変数のタイプによって異なります。

- 実数値変数に対しては、次に示すように、括弧内の変数を閉じることで値による引き渡しを強制することができます。

```
CALL prog_a((count))
```

- 文字列変数の場合、次に示すようにして空の文字列（""）を変数に追加することができます。

```
CALL prog_b($str.var+"")
```

- 変換変数（start など）の場合、次に示すように、その変数と NULL 変換で構成される複合変換によって、同等の変換値を指定することができます。

```
CALL prog_c(start:NULL)
```

上述のとおり、arg_list 内の項目は、呼び出されたサブルーチンの該当する項目と一致する必要があります。スカラ同士および同次元配列の直接的一致に加えて、低次配列の代わりにより高次の配列を引き渡すいくつかの状況があります。例えば、次の事例はすべて有効です。

- 配列要素をスカラへ引き渡す:

```
CALL example(a[1])      --->    .PROGRAM example(b)
CALL example(a[1,2])    --->    .PROGRAM example(b)
CALL example(a[1,2,3]) --->    .PROGRAM example(b)
```

- 配列の 1 次元を 1 次元配列に引き渡す:

```
CALL example(a[])       --->    .PROGRAM example(b[])
```

```
CALL example(a[1,])     --->    .PROGRAM example(b[])
CALL example(a[1,2,])   --->    .PROGRAM example(b[])
```

- 配列の 2 次元を 2 次元配列に引き渡す:

```
CALL example(a[,])      --->    .PROGRAM example(b[,])
CALL example(a[1,,])    --->    .PROGRAM example(b[,])
```

- 配列の 3 次元を 3 次元配列に引き渡す:

```
CALL example(a[,,])     --->    .PROGRAM example(b[,,])
```

例

次の例では、プログラム「pallet」を実行して、変数「count」へのポインタを引き渡します。RETURN コマンドをサブルーチン「pallet」で実行する場合、制御は CALL コマンドを含むプログラムに返され、変数「count」には対応するサブルーチン引数の現在値が含まれます。

```
CALL pallet(count)
```

次の例ではプログラム「cycle」を実行します。値 1 がプログラム「cycle」の最初のパラメータに引き渡され、その 2 番目のパラメータは未定義であり、3 番目のパラメータは式 n+3 の値を受けます。「cycle」プログラムに 3 つ以上のパラメータがある場合、残りのパラメータはすべて未定義です。

CALL コマンドの引数には変数がないため、プログラム「cycle」によってデータは返されません。

```
CALL cycle(1, , n+3)
```

関連キーワード

CALLS

RETURN

CALLS

現在実行しているプログラムを中断し、文字列値で指定される新しいサブルーチンプログラムを使用して実行を継続します。

構文

CALLS string(arg_list)

使用上の留意事項

CALLS は通常の CALL コマンドよりも実行に時間が相当かかります。CALLS は必要な場合に限って使用すべきです。

パラメータ

string 実行する新しいプログラムの名称を定義する、文字列値、変数または式です。名称の文字は小文字または大文字にすることができ、その長さは 1～15 文字である必要があります。

arg_list 現在実行中のプログラムと新しいプログラムの間で引き渡されるサブルーチン引数をコンマで区切ったオプションのリストです。引数リストを指定しない場合、program パラメータの後の括弧は省略可能です。サブルーチン引数に関する詳細については CALL コマンドを参照してください。

注: 引数リストは文字列パラメータの一部として定義されていないので、指定した CALLS コマンドで呼び出したすべてのサブルーチンには等価な引数のリストが存在する必要があります。

詳細

CALLS コマンド関数は CALL コマンドとほぼ同一です。この 2 つの唯一の違いは、サブルーチン名の指定方法です。CALL では名前をコマンドステップで明確に入力する必要があります。CALLS では名前を文字列型の変数または式で指定することが可能であり、プログラムを実行する際にその値を定義することができます。そのため、プログラムが環境に応じて異なるサブルーチンを呼び出すことができます。

CALL コマンドの場合、サブルーチンで RETURN コマンドを実行すると、実行は自動的に現在のプログラムに戻ります。CALLS コマンドの直後のキーワードを使用して実行が継続します。

例

以下の例では、CALLS コマンドを使用することで、プログラム実行時にその名前が決まるサブルーチンの実行方法を実演しています。

プログラムは 4 桁の入力セット（1001～1004）を読み込んで、16 の異なる部品タイプのどれを適用して「type」変数に返すかを判断します。部品タイプは 16 進数とみなされ、「$type」変数として対応する ASCII 文字に変換されます。この文字が定義されると、適切なサブルーチン（「part.0」、「part.1」、...、「part.f」）

が呼び出されます。この部品タイプ値はサブルーチンに引き渡される 2 次元配列引数の部分を選択するためにも使用されます。

```
type = BITS(1001,4)
$type = $ENCODE(/H0, type)
CALLS "part."+$type(arguments[type,], status)
```

STATUS 実数型関数を使用して、提示されるサブルーチンを呼び出し前に確実に存在させることで、この例をよりロバストにすることができます。この関数を使用すると、未定義のプログラム名によるエラーの可能性を回避できます。

関連キーワード

CALL

CALLS

.PROGRAM

RETURN

STATUS

CASE

該当する値を定義することで CASE 構造の処理を開始します。

構文

CASE value OF

使用上の留意事項

このコマンドは完全な CASE 構造の一部である必要があります。

パラメータ

value	どのキーワードを実行するかを判断するために CASE 構造において一致すべき値を定義する実数値、変数または式です。

詳細

CASE コマンドは eV+ で使用可能な柔軟な構造です。任意の数のグループから 1 つのキーワードグループを実行する手段を提供します。完全な構文は以下のとおりです（空白行は不要です）。

```
CASE evaluation OF
  VALUE evaluation_condition_1,...:
    group_of_steps
  VALUE evaluation_condition_2,...:
    group_of_steps
      .
      .
      .
  ANY
    group_of_steps
END
```

縦に並んだ 3 つのドットは任意の数の VALUE ステップを使用して追加のキーワードグループを実行できることを示します。

上記の ANY ステップはオプションです。CASE 構造には ANY ステップは 1 つしか存在できず、上記のように構造内の最後のグループをマークする必要があります。

上の例で示すように、ANY およびEND ステップは、それらが単独で行にある必要があります。ただし他のすべての命令と同様に、それらの行にコメントを付けることはできます。

CASE 構造は以下のようにして処理されます。

1. CASE キーワードに続く式を評価します。
2. 判定条件に適合するものが見つかるまで、すべての VALUE ステップをスキャンします。
3. VALUE ステップに続くキーワードを実行します。
4. END ステップ後の最初のキーワードまで実行を継続します。

判定条件に適合する VALUE ステップが見つからず、構造内に ANY ステップがある場合、ANY ステップに続くキーワードを実行します。

構造内で VALUE 判定条件が満たされず、ANY ステップがない場合、CASE 構造のすべてにおいて、実行される命令はありません。

例

以下の例に CASE コマンドの基本関数を示します。

```
CASE number OF
  VALUE 1:
    TYPE "one"
  VALUE 2:
    TYPE "two"
  ANY
    TYPE "Not one or two"
END
```

以下のサンプルプログラムではテスト値の入力を促します。値がマイナスである場合、プログラムはメッセージを表示して終了します。それ以外の場合、CASE コマンドを使用して入力値を以下の 3 グループのどれかの構成要素として分類します。

1. 0～10 までの中の偶数
2. 0～10 までの中の奇数
3. それ以外のすべての数字

```
PROMPT "Enter a value from 0 to 10: ", x
CASE x OF
  VALUE 0, 2, 4, 6, 8, 10:
    TYPE "The number", x, " is EVEN"
  VALUE 1, 3, 5, 7, 9:
    TYPE "The number", x, " is ODD"
  ANY
    TYPE x, " is not an integer from 0 to 10"
END
```

次の例では、CASE コマンドを使用した文字列データの評価方法を示します。

```
PROMPT "Enter a number", x
CASE TRUE OF
  VALUE (x > 0):
    TYPE "The number was greater than 0."
  VALUE (x == 0):
    TYPE "The number was equal to 0."
  VALUE (x < 0):
    TYPE "The number was less than 0."
.END
```

次の例では、CASE コマンドを使用した文字列データの評価方法を示します。

```
CASE TRUE OF
  VALUE ($string1=="hello"):
    TYPE "string=hello"
  VALUE($string1=="world"):
    TYPE "string=world"
  ANY
    TYPE "unknown string"
END
```

関連キーワード

ANY

END

TRUE

VALUE

CLEAR.EVENT

指定されたタスクに関連するイベントをクリアします。

構文

CLEAR.EVENT task, flag

パラメータ

task	クリアされるイベントのタスクを指定する、整数と解釈されるオプションの実数値、変数または式です。有効な範囲は 0 以上 6 以下、または 0 以上 27 以下です。このパラメータを省略した場合、現在のタスクの番号が使用されます。
flag	使用しません。デフォルトは 1 です。

詳細

このコマンドは指定されたタスクに関連するイベントをクリアします。

デフォルトのクリアされるイベントは、ステートメント WAIT.EVENT 1 が待機している入出力完了イベントです。このイベントは、任意の入出力キーワード作動の実行（完了ではない）によってクリアされます。

関連キーワード

GET.EVENT

SET.EVENT

WAIT.EVENT

CLEAR.LATCHES

選択したデバイスのラッチバッファを空にします。

構文

CLEAR.LATCHES (select)

パラメータ

select	この関数が実行されてからラッチが発生したかどうかを判定する整数値、式または実数変数です。設定値は以下に示す通りです。
0	現在選択しているロボットのラッチバッファをクリアする
-n (< 0)	ベルトn のラッチバッファをクリアする

詳細

このプログラムコマンドは指定したラッチバッファに関連するイベントおよび情報をクリアします。

LATCHED 実数値関数とは対照的に、ラッチイベントデータを取得できなくします。

例

次の例は一般的に、V+ プログラムが実行されていなかったときに検出されたと思われるラッチがバッファ（ファーストインファーストアウト）からクリアされ、新しいラッチがすべて正確であることを確実にするために初期化プログラムで使用されます。次の例において、V+ V+ プログラムを停止しても、ベルト 1 は引き続きラッチを生成する場合、これらのラッチはまだロボットの上流にあるオブジェクトに関連しないものである可能性があります。これらのラッチが依然有効である可能性のあるアプリケーションの場合を除き、プログラムが起動するときにラッチバッファをクリアするのは良い習慣です。

```
CLEAR.LATCHES(-1)
```

関連キーワード

DEVICE

LATCHED

LATCH

#PLATCH

COARSE

ロボットに対して低精度ヌル許容誤差を可能にします。

構文

COARSE tolerance ALWAYS

使用上の留意事項

ALWAYS パラメータが指定されていない限り、次のロボット動作しか影響を受けません。

tolerance パラメータが指定されていると、ALWAYS 指定の有無に関わらず、その値は現在の実行サイクルで実行されるその後の COARSE プログラム命令に対するデフォルトになります。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、ステートメント FINE 100 ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

COARSE コマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、COARSE コマンドを実行するとエラーになります。

パラメータ

tolerance	現在の実行タスクに割り付けられているロボットの各ジョイントに使用される標準的低精度許容誤差のパーセンテージを指定するオプションの実数値、変数または式です。デフォルト値については以下の「詳細」の項を参照してください。
ALWAYS	デフォルト条件として COARSE を確立するオプションの修飾子です。FINE プログラムコマンドによって無効になるまで、COARSE は有効のままになります。ALWAYS を指定しない場合、COARSE コマンドは次のロボット動作だけに適用されます。

詳細

COARSE プログラムコマンドはロボット動作サーボ系統において低精度機能を可能にして、動作終了時にロボット関節の最終位置でより大きな誤差を許容するようにします。これにより、高精度が不要な場合により高速な動作の実行が可能になります。

ロボットを定速で動作させることで、サーボを高精度に設定できない場合、ベルトトラッキングアプリケーションにおいてより低い精度がときどき必要になります。

tolerance パラメータを指定する場合、新しい設定は次の動作の開始時に有効になります。ALWAYS 指定の有無に関わらず、この値は現在の実行サイクルで実行されるその後の COARSE コマンドに対するデフォルトになります。COARSE 許容誤差を変更しても FINE 許容誤差に影響はありません。

tolerance パラメータを省略した場合、割り付けられているロボットの最新の設定が使用されます。プログラムの実行が始まるか、新しい実行サイクルが開始されると、デフォルト設定は 100% に戻り、ロボットがプログラムに割り付けられていると仮定します。

例

以下の例では、次の動作だけに低精度機能を有効にしています。

```
COARSE
```

次の例では、次の動作に低精度機能を有効にし、各関節に対して、許容誤差設定を標準許容誤差の 150% に変更しています。

```
COARSE 150
```

次の例では、明確に無効になるまで低精度機能が有効です。

```
COARSE ALWAYS
```

関連キーワード

CALL

CALLS

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

SSTEP

XSTEP

CPOFF

次の動作（またはその後のすべての動作）の完了時およびヌル位置エラー時にロボットを停止するよう、eV+ システムに指示します。

構文

CPOFF *ALWAYS*

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボット動作しか影響を受けません。これは eV+ システムのデフォルト状態です。

CP システムスイッチが無効な場合、実行した CPOFF コマンドに関わらず連続経路処理は決して発生しません。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、CPON ALWAYS が仮定されます。

CPOFF プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットだけに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、CPOFF コマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として CPOFF を確立するオプションの修飾子です。CPOFF コマンドに ALWAYS が含まれる場合、CPON コマンドによって無効になるまで CPOFF は継続的に有効です。ALWAYS を指定しない場合、CPOFF コマンドは次のロボット動作だけに適用されます。

詳細

CPOFF が有効な場合、ロボットは次のロボット動作が完了すると停止し、最終位置誤差がヌルになります（必要な場合）。

BREAK プログラムコマンドが動作の後に実行されて、連続経路処理を停止させるのとは異なり、CPON および CPOFF は動作の前に実行され、次の動作コマンドの連続経路処理に影響を与えます。BREAK が 1 つの動作だけに適用されるのに対して、CPOFF はその後の動作すべてに適用することができます。

> **注:** BREAK プログラムコマンドは、その動作が終了するまでプログラムの実行をブロックすることで、連続経路処理を終了させます。CPOFF によって、軌道ジェネレータは、eV+ プログラムの先読み処理に影響を与えることなく、継続経路を終了させます。

例

次の例では「point1」への移動が完了し一度減速停止した後、ロボットを「point2」に移動させます。pos1 と pos2 の間のロボット動作はブレンド処理されません。

```
CPOFF ALWAYS
MOVES pos1
MOVES pos2
```

関連キーワード

BREAK

CP

CPON

SELECT （プログラムコマンド）

SELECT（実数値関数）

CPON

eV+ システムに対して、連続経路の一部として次の動作（またはその後の動作すべて）を実行するように命令します。

構文

CPON *ALWAYS*

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボット動作しか影響を受けません。

CP システムスイッチが無効な場合、実行した CPON コマンドに関わらず連続経路処理は決して発生しません。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、CPON ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

CPON プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットだけに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、CPON コマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として CPON を確立するオプションの修飾子です。ALWAYS を指定すると、CPOFF プログラムコマンドによって無効になるまで、CPON は有効のままになります。ALWAYS を指定しない場合、CPON コマンドは次のロボット動作だけに適用されます。

詳細

CPON が有効な場合、1 つの連続経路にまとめられた一連の動作オペレーションを実行することができます。それぞれの動作は指定した位置でロボットを停止することなく連続して実行されます。

BREAK プログラムコマンドが動作の後に実行されて、連続経路処理を停止させるのとは異なり、CPON および CPOFF は動作の前に実行され、次の動作コマンドの連続経路処理に影響を与えます。

> **注:** BREAK プログラムコマンドは、その動作が終了するまで eV+ プログラムの実行をブロックすることで、連続経路処理を終了させます。CPOFF によって、軌道ジェネレータは、eV+ プログラムの先読み処理に影響を与えることなく、継続経路を終了させます。

CPON コマンドを実行すると連続経路処理を発生させることができますが、以下の任意の状態では連続経路を中断し、CPON 機能が無効になります。

- 次の動作の完了前にその後の動作が実行されていない。
- CP システムスイッチが無効。

- 次の動作オペレーションの後に、明示的または暗示的に動作の終了を生じさせるオペレーション（例えば、BREAK）がある。

例

次の例では、ロボットに対して事前に他の方法を用いてブレンドしないように命令している場合に、ロボットは「pos1」および「pos2」と定義された位置の間でブレンド動作を開始します。

```
CPON ALWAYS
MOVES pos1
MOVES pos2
```

関連キーワード

BREAK

CP

CPOFF

SELECT（プログラムコマンド）

SELECT（実数値関数）

CYCLE.END

指定されたタスク内で実行中のプログラムが STOP プログラムコマンド（または同等のもの）を次に実行するときに、そのプログラムを終了します。

指定されたタスクにおいて実行されているプログラムが実行を完了するまで実行可能なプログラムの処理を中断します。

構文

CYCLE.END task_num, stop_flag

使用上の留意事項

指定したプログラムタスクが有効でない場合、CYCLE.END プログラムコマンドに効果はありません。

指定したプログラムタスクが実行を完了するまで、CYCLE.END プログラムコマンドはそのキーワードを含む実行プログラムを中断します。CYCLE.END コマンドによって実行が中断されている間にプログラムをアボートした場合、stop_flag が TRUE であれば、CYCLE.END コマンドが指定するプログラムタスクは終了したままになります。

パラメータ

task_num	モニタまたは終了するプログラムタスクを指定する、整数と解釈されるオプションの実数値、変数または式です。 タスク番号が指定されていない場合、CYCLE.END プログラムコマンドは常にタスク 0 にアクセスします。
stop_flag	論理値 TRUE または FALSE として解釈されるオプションの実数値、変数または式です。このパラメータを省略するか、パラメータが値 0 を持つとすると、指定したタスクは停止しないものの、CYCLE.END は以下に説明するようにその他のすべての機能を持ちます。このパラメータが 0 以外の値である場合、選択したタスクは現在のサイクルの末尾で停止します。

詳細

stop_flag パラメータの値が TRUE の場合、未実行であるプログラムサイクルの数に関わらず、指定したプログラムタスクは、STOP プログラムコマンド（または同等のもの）を次に実行するときに停止します。

> **注:** CYCLE.END は連続内部ループを持つプログラムを終了させません。該当するプログラムは ABORT キーワードで終了させる必要があります。

stop_flag パラメータに関わらず、このコマンドはプログラムが終了するまで待機します。終了中のプログラムが内部ループしているために、現在の実行サイクルが終了しない場合、CYCLE.END コマンドは永遠に待機していまいます。

終了させるプログラムを待機している CYCLE.END から次に進むためには、CYCLE.END コマンドを実行したプログラムタスクに対して ABORT モニタコマンドを発行することで CYCLE.END を待っているプログラムをアボートします。

例

次の例では、プログラムタスクを別のプログラムタスクから開始させる方法を実演しています。ABORT および CYCLE.END プログラムコマンドを使用して、指定したプログラムタスクがもはや有効でないことを確認します。

```
ABORT 3
CYCLE.END 3
EXECUTE 3 new.program
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

CYCLE.END（モニタコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

KILL（モニタコマンド）

KILL（プログラムコマンド）

STATUS（モニタコマンド）

STATUS（実数値関数）

STOP

DECOMPOSE

ロケーション値の個々の成分の実数値を抽出します。

構文

DECOMPOSE array_name[index] = **location**

パラメータ

array_name	要素を定義する実数値配列の名称です。
index	定義される最初の配列要素を特定するオプションの整数値です。index を省略すると0 が仮定されます。多次元配列を指定する場合、数値を割り当てると右端の index だけがインクリメントします。
location	ロケーション値であり、成分値に分解されます。これは変換値またはプレシジョンポイント値にすることができ、変数またはロケーション値関数で定義することができます。

詳細

このプログラムコマンドは、値を指定した位置の成分に対応する命名された配列の連続する要素に割り当てます。

位置が変換値である場合、X、Y、Z、ヨー、ピッチ、ロールに対応する 6 要素が定義されます。

位置がプレシジョンポイント値である場合、個々の関節の位置に対応するロボット関節の番号に応じて 1～12 の要素が定義されます。

例

以下の例では、変換 "part" の成分を配列 "x" の要素 0 から 5 に割り当てます。

```
DECOMPOSE x[] = part
```

以下の例では プレシジョンポイント"#pick" の成分を配列要素 "angles[4]" およびそれ以降に割り当てます。

```
DECOMPOSE angles[4] = #pick
```

次の例では、多次元変数配列「point」に分解します。

```
SET point[0] = TRANS(1,10,2,10,3,10)
SET point[1] = TRANS(1,20,2,20,3,20)
SET point[2] = TRANS(1,30,2,30,3,30)

FOR i = 0 TO 2
      DECOMPOSE a[i,] = point[i]
  END

  FOR i = 0 TO 2
      FOR j = 0 TO 5
          TYPE "a[", i, ",", j, "]", a[i,j]
```

```
        END
    END
```

関連キーワード

#PPOINT

TRANS

DEF.DIO

各種キーワードで使用するためにバーチャルデジタル I/O に標準 eV+ 信号番号を割り当てます。

構文

DEF.DIO signal = count, operation

使用上の留意事項

このプログラムはホスト I/O 信号番号 4001～4009 をサポートしていません。詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

パラメータ

signal	出力に関しては 17～505、入力に関しては 1017～1505 の V+ 信号番号を示す値です。
count	希望する信号の数です。デフォルトは 8 で範囲は 1～32 です。
operation	オプションの整数または実数値式であり、実行するアクションを示します。省略した場合またはプラスの場合、信号が設定されます。マイナスの場合、信号は削除されます。

詳細

それぞれの DEF.DIO オペレーションは 32 の入力または 32 の出力しかマッピングできません。ただし、複数のオペレーションを作成すると、より多くの信号をマッピングすることができます。このようにしてマッピングされた信号はソフト信号と同等であり、SIG 関数または SIGNAL および BITS プログラムコマンドを使用して読み取りまたは設定をすることができます。

信号が既にリアルまたはバーチャルなデジタル I/O にマッピングされている場合、エラー（-405）「Illegal digital signal」（不正なデジタル信号）が返されます。

例

次の例では入力信号 1033～1048 をバーチャル信号として定義します。

```
DEF.DIO 1033 = 16
```

関連キーワード

BITS

IO

RESET

RUNSIG

SIG

SIG.INS

SIGNAL

DEFBELT

コンベアトラッキングロボットを使用するためのベルト変数を定義します。

構文

DEFBELT %belt_var = nom_trans, belt_num, vel_avg, scale_fact

使用上の留意事項

ロボットが指定したベルト変数に対して移動している間は、DEFBELT プログラムコマンドを実行することはできません。

このコマンドを使用してベルト変数を初期化すると、そのウィンドウパラメータが設定されてそのロボットの作業容積内の任意の位置を可能にしますが、ベルトウィンドウは設定されません。詳細については、WINDOW プログラムコマンドを参照してください。

パラメータ

%belt_var 定義するベルト変数の名称です。

nom_trans コンベアベルトの位置および向きを定義する変換値です。これは変換変数、変換値関数または複合変換によって与えることが可能です。

ノミナル変換の X 軸はベルトの移動方向を定義します。通常、ベルトは +X 方向に沿って移動します。この変換で定義される X-Y 面はベルト表面に平行です。ノミナル変換によって定義される(X, Y, Z)位置は、ロボットを基準としたベルトのおよその中心を定義します。

belt_num ベルトの瞬間位置を読み取るために使用されるエンコーダの番号です。1～99 の番号を持つエンコーダを指定することができます。これは、定数、変数または式で指定することができます。

vel_avg このパラメータは現在無視されていますが、なにか値を与える必要があります。

scale_fact コンベアベルトの動きをコンベアに取り付けられているエンコーダのカウント数に関連付けるキャリブレーション定数です。この値は定数、実数型変数、または式として提供することができ、ベルトの単位移動量(mm)とエンコーダのカウント数の比率と解釈されます。

詳細

コンベアベルトはベルト変数でモデリングされます。ベルト変数には DEFBELT プログラムコマンドのパラメータの他に次の情報を含みます。

- ベルトに沿ったロボットの作業範囲を定義するウィンドウパラメータ（WINDOW プログラムコマンドを使用して設定）。

- ベルト基準フレームの原点を調整するために使用されるエンコーダオフセット（SETBELT プログラムコマンドを使用して設定）。

ベルト変数には次の特性があります。

- ベルト変数名の前には必ずパーセント記号（%）が付いていなければなりません。他の点では、変数名に対する一般規則が適用されます。
- ベルト変数配列（例えば %b[x]）は許容されます。
- ベルト変数は他の変数同様、サブルーチンパラメータとして引き渡すことができます。
- ベルト変数は、DEFBELT コマンドを使用した場合に限って定義できます。それらのための割り当てコマンドはありません。次のステートメントは無効です。

```
%new_belt = %old_belt
SET %new_belt = %old_belt
```

- ベルト変数を大量記録装置に保存することはできません。
- DEFBELT コマンドでパラメータを定義するために使用される変数は、大量記録装置に保存することができます。

例

次の例では、ベルト変数「%belt.var」を定義します。「b.num」の値はこのベルト変数に関連するエンコーダの番号である必要があります。変数「b.num」はベルトの位置および向き、速度の平滑化およびエンコーダのスケール係数を記述するデータ配列のインデックスとして使用されます。

```
DEFBELT %belt.var = belt.nom[b.num], b.num, v.avg[b.num],
belt.sf[b.num]
```

関連キーワード

BELT（実数値関数）

BELT.MODE

BSTATUS

SETBELT

WINDOW（プログラムコマンド）

WINDOW（実数値関数）

DEPART

現在のロケーションから離れるロボットモーションをジョイント補間モーションを用いて開始します。

構文

DEPART distance

使用上の留意事項

このプログラムコマンドは、タスクがロボットをアタッチしている限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行するとエラーになります。

パラメータ

distance 現在のロボットのロケーションと目的のロケーションとの間の、ロボットツール Z 軸に沿った距離（mm）を指定する実数値式です。

プラスの距離は、現在のロケーションからツールを後方（ツール Z 軸のマイナス方向）へ移動します。マイナスの距離は、ツールを前方（ツール Z 軸のプラス方向）へ移動します。

詳細

このプログラムコマンドにより、現在のロケーションから、現在のツール Z 軸に沿って測定して、所定の距離オフセットした新しいロケーションへのジョイント補間ロボットモーションが開始されます。

例

次の例では、ジョイント補間モーションを用いて、ロボットツールを現在のロケーションから 80 mm 戻します。

```
DEPART 80
```

関連キーワード

APPRO

APPROS

DEPARTS

MOVE

MOVEF

MOVES

SELECT（プログラムコマンド）

SELECT（実数値関数）

DEPARTS

現在のロケーションから離れるロボットモーションを直線モーションを用いて開始します。

構文

DEPARTS distance

使用上の留意事項

DEPARTS プログラムコマンドは直線モーションを発生させ、その間、設定の変化は許容されません。

このコマンドは、タスクがロボットをアタッチしている限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行するとエラーになります。

パラメータ

distance	現在のロボットのロケーションと目的のロケーションとの間の、ロボットツール Z 軸に沿った距離（mm）を指定する実数値式です。 プラスの距離は、現在のロケーションからツールを後方（ツール Z 軸のマイナス方向）へ移動します。マイナスの距離は、ツールを前方（ツール Z 軸のプラス方向）へ移動します。

詳細

このコマンドにより、現在のロケーションから、現在のツール Z 軸に沿って測定して、所定の距離オフセットした新しいロケーションへの直線ロボットモーションが開始されます。

例

次の例では、ロボットツールを現在のロケーションから直線パスに沿って（2*offset）mm 引きます。

```
DEPARTS 2*offset
```

関連キーワード

APPRO

APPROS

DEPART

MOVE

MOVEF

MOVES

SELECT（プログラムコマンド）

SELECT（実数値関数）

DETACH

指定したデバイスをアプリケーションプログラムの制御から解放します。

構文

DETACH (logical_unit)

使用上の留意事項

ロボットを割り付け解除すると連続経路動作において中断を発生させます。

指定されたデバイスにおいてディスクファイルが開いていると、DETACH によって自動的に FCLOSE が作動します。

EXECUTE キーワードを処理してそのタスクを開始すると、ロボットはタスク 0 に自動的に割り当てられ、DRY.RUN システムスイッチは無効になります。実行プログラムが開始されると、その他すべての論理ユニットは自動的に割り付け解除されます。自動的割り付け解除を生じさせるその他のイベントを以下に示します。

パラメータ

logical_unit	割り付け解除されるデバイスを指定する整数と解釈されるオプションの実数値、変数または式です。論理ユニット番号に関する説明については、ATTACH プログラムコマンドを参照してください。 論理ユニット番号が省略されている場合、括弧を省略できます。これにより、ロボットは割り付け解除されます。

詳細

このプログラムコマンドは指定されたデバイスをアプリケーションプログラムの制御から解放します。デバイスが割り付けられていなくても、エラーは発生しません。

ATTACH プログラムコマンドを使用すると、指定されたデバイスの制御をプログラムに戻すことができます。

logical_unit が 0 である、または 省略されている場合、プログラムはロボットの制御を放棄します。ロボットが割り付け解除されている間は、ロボットの電源を OFF および ON にすること、ペンダントを使用してロボットを動かすこと、およびシステムコントローラに複数のロボットが接続していれば、別のロボットを選択することができます。ロボットを割り付け解除すると 1 回 16 ms のシステムサイクル遅延が発生します。

これは、特定の作動に対してロボットのあるべき位置を定義する必要のあるアプリケーションに有用です。ティーチングプログラムはロボットを DETACH することが可能であり、方向をモニタウィンドウまたはペンダントに出力します。次いでこのペンダントを使用してロボットを希望する位置に移動することができます。モニタウィンドウまたはペンダントを使用して入力を受け入れることができます。

ディスク論理ユニットが割り付け解除された場合、対応する ATTACH プログラムコマンドで指定されたデバイスは無視されます。デフォルトのデバイスが希望するものでない場合、その後 ATTACH コマンドを使用してもう一度そのデバイスを指定する必要があります。

以下のイベントでは、すべての論理ユニット（当該ロボットを除く）が影響のあるプログラムタスクから自動的に割り付け解除されます。

- EXECUTE キーワードの処理
- KILL キーワードの処理
- ZERO モニタコマンドの処理
- プログラム実行の通常の終了

プログラムの実行が異常終了すると、ティーチペンダントは割り付け解除されますが、その他すべてのデバイスは割り付けられたままになります。プログラム実行の異常終了は HALT または STOP プログラムコマンド以外の原因に関連するものです。そのタスクがその後再開されると、モニタウィンドウおよびペンダントが終了前に割り付けられていたのであれば、プログラムはそれらを自動的に再度割り付けます。

> **注:** その間に別のプログラムタスクをペンダントに割り付ける可能性があります。この場合、停止していたタスクが再始動すると、エラーメッセージが発生します。

例

次の例では、ロボットのプログラム制御を解放します。

```
DETACH
```

次の例では、ペンダントのプログラム制御を中断します。

```
DETACH (1)
```

関連キーワード

ATTACH

DISABLE

1 つ以上のシステムスイッチを OFF にします。

構文

DISABLE switch, ..., switch

使用上の留意事項

指定したシステムスイッチがインデックス修飾子を受け入れ、そのインデックスが 0 であるか括弧付きまたは括弧なしで省略されている場合、そのスイッチ配列のすべての要素が有効になります。

パラメータ

switch OFF にするシステムスイッチの名称です。

その名称はそのスイッチを一意に特定する最小長さに短縮することができます。例えば、MESSAGES システムスイッチは、文字 ME で始まる他のスイッチがないので "ME" と呼称することができます。

詳細

システムスイッチを OFF にするとその制御機能は無効になり、利用できなくなります。ENABLE キーワードを使用してスイッチを ON にすると関連する機能が有効になり、使用できるようになります。

> **重要:**このシステムスイッチはすべてのプログラムタスクで共有されています。複数のタスクでスイッチを有効または無効にする場合は、注意するようにしてください。そうしないと、1 つ以上のタスクにおいてスイッチが正しく設定されない可能性があります。
>
> **注:** DRY.RUN システムスイッチを無効にしても、次にタスク 0 に対して EXECUTE キーワードを処理するか、ロボットに対して ATTACH プログラムコマンドを実行するか、または CALIBRATE キーワードを処理するまでは、効果はありません。

SWITCH モニタコマンドまたは SWITCH 実数値関数を使用して、システムスイッチのステータスをいつでも判定することができます。DISABLE プログラムコマンドを使用した場合と同様に、SWITCH プログラムコマンドを使用してスイッチを無効にすることができます。

例

次の例では、MESSAGES システムスイッチを OFF にします。

```
DISABLE MESSAGES
```

関連キーワード

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

SWITCH（モニタコマンド）

SWITCH（プログラムコマンド）

SWITCH（実数値関数）

DO

DO プログラム構造体を作成します。

構文

DO

使用上の留意事項

DO プログラム構造体には UNTIL コマンドを含める必要があります。

詳細

DO 構造体は、制御式に基づいてキーワードグループの実行を制御する方法を提供します。DO 構造体の構文を以下に示します。

```
DO
    group_of_steps
UNTIL logical_expression
```

DO 構造体の処理は以下のように説明することができます。

1. コマンドステップのグループを実行する。
2. 論理式を判定する。結果が FALSE の場合、項目 1 に戻る。それ以外の場合は項目 3 に進む。
3. UNTIL コマンドの後の最初のコマンドからプログラムの実行を継続する。

この構造体を使用する場合は、キーワードで囲まれているグループ内である動作が発生し、この構造体から抜けるべき時には論理式の結果をFALSE からTRUE に変化させることが想定されます。または、logical_expression を、デジタル I/O シグナルの状態を判定する式に置き換えることができます（以下の例を参照してください）。

DO 構造体内のキーワードグループは必ず 1 回以上実行されます。この機能のバリエーションについては WHILE コマンドを参照してください。

DO コマンドと UNTIL コマンドの間にキーワードを入れる必要はありません。そういったキーワードがない場合、実行プログラムが UNTIL コマンドに続くキーワードによって継続する際に、適合するものがあるまで、UNTIL の基準を継続的に判定します。

例

次の例では、DO 構造体を使用して、ある場所から別の場所への部品の移動を含むタスクを制御します。このシーケンスでは、部品バッファが満杯になると、デジタル信号ライン「buffer.full」が ON 状態に変わります。その後、ロボットは一般的に異なる動作シーケンスを実行します。

```
.
.
DO
    CALL get.part()
    CALL put.part()
UNTIL SIG(buffer.full)
```

.
.

関連キーワード

DO

EXIT

NEXT

UNTIL

WHILE

DOS

文字列式で定義されるキーワードを実行します。

構文

DOS string, error

使用上の留意事項

このコマンドを実行する前に、文字列を ASCII から eV+ で使用されている内部表現に変換する必要があります。このためDOS コマンドはキーワードを直接発行するより実行速度が遅くなります。

この文字列は宣言ステートメントや制御構造ステートメントのほとんどを定義することができません。

文字列がコマンド行または空白行を定義している場合、DOS コマンドは無視されます。

このコマンドで参照されている変数が現在のプログラムコンテキスト内で見つからない場合、その変数はグローバルであると仮定されます。このコマンドによって（例えば、割り当てステートメント内で）生成された新しい変数は、グローバルタイプとして生成されます。通常変数タイプの確認が実行され、タイプに矛盾があるとエラーが発生します。

単一行の制御ステートメント GOTO, IF ...GOTO、CALL、および CALLS は許可され、正常に実行します。マルチライン制御構造（CASE ...END、IF ...ELSE ...END など）は DOS コマンドによって実行することはできません。

パラメータ

string	実行するキーワードを定義するオプションの文字列定数、変数または式です。このコマンドにはラベルフィールド（無視される）が含まれることがあり、その後に標準コメントフィールドが続くことがあります。先頭および末尾のスペースやタブは無視されます。
error	コマンドによって作成された構文解析または実行エラーを受けるオプションの実数型変数です。コマンドが無事に処理されるとこの値は 1 に設定されます。コマンドに問題が生じると、標準 eV+ エラー番号を返します。 このパラメータを省略している場合にエラーが発生すると、実行プログラムが停止し、該当するエラーメッセージが表示されます。

詳細

DOS（DO String）コマンドはランタイム中にプログラムを変える方法を提供します。まるでプログラムにおいて通常のキーワードとして入力されたようにして、文字列式で定義された内蔵キーワードを実行します。

このキーワードは現在のプログラムのコンテキストにおいて実行されます。任意のサブルーチン引数、自動変数またはローカル変数にアクセスできます。

例

以下の例では、変数 "var" に値 123 を割り当てます。"var" が未定義の場合、"var" という名称の新しいグローバル変数が作成されます。エラーがあると、実行プログラムが停止します。

```
DOS "var = 123"
```

以下の例では、文字列型変数 $ins に含まれるキーワードを実行します。エラーが発生した場合、実数値変数「status」 に eV+ のエラーコードを配置して、実行を継続します。

```
DOS $ins, status
```

関連キーワード

DO

MCS

DRIVE

ロボットの個々のジョイントを動かします。

構文

DRIVE joint, change, speed

使用上の留意事項

DRIVE プログラムコマンドは、タスクがロボットをアタッチしている限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、DRIVE コマンドを実行するとエラーになります。

パラメータ

joint 移動するロボットのジョイント番号です。これは、定数、変数または式で指定することができます。

change そのジョイントポジションで希望する変化量です。これは、定数、変数または式で指定することができます。この値はプラスまたはマイナスにすることができます。

この値は使用されるユニット内で解釈されて、ジョイントのポジションを測定するために使用されます。回転式ジョイントの変化は、そのジョイントの移動角度(°)となります。直動ジョイントの変化は、そのジョイントの移動距離(mm)となります。

speed モーションに対して使用する一時的なプログラム速度であり、現在のプログラム速度設定のパーセンテージとみなされます。これは、定数、変数または式で指定することができます。

詳細

DRIVE プログラムコマンドは 1 つの指定したロボットジョイントに作用して change パラメータで指定した量（° または mm）だけ位置を変化させます。ジョイントパラメータは 1、2、...、n にすることができます。ここで n はそのロボットのジョイント数です。

モーション速度はこのコマンドが与える速度とモニタ速度設定の組み合わせによって制御されます。標準プログラム速度設定は使用されません。モーション速度に関する情報については、SPEED モニタコマンドおよび SPEED プログラムコマンドを参照してください。

DURATION プログラムコマンドが確立した時間設定も動作実行時間に影響を与えます。

例

次の例では、ジョイント 2 を モニタ速度の 75% の速度で、マイナス方向に 62.4° 駆動させることで同ジョイントの角度を変更します。

```
DRIVE 2,-62.4,75
```

関連キーワード

SELECT

SELECT（プログラムコマンド）

DURATION

以降のロボットモーションの最大実行時間を設定します。

構文

DURATION time *ALWAYS*

使用上の留意事項

ALWAYS パラメータを指定していない場合、次のロボットモーションだけが影響を受けます。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、ステートメント DURATION 0 ALWAYS が仮定されます。

DURATIONプログラムコマンドは、 DRIVE プログラムコマンドに影響します。

SPEED モニタコマンドの設定は DURATION 設定の結果に影響を与えます。

DURATION コマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、DURATION コマンドを実行するとエラーになります。

パラメータ

time 次のロボットモーションの実行にかける最小時間(s)を指定する実数式です。

この値が 0 の場合、ロボットモーションはこの時間を考慮せずに実行され、SPEED および ACCEL キーワードの該当する値だけを使用します。

ALWAYS 新しい時間が有効となる範囲を決定するオプションのキーワードです。

ALWAYS が含まれる場合、この時間設定が別の DURATION コマンドによって変更されるまで、指定された時間がその後のすべてのロボットモーションに適用されます。ALWAYS が含まれない場合、指定された時間は次のロボットモーションだけに適用されます。

詳細

このコマンドは、以降のロボットモーションの最小実行時間を設定します。DURATION コマンドで指定した時間が、現在の動作速度および加速度設定を使用して eV+ ロボットモーション軌道ジェネレータが計算した時間よりも短い場合、その時間は何の効果もありません。この時間が軌道ジェネレータの計算した時間よりも長い場合、その動作は遅くなって、経過時間は指定した時間にほぼ等しくなります。

> **注:** 実際のモーション時間は、量子化効果および加減速プロファイルによってこの時間設定とは若干異なります。

DURATION コマンドは動作全体の時間を指定するのではなく、定速部の最小時間に加減速部の 1/2 を加えた時間を指定します。個々のモーションを滑らかにつないだ連続パスモーションでは正しい時間を得ますが、単一モーションは指定時間よりも長くなります。モーション時間は基本的に、より長い時間を与える値を使用して、DURATION または SPEED の値で定義されます。

定期的な外部信号の値を使用して、ロボットモーション中にロボットのパスを連続的に変更する場合を考慮してください。DURATION コマンドを使用して、モーション実行時間をセンサのサンプリングレートおよび処理時間に一致させることができます。こうすることで、新しい情報を処理しながら、確実にロボットを動かし続けることができます。

例

次の例では、外部センサを評価して、ロボットを計算で得られた位置に移動します。このシーケンスは 96 ms （6/TPS 秒）の間隔で 20 回繰り返されます。ここでは、eV+ 軌道ジェネレータに対して 16 ms のデフォルト時間を仮定しています。モーション速度は、DURATION 設定によって確実に調整されるようにするために非常に大きく設定されています。

```
DURATION 6/TPS ALWAYS
SPEED 200 ALWAYS

    FOR i = 1 TO 20
        CALL read.signal(loc)
        MOVE loc
    END
```

関連キーワード

ACCEL

DURATION

SELECT

SELECT(実数値関数)

SPEED(モニタコマンド)

SPEED (プログラムコマンド)

ELSE

IF ... THEN 制御構造のステートメントの代替グループを区切ります。

構文

ELSE

使用上の留意事項

ELSE は、IF ...THEN ...ELSE ...END 制御構造内でのみ使用できます。

詳細

IF logical_expr THEN 制御構造内の論理式の値が 0 以外の場合に実行されるステートメントグループの末尾、およびこの値が 0 の場合に実行されるステートメントグループの先頭を示します。

例

次のプログラム例では、変数「input.signal」が定義されているかどうかを判定します。定義されている場合、プログラムは「input.signal」の値が示す信号を確認し、その設定に応じて異なるメッセージを入力します。外側の IF には ELSE 節は含まれません。

```
IF DEFINED(input.signal) THEN
    IF SIG(input.signal) THEN
        TYPE "The input signal is ON"
    ELSE
        TYPE "The input signal is OFF"
    END
END
```

関連キーワード

IF ...THEN

ENABLE

1 つ以上のシステムスイッチを ON にします。

構文

ENABLE switch, ..., switch

使用上の留意事項

ENABLEはプログラム実行中に使用することができます。

指定したシステムスイッチがインデックス修飾子を受け入れ、そのインデックスが 0 であるか括弧付きまたは括弧なしで省略されている場合、そのスイッチ配列のすべての要素が有効になります。

パラメータ

switch ON にするシステムスイッチの名称です。

その名称はそのスイッチを一意に特定する最小長さに短縮することができます。例えば、MESSAGES スイッチは、文字 ME で始まる他のスイッチがないので「ME」と呼称することができます。

詳細

システムスイッチを ON にするとその制御機能は有効になり、利用できるようになります。DISABLE キーワードを使用してスイッチを OFF にすると関連する機能が無効になり、使用できなくなります。

> **重要:**このシステムスイッチはすべてのプログラムタスクで共有されています。複数のタスクでスイッチを有効または無効にする場合は、注意するようにしてください。そうしないと、1 つ以上のタスクにおいてスイッチが正しく設定されない可能性があります。

SWITCH モニタコマンドまたは SWITCH 実数値関数を使用して、システムスイッチのステータスをいつでも判定することができます。SWITCH プログラムコマンドを使用した場合と同様に、ENABLE プログラムコマンドを使用してスイッチを設定することができます。

例

次の例では、MESSAGES システムスイッチを ON にします。

```
ENABLE MESSAGES
```

関連キーワード

DISABLE

DISABLE

SWITCH

SWITCH（プログラムコマンド）
SWITCH（実数値関数）

END

制御構造体の末尾を示します。

構文

END

使用上の留意事項

すべての END プログラムコマンドは、CASE、FOR、IF ... GOTO、IF ...THEN、またはWHILE 制御構造の一部とする必要があります。

詳細

すべての CASE、FOR、IF ... GOTO、IF ...THEN、またはWHILE 制御構造はこの END プログラムコマンドによって末尾が示されている必要があります。プログラムに適正な数の END コマンドが存在にしないときに、プログラムの編集を行うと、eV+ エディタにはエラーメッセージが表示されます。

例

次の例では、WHILE 制御構造でのEND の使用方法を示します。

```
WHILE TRUE DO
    ;statements
END
```

次の例では、FOR 制御構造でのEND の使用方法を示します。

```
FOR i = 0 TO 10
    ;statements
END
```

次の例では、IF ... THEN 制御構造でのEND の使用方法を示します。

```
IF (condition) THEN
    ;statements
ELSE
    ;statements
END
```

次の例では、CASE 制御構造でのEND の使用方法を示します。

```
CASE condition OF
    VALUE (condition):
        ;statements
    VALUE (condition):
        ;statements
     VALUE (condition):
        ;statements
     ANY
        ;statements
END
```

次の例では、IF ... GOTO 制御構造でのEND の使用方法を示します。

```
    ATTACH(dlun, 4) "DISK"
    IF IOSTAT(dlun) < 0 GOTO 100
```

```
        FOPENW(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
            ...
        FCLOSE(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
        DETACH(dlun)
    100 IF IOSTAT(dlun) < 0 THEN
            TYPE $ERROR(IOSTAT(dlun))
        END
```

関連キーワード

CASE

FOR

IF ... GOTO

IF ...THEN

WHILE

ESTOP

非常停止信号を受信したときと同様にロボットを停止します。

構文

ESTOP robot_num

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値と解釈される実数値、変数、または式です。オプションです。 このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

このキーワードは即座に非常停止信号を発行して通常の電力停止シーケンスを進めます。これはペンダントの E-STOP ボタンを押すのと機能的に同じです。

robot_numで指定されたロボットが接続されていなくても、エラーは発生しません。

例

次の例ではロボット2 に対して非常停止と電力停止シーケンスを開始します。

```
ESTOP 2
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

BRAKE

ESTOP（プログラムコマンド）

PANIC（モニタコマンド）

PANIC （プログラムコマンド）

STATE

EXECUTE

制御プログラムの実行を開始します。

構文

EXECUTE /C **task program** (param_list), cycles, step, priority[i]

使用上の留意事項

プログラムは指定されたプログラムタスクとして実行中であってはいけません。

パラメータ

/C	選択されたロボットを条件付きで割り付けるオプションの修飾子です。修飾子には、タスク0の実行開始時にのみ効果があります。
task	有効にするプログラムを指定する実数値または式です。プログラムタスクに関する詳細情報については*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
program	実行する予定のプログラムの名称です。
param_list	コンマで区切られた、定数、変数または式のオプションのリストであり、指定したプログラムの .PROGRAM ステートメント内の引数のタイプおよび数に対応する必要があります。プログラムによって引数が要求されていない場合、このリストは空になりますが、括弧は入力する必要があります。 プログラムパラメータは、希望すれば、省略するパラメータをカンマを使ってスキップすることで省略できます。リストの最後のパラメータが省略されている場合、コンマは必要ありません。省略されたパラメータは呼び出されたプログラムに未定義として引き渡され、DEFINED 実数値関数で検出することができます。 自動変数およびサブルーチン引数は EXECUTE コマンドにおける参照では引き渡すことができません。これらは値として引き渡される必要があります(CALL プログラムコマンド参照)。 パラメータは開始される新しいタスクのコンテキスト内で判定されます。
cycles	実行されるプログラム実行サイクルの数を指定する、整数と解釈されるオプションの実数値、変数または式です。省略した場合、サイクルカウントは 1 と仮定されます。無制限のサイクルについては任意のマイナス値を指定します。最大ループカウント値は、32,767です。
step	実行プログラムを開始するステップを指定する、整数と解釈されるオプションの実数値、変数または式です。省略した場合、プログラムの実行はそのプログラム内の実行可能ステートメントで始まります(先頭の空白およびコメント行、ならびにすべての AUTO および LOCAL ステートメントの後)。

priority[]	そのタスク内のデフォルトの実行優先度を無効にするために eV+ によって使用される、オプションの整数と解釈される実数型配列です。この配列には 16 の要素があり、16 ms 間隔でプログラムの優先度を指定します。指定されている場合、この要素は -1 以上 64 以下である必要があります。詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。
i	影響のあるロボットの番号を示す整数値と解釈されるオプションの実数値、変数または式です。

詳細

このコマンドは、指定された制御プログラムの実行を開始します。このプログラムは指定されたプログラムの step から cycles 回実行されます。

プログラムが他のプログラムの実行を開始すると、開始させた方のプログラムは STATUS および ERROR 関数を使用してもう一方のプログラムのステータスを監視します。

オプションの /C 修飾子はタスク0 で実行を始める場合に限って有効になります。/C を指定しない場合、ロボットを割り付けることができないと、タスク 0 に対する EXECUTE コマンドはうまく実行されません。割り付けのためには、ロボットのキャリブレーションが実施されていて、アーム電源が有効になっているかまたは DRY.RUN システムスイッチが有効である必要があります。/C を指定する場合、タスク 0 に対する EXECUTE コマンドはロボットを割り付けようとしますが、ロボットを割り付けることができない場合、エラー表示なしにタスク 0 の実行が継続してしまいます。

プログラムの実行を開始する際は、特定のデフォルト条件が仮定されます。それらは以下のステートメントと同等です。

```
CPON ALWAYS
DURATION 0 ALWAYS
FINE 100 ALWAYS
LOCK 0
MULTIPLE ALWAYS
NULL ALWAYS
OVERLAP ALWAYS
SPEED 100,100 ALWAYS
SELECT ROBOT = 1
```

以降の動作のために、ロボットの設定は保存されます。

STOP プログラムコマンドを実行した場合、RETURN プログラムコマンドをトップレベルプログラムで実行した場合、またはそのプログラムの最後に定義したステップに遭遇した場合、実行サイクルは終了します。cycles の値は -32,768 ～ 32,767 の範囲にすることができます。cycles が省略された場合や値 0 または 1 を持つ場合、プログラムは 1 回だけ実行されます。cycles にマイナス値を指定すると、HALT プログラムコマンドを実行する、エラーが発生する、またはプログラムの実行がアボートされるまで、HALT プログラムコマンドを実施します。

> **注:** 実行サイクルが開始される都度、実行パラメータはデフォルト値にリセットされます。これには動作速度、ロボット設定およびサーボモードが含まれます。現在選択しているロボットは変更されません。

step を指定すると、そのステップを最初のパスとしてプログラムの実行が始まります。連続したcyclesが、プログラムの最初の実行可能ステップから必ず開始されます。

すべてのEXECUTE パラメータは開始される新しいタスクのコンテキスト内で判定されます。これにより、EXECUTE コマンドを使用し、タスク従属値（TASK 関数など）を引き渡そうとすると予期せぬ結果になる場合があります。こうした事態では、タスク従属値に呼び出し側タスクを反映したい場合、タスク従属値を変数に割り当て、その変数を引き渡す必要があります。

例

次の例では、「assembly」という名称のプログラムの実行を（タスク 0 として）開始し、HALT プログラムコマンドが実行される、エラーが発生する、またはプログラムの実行がアボートされるまで実行を継続します。

```
EXECUTE 0 assembly, -1
```

以下の例では、プログラムタスク 2 を使用して "test" という名称のプログラムの実行を開始します。パラメータ値 1 および 2 がプログラムに引き渡されます。

```
EXECUTE 2 test(1,2)
```

次の例では、アプリケーションプログラムを別のアプリケーションプログラムから開始する方法を実演しています。ABORTおよび CYCLE.END プログラムコマンドを使用して、指定したプログラムタスクがもはや有効でないことを確実にしています。

```
ABORT 3
CYCLE.END 3
EXECUTE 3 new.program
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

CALL （プログラムコマンド）

CYCLE.END（モニタコマンド）

CYCLE.END （プログラムコマンド）

EXECUTE（モニタコマンド）

KILL（モニタコマンド）

KILL （プログラムコマンド）

PRIME

PROCEED

RETRY

SSTEP

STATUS

STATUS（実数値関数）

XSTEP

EXIT

制御構造体の n 番目にネストされたループに続くステートメントへ分岐します。

構文

EXIT count

使用上の留意事項

このコマンドは FOR、WHILE、および DO 制御構造体で使用されます。

パラメータ

count	終了するネストされた構造体の数を指定するオプションの整数値です (式および変数は許容されません)。デフォルト値は 1 です。

詳細

EXIT コマンドに到達すると、その制御構造体を終了し、実行を継続して、終了した最も外側の制御構造体に続く最初のコマンドを実行します。

例

以下の例では、入力信号 1001 を設定すると、1 つの制御構造体を終了します。1002 を設定すると、3 つ制御構造体を終了します。

```
FOR i = 1 TO 40
     WHILE ctrl.var DO
         DO
             IF SIG(1002) THEN
                 EXIT 3
             END
             IF SIG(1001) THEN
                 EXIT
             END
         UNTIL FALSE
         count = count+1
     END
END
```

関連キーワード

DO

FOR

NEXT

WHILE

EXTERNAL

eV+ と NJ シリーズロボット統合CPU ユニット の間で共有される変数を宣言します。

構文

EXTERNAL variable, ..., variable

使用上の留意事項

EXTERNAL ステートメントはプログラム内の実行可能なステートメントの前に必ず現れなければなりません。

外部変数はNJ シリーズロボット統合CPU ユニット内でも同じ名前で宣言する必要があります。詳細については、*『NJ シリーズ ロボット統合 CPU ユニット ユーザーズマニュアル (SBCE-436)』* を参照してください。

外部変数はグローバル変数と同様にして V+ プログラムとモニタウィンドウからアクセスすることができます。

この変数は自動的に倍精度データタイプになります。

外部変数の値が NJ シリーズロボット統合CPU ユニット において定義されるデータタイプの範囲外である場合、そのユニットにおいて未定義の値を持ちます。

外部変数は同じプログラム内の別の変数タイプ（GLOBAL、LOCAL、AUTO）として同じ名称で宣言することはできません。

パラメータ

variable 倍精度データタイプとしての変数の名称です。それぞれの変数は単純な変数または配列である場合があります。

配列を使用する場合、配列のサイズをNJ シリーズロボット統合CPU ユニットの宣言において定義し変数パラメータの配列サイズと一致させる必要があります。配列変数にインデックスを指定してはいけませんが、変数名の後に括弧 [] を含める必要があります。配列サイズの範囲は 0 から99 で 1 次元だけとし、固定長の配列が許容されます。

変数名については以下の項の詳細内容を参照してください。

詳細

外部変数を宣言すると、任意のV+ タスクで実行される任意のV+ プログラムで使用できるようになります。

外部変数を宣言すると、その変数が消去されるか、またはその変数を参照するすべてのV+ プログラムがシステムメモリから削除されない限り、その変数を任意の実行プログラムで使用できます。

以下の条件のとき、外部変数の値をNJ シリーズロボット統合CPU ユニットから読み出すとエラーが発生します。

- NJ シリーズロボット統合CPU ユニット に変数が存在しない（-406、プログラム名または変数名未定義）
- 配列の制限を超過（-404、不正な配列インデックス）
- 変数タイプがサポートされていないまたはタイプが一致しない（-404、不正な配列インデックス）

変数名

外部変数の名称は eV+ システム内および NJ シリーズロボット統合CPU ユニット で同一である必要があります。

変数名はアルファベット文字（a～z）で始まる必要があり、0～9、a～z、ピリオドおよびアンダースコアを含めることができます。

変数名の最大長は 15 バイトです。

例

次の例では、外部変数「myvar」を宣言してこの外部変数にアクセスしてからその値を変更します。

```
EXTERNAL myvar
IF myvar==1 THEN
    myvar=myvar+1
END
```

関連キーワード

AUTO

GLOBAL

LOCAL

FCLOSE

指定した論理ユニットにおいて現在開いているファイルを閉じます。

構文

FCLOSE (logical_unit)

使用上の留意事項

その論理ユニットにおいてファイルを開いていない場合、IOSTAT 実数型関数がエラーコードを返してもエラーは発生しません。

パラメータ

logical_unit アクセスされるデバイスを指定する実数値、変数または式です。整数として解釈されます。論理ユニット番号に関する説明については、ATTACH コマンドを参照してください。

詳細

プログラムが FOPEN コマンドで開かれているファイルへのアクセスを終了したら、プログラムは FCLOSE コマンドを実行してそのファイルを閉じる必要があります。FCLOSE によりそのファイルは eV+ モニタプログラムやその他のプログラムによるアクセスから解放されます。さらに、書き込みのために開かれていたファイルについては、FCLOSE が、eV+ によって依然バッファされているデータを書き出して、ファイルディレクトリ情報を更新します。このオペレーションを実行しない場合、ディスクファイルに書き込んだ情報すべてが含まれていない可能性があります。

論理ユニットを割り付け解除した場合、FOPEN を発行したプログラムが実行を終了した場合、またはプログラムタスクの KILL が実行された場合、そのユニットで FCLOSE オペレーションが自動的に実行されます。

IOSTAT 実数型関数を使用してクローズオペレーションが問題なく完了したことを確認する必要があります。指定した論理ユニットで現在ファイルが開かれていないと、ファイルが開られなかったというエラーコードが返されます。

関連キーワード

ATTACH

DETACH

FOPEN

FOPENR

IOSTAT

KILL（モニタコマンド）

KILL （プログラムコマンド）

FCMND

論理ユニットが指定した入出力デバイスに対してそのデバイス特有のコマンドを作成します。

構文

FCMND (logical_unit, command_code) $out_string, $in_string

使用上の留意事項

参照される論理ユニットは事前に割り付けられている必要があります。

現在のデフォルトデバイス、ユニットおよびディレクトリパスは任意のディスクファイル仕様に適切と見なされます。詳細については DEFAULT モニタコマンドを参照してください。

パラメータ

logical_unit	アクセスするデバイスを定義する実数値式です。論理ユニット番号に関する説明については、ATTACH プログラムコマンドを参照してください。
command_code	実行するコマンドを定義する実数値式です。
$out_string	実行するオペレーションを定義するために、コマンドコードと一緒にデバイスに送られる文字列定数、変数、または式です。
$in_string	オプションの文字列変数です。この変数はコマンドの結果としてデバイスから返される情報を受けます。

詳細

FCMD プログラムコマンドにより、プログラムがデバイス特有のコマンドシーケンスを作成することが可能になります。例えば、このコマンドを使用して、ファイルを削除するため、またはファイル名を変更するためにディスクにコマンドを送ることができます。これらは通常は eV+ プログラムが実行しない維持オペレーションであるため、これらのオペレーションを実施するための特別目的の eV+ プログラムコマンドは存在しません。

このコマンドの指定におけるエラー（無効なユニットへアクセスを試みるなど）はプログラムエラーを生じ、実行プログラムを停止します。デバイスの未準備など、実際にオペレーションの実施に関連するエラーは、プログラムの通常の運用において発生するおそれのあるものであるため、実行プログラムを停止することはありません。これら通常のエラーは、FCMND を実行した後に IOSTAT 関数を使用して検出可能です。

> **注:** 一般的に、IOSTAT 関数を使用することでそれぞれの FCMND オペレーションが問題なく完了していることを常に試験するのはよい習慣です。

ファイルコマンドコード

FCMND コマンドを実施した後、論理ユニットでファイルを開くことができません。

6 ファイル名を変更します。$out_string パラメータには、必要なディスクユニットおよびディレクトリパス指定を含んだファイルの新しい名称が入っている必要があります。$in_string パラメータには旧ファイル名が入っている必要があります。

14 サブディレクトリを作成します。$out_string パラメータにはサブディレクトリの指定が含まれる必要があります。これには、現在のデフォルトディスクにアクセスしていない場合、オプションのユニット名が含まれます。サブディレクトリの仕様に関する説明については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

> **注:** 指定されたディレクトリパス内の最終的サブディレクトリだけがこのオペレーションで作成されます。すべての中間サブディレクトリが作成されるわけではありませんが、それらはすべて存在している必要があります。

15 サブディレクトリを削除します。$out_string パラメータにはサブディレクトリの指定が含まれる必要があります。これには、現在のデフォルトディスクにアクセスしていない場合、オプションのユニット名が含まれます。サブディレクトリの仕様に関する説明については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

> **注:** 指定されたディレクトリパス内の最終的サブディレクトリだけがこのオペレーションで作成されます。すべての中間サブディレクトリが作成されるわけではありませんが、それらはすべて存在している必要があります。

19 指定された論理ユニットで現在開かれているファイルに対して作成日時をアサートします。このファイルコマンドはディスクファイルが開かれていればいつでも発行できます。ファイルが閉じられているときに発行されると、ファイルの日時は現在の日時ではなく指定された値に等しく設定されます。ファイルが閉じられているときにこのコマンドが発行されると、eV+ はアーカイブされていないビットを自動的にはアサートしません。入力文字列には、以下の留意事項を守った date および time が入っている必要があります。

- date は 16 ビット整数ワードであり、日付を TIME および $TIME 関数で使用される標準圧縮フォーマットで示します。
- time は時刻を標準圧縮フォーマットで示す 16 ビットの整数ワードです。
- このファイルコマンドコードはローカルディスクドライブだけに適用されます。

20 ローカルディスクの未使用 KB 数および総 KB 数を返します。返される文字列は uuuuu/ttttt の形式になります。ここで uuuuu は未使用 KB 数、ttttt は総 KB 数です。ファイルは、先読みを無効にしたドライブで開く必要があります。開かれたファイルはディスクユニットを特定します。

21 指定された論理ユニットで現在開かれているファイルに対して作成日時を読み取ります。このファイルコマンドはディスクファイルを開いた後であればいつでも発行できます。通常このコマンドは、ファイルを開いた時点でディクスディレクトリから読み込んだ値を返します。しかしファイル作成日時をアサートするためにFCMND 19 ステートメントが発行されている場合、そのFCMND 21 ステートメントはFCMND 19 ステートメントが設定した値を返します。このコマンドが返す文字列には日時が含まれます(INTBを使用して値を抽出します)。以下の点について留意する必要があります。

- dateは16 ビット整数ワードであり、日付をTIME および$TIME 関数で使用される標準圧縮フォーマットで示します。
- timeは時刻を標準圧縮フォーマットで示す16 ビットの整数ワードです。
- このファイルコマンドコードはローカルディスクドライブだけに適用されます。

TCP コマンドコード

600 ステートメントFCMND (lun, 600) $INTB(handle) においてハンドル番号「handle」によって特定されるクライアントのTCP サーバ側からの閉鎖接続を開始します。

> **注:** 閉鎖接続リクエストはより一般的には、クライアント側から開始されます。

601 PING モニタコマンドを開始します。結果として返されるIOSTAT 関数値はネットワーク上でクライアントを発見したこと示す1 か、ネットワークのタイムアウトを示す-562 のどちらかになります。

例

以下の例では、クライアントがネットワーク上にあるか確認します。

```
FCMND (lun, 601) "node_address", $str
```

関連キーワード

ATTACH

DETACH

FDELETE（モニタコマンド）

FDELETE（プログラムコマンド）

FDIRECTORY

FOPEN

FRENAME

IOSTAT

MCS

FCOPY

既存のディスクファイルの情報を新しいディスクファイルにコピーします。

構文

FCOPY err, **$new_file = $old_file**

パラメータ

err	エラーを返すために使用するオプションのパラメータです。
$new_file	新たに作成するディスクファイルのファイルを指定する文字列定数、変数または式です。ピリオド（.）およびファイル名の拡張子を省略すると、デフォルトは空の式になります。必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。詳細については DEFAULT コマンドを参照してください。
$old_file	既存のディスクファイルを指定する文字列定数、変数または式です。ピリオド(.)およびファイル名の拡張子を省略すると、デフォルトは空の式になります。必要に応じて、現在のデフォルトのデバイス、ユニット、ディレクトリパスが適用されます。

詳細

新しいファイルが既に存在している、または古いファイルが存在していない場合、エラーが報告され、コピーは実行されません。既存のファイルを上書きすることはできません。既存のファイルをまず FDELETE コマンドを使用して消去する必要があります。

コピーする予定のファイルが読み取り専用属性の場合、新しいファイルも同じ属性になります。保護属性のあるファイルをコピーすることはできません。ファイル保護属性に関する説明については FDIRECTORY を参照してください。

ファイルをコピーすると、ファイルの作成日が標準ファイル属性と共に保存されます。ファイルがアーカイブされていないことを示すために、アーカイブビットだけがクリアされます。

通常、ファイル指定は、次の 6 つの要素から構成されます。

1. オプションの物理デバイス（例、DISK>）
2. オプションのディスクユニット（例、D:）
3. オプションのディレクトリパス（例、DEMO¥）
4. ファイル名（例、NEWFILE）
5. ピリオド文字（「.」）
6. ファイルの拡張子（例、V2）

例

次の例では、ディスクデバイス "D" に "newfile.v2" という名称のファイルを作成します。このファイルはディスクデバイス "D" にある "oldfile.v2" という名称の既存ファイルの完全なコピーです。

```
FCOPY "D:\newfile.v2" = "D:\oldfile.v2"
```

関連キーワード

DEFAULT

FCOPY（モニタコマンド）

FRENAME

FDELETE

指定したディスクファイルを削除します。

構文

FDELETE (lun) object

使用上の留意事項

論理ユニット番号をアタッチする必要がありますが、その論理ユニットで現在オープンされているファイルがあってはいけません。

FCLOSE開いているディスクファイルをこのキーワードを使用して消去する前に発行する必要があります。

パラメータ

lun	ディスクに対応する、整数と解釈される実数値、変数または式です。論理ユニット番号に関する説明については、ATTACH コマンドを参照してください。
オブジェクト	削除するディスクを指定する文字列定数、変数または式です。指定したオブジェクトが存在しない場合、IOSTAT でエラー "Nonexistent file"が報告されます。 この文字列にはオプションのディスクユニットおよびオプションのディレクトリパスを含めることが可能であり、ファイル名、ピリオド（.）およびファイル拡張子が含まれている必要があります。現在のデフォルトのディスクユニットおよびディレクトリパスは適正なものとみなされます。詳細については DEFAULT コマンドを参照してください。

詳細

ディスク論理ユニット番号を指定すると、オプジェクトパラメータは削除するディスクファイルの指定であると解釈されます。何らかの理由（例えば、ファイルが存在しないまたはディスクが保護されている）で削除ができなかった場合、IOSTAT 実数値関数を介してエラーが返されます。

例

次の例では、文字列変数 $file のファイル指定によって定義されるディスクを閉じてから消去します。

```
FCLOSE (5) $file

FDELETE (5) $file
```

関連キーワード

ATTACH

FCLOSE

FDELETE（モニタコマンド）

FOPEN

IOSTAT

FEMPTY

ディスクに対して使用する内部バッファを、必要に応じてそのバッファをファイルに書き込むことで、空にします。

構文

FEMPTY (logical_unit)

使用上の留意事項

ファイルにアクセスする場合、指定された論理ユニットでのランダムアクセスのためにファイルを開く必要があります（詳細については FOPEN_ コマンド参照）。

パラメータ

lun	アクセスされるデバイスを指定する実数値、変数または式(整数と解釈される)です。論理ユニット番号に関する説明については、ATTACH コマンドを参照してください。

詳細

ディスクファイルの I/O にランダムアクセスするとき、eV+ はデータをディスクに 512 バイト（文字）のブロックで書き込みます。効率化のため、長さ512 バイト未満のレコードをWRITE コマンドを使用して書き込んだ場合、データを内部バッファに保存し、ディスクには実際には書き込まずに後で書き込むようにします。

ディスク論理ユニットが参照されている場合、FEMPTY コマンドは、eV+ に対して内部バッファの内容を速やかにディスクファイルに書き込むように命じます。これは、データ保全性が特に重要なアプリケーションで有用です。

IOSTAT 実数値関数を使用して、FEMPTY オペレーションによって生じたエラーを判断することができます。

例

次の例では、論理ユニット 5 の内部出力バッファを空にして、ディスクに速やかに書き込みます。

```
FEMPTY (5)
```

関連キーワード

ATTACH

FOPEN

FOPENA

FOPEND

FOPENR

FOPENW

IOSTAT

WRITE

FINE

ロボットに対して高精度ヌル許容誤差を有効にします。

構文

FINE tolerance *ALWAYS*

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボットモーションしか影響を受けません。

tolerance パラメータが指定されていると、ALWAYS 指定の有無に関わらず、その値は現在の実行サイクルで実行されるその後の FINE プログラム命令に対するデフォルトになります。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、ステートメント FINE 100 ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

FINE プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによってアタッチされていない限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、FINE コマンドを実行するとエラーになります。

パラメータ

tolerance	現在の実行タスクにアタッチされているロボットの各ジョイントに使用される標準的高精度許容誤差のパーセンテージを指定するオプションの実数値、変数または式です。
ALWAYS	デフォルト条件として FINE を確立するオプションの修飾子です。COARSE プログラムコマンドによって無効になるまで、FINE は有効のままになります。ALWAYS を指定しない場合、FINE コマンドは次のロボットモーションだけに適用されます。

詳細

FINE プログラムコマンドはロボットモーションサーボ系統において高精度機能を可能にして、モーション終了時にロボットジョイントの最終ポジションで小さな誤差を許容するようにします。これにより、高精度な動作を生み出しますが、各モーションの最後での設定時間が増加することで、サイクル時間は増加します。

tolerance パラメータを指定する場合、新しい設定は次のモーションの開始時に有効になります。ALWAYS 指定の有無に関わらずこの値は現在の実行サイクルで実行されるその後の FINE コマンドに対するデフォルトになります。FINE 許容誤差を変更しても COARSE 許容誤差に影響はありません。

tolerance パラメータを省略した場合、アタッチされているロボットの最新の設定が使用されます。プログラムの実行が始まるか、新しい実行サイクルが開始されると、デフォルト設定は 100% に戻り、ロボットがプログラムにアタッチされていると仮定します。

例

次の例では、次のモーションだけに高精度機能を有効にしています。

```
FINE
```

次の例では、次のモーションに高精度機能を有効にし、各ジョイントに対して、許容誤差設定を標準許容誤差の 50% に変更しています。

```
FINE 50
```

次の例では、明確に無効になるまで高精度機能が有効です。

```
FINE ALWAYS
```

関連キーワード

COARSE

CONFIG

DELAY.IN.TOL

NONULL

NULL

SELECT（プログラムコマンド）

SELECT（実数値関数）

FLIP

次の動作に関するロボットの設定変更を要求して、ロボットの手首のピッチ角がマイナスの値になるようにします。

構文

FLIP

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがFLIP 設定をサポートしていない場合、ロボットはこの命令を

無視します。

FLIP プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。ロボットが割り付けられていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、FLIP コマンドを実行するとエラーになります。

詳細

FLIP 設定を有効にすると、手首関節がマイナス方向へ回転します。

NOFLIP 設定を有効にすると手首関節がプラス方向へ回転します。

手首関節の角度は ±180° で表現されます。

> **注:** ロボットは関節補間動作中に限って設定を変更できます。

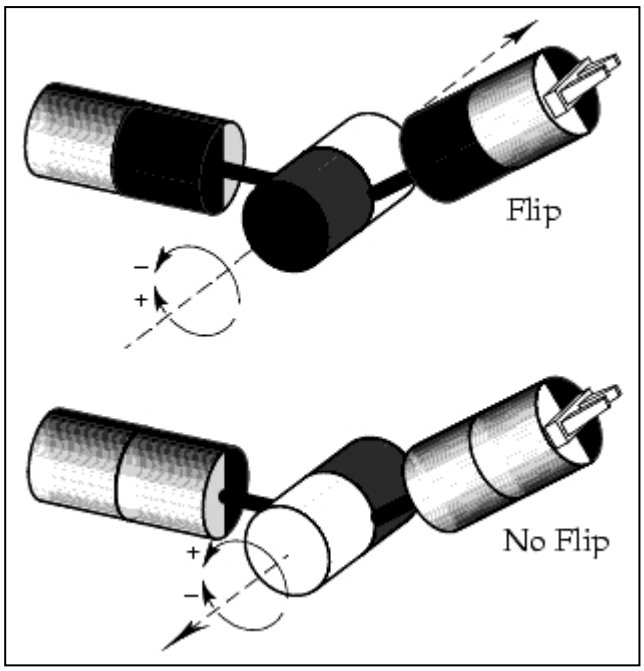


図3-7. FLIP / NOFLIP 設定

以下の図に、6 軸ロボットのFLIP 設定とNOFLIP 設定を示します。

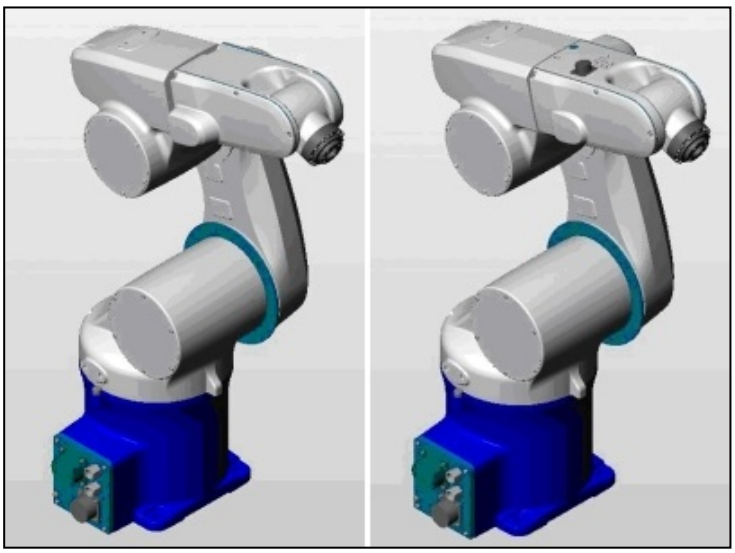


図3-8. Viper 650 ロボットの FLIP/NOFLIP の例

例

以下の例では、FLIP および NOFLIP コマンドの使用方法を示します。

```
FLIP
MOVE loc_a

NOFLIP
```

```
MOVE loc_a
```

関連キーワード

CONFIG

NOFLIP

SELECT（プログラムコマンド）

SELECT（実数値関数）

FOPEN

新しい TCP 接続を作成して開きます。

構文

FOPEN (lun, mode**) attribute**

使用上の留意事項

開く操作を成功させるには、論理ユニットを割り付けておく必要があります。

パラメータ

lun	TCP デバイスに割り当てた論理ユニット番号を定義する、実数値、変数または式(整数値と解釈される)です。ユニット番号に関する説明については、ATTACH コマンドを参照してください。
mode	TCP 接続のタイプです。0 = クライアントモード、16 = サーバモード。
attribute	サーバモードで TCP 接続を開いた場合、この文字列はサーバの特性を定義します。クライアントモードで接続を開いた場合、この文字列は、接続の特性に加えて、サーバ名を定義します。 属性リストは TYPE コマンドの出力仕様と同じように処理され、TCP ドライバに引き渡される 1 つの文字列を構成するために使用されます。この文字列は 512 文字を超えてはいけません。 **注:** eV+ 文字列リテラルまたは文字列変数は 128 文字を超えることができません。128 文字を超える属性リストを作成するためには、複数の文字列を廉価つする必要があります。 属性リストは、カンマで区切られた 1 つ以上の構成要素で構成することができます。個々の構成要素は以下のいずれかの方法で表現することができます。 • 文字列定数、変数または式。 • 制御文字列内で使用される値を決定するために判定される実数型の定数、変数または式。 • 制御文字列の情報の様式を決定するフォーマット制御指定子。

詳細

TCP/IP 接続はサーバモードまたはクライアントモードで開くことができます。サーバモードの場合、（/CLIENTS に割り当てられる値に応じて）1 つ以上のクライアントをその後の通信のためにサーバに接続できます。

クライアント-サーバ間の接続を確立するためには、クライアントがサーバのポート番号を知っている必要があります。そのため、サーバ接続を開くために FOPEN 命令を使用する場合、/LOCAL_PORT 属性を使用してポートを明確に定義します。クライアントが使用するポート番号をサーバが知っている必要はありません。

標準の TCP アプリケーションパッケージが使用するポート番号は 0～255 です。独自のカスタムプロトコルを書き込んでいる場合、255 よりも大きいポート番号を使用します。

FOPEN 命令に対する有効な TCP 属性を以下の表にしまします。

表3-6. FOPEN TCP 属性

属性:	/CLIENTS
説明:	そのサーバで許容されるクライアント接続の数を定義します。省略すると、1 つのクライアント接続を仮定します。クライアント接続の最大数は 31 です。
属性:	/LOCAL_PORT
説明:	その接続のローカルポート番号を定義します。省略すると、ローカルボート番号が自動的に割り当てられます。
属性:	/REMOTE_PORT
説明:	クライアント接続を確立するサーバのポート番号を定義します。これはクライアント接続を確立する際に与えられる必要があります。

例

以下の例では、ローカルポート 260 が 5 つのクライアント接続を受けるように TCP サーバを構成します。

```
FOPEN (lun, 16) "/LOCAL_PORT 260 /CLIENTS 5"
```

以下の例では、"server1" という名称のサーバのポート番号 260 に接続するように TCP クライアント接続を構成します。

```
FOPEN (lun, 0) "server1 /REMOTE_PORT 260"
```

関連キーワード

ATTACH

DETACH

FCLOSE

FDELETE

FEMPTY

FSET

IOSTAT

FOPENA

読み書き追加アクセスのためにファイルを開きます。指定したファイルがすでに存在しない場合、ファイルが作成されます。

構文

FOPENA (lun, record_len, mode**) file_spec**

使用上の留意事項

ファイルが既に存在する場合、エラーは発生せず、ファイルの位置はファイルの最後に設定されます。書き込み操作をしてから既存のファイルに追加を行います。

開く操作を成功させるには、論理ユニットを割り付けておく必要があります。

システム全体でいつでも、最大 60 個のディスクファイルおよび 160 個のネットワークファイルを開くことができます。これにはすべてのプログラムタスクおよびシステムモニタ（例えば FCOPY コマンド）によって開かれたファイルが含まれます。

パラメータ

lun

アクセスするディスクデバイスの論理ユニット数を定義する実数値式です。

ユニット番号に関する説明については、ATTACH コマンドを参照してください。

record_len

読み書きされるレコードの長さを定義する、オプションの実数値式です。

レコード長を省略した場合および 0 の場合、可変長レコードが処理されます。この場合、レコードのランダムアクセスは実行できません。

レコード長が 0 以外の場合、処理される固定長レコードの長さ（文字数）を指定します。固定長レコードの場合、ランダムアクセスが許されます。

mode

読み取りアクセスの実行方法を定義する、オプションの実数値式です。指定される値は以下に示すように連続するビットフラグと解釈されます。モード値が指定されていない場合、すべてのビットはクリアであると仮定されます。

ビット 1（最下位ビット）：先行読み取りを無効にします（マスク値 = 1）

このビットが OFF の場合、eV+ はファイルを開いたとき（先行読み取りしたとき）、およびその後 READ リクエストが後続することが予測されるそれぞれの READ コマンドの後で、レコードを読み取ります。このビットが ON の場合、こういった先行読み取りは実行されません。

ビット 2: ランダムアクセスを有効にします（マスク値 = 2）

このビットが OFF の場合、順番にファイルにアクセスします。レコードはファイル内での作成順に読み取りまたは書き込みされます。

このビットが ON の場合、ランダムアクセスを使用してファイルにアクセスしますが、これは固定長レコードを持つディスクファイルしか許されません。ランダムアクセスモードでは、READ または WRITE コマンド内のレコード番号パラメータによってアクセスするレコードが指定されます。

ビット 4: 強制ディスク書き込み（マスク値 = 8）

ON の場合、書き込みアクセスのために開かれるディスクファイルにレコードが書き込まれる都度、物理ディスクに書き込みが行われます。書き込みの都度、ディレクトリまたはファイル割り当て情報が更新されます。このモードは書き込み後にファイルを閉じる際と同等（かそれよりも高速）です。これは通常のバッファされたモードよりも非常に低速ですが、書き込まれた情報がシステムクラッシュや停電で失われることがないことを保証します。

このモードは原則的に、長時間にわたって開いたままであり断続的に更新されるログファイルと共に使用されることを目的としています。これらのファイルタイプに対して、このモードの著しい追加負荷はその利点ほど重要ではありません。

file_spec

開かれるファイルを指定する文字列定数、変数または式です。この文字列にはオプションのディスクユニットおよびオプションのディレクトリパスを含めることが可能であり、ファイル名、ピリオド（.）およびファイル拡張子が含まれている必要があります。

現在のデフォルトディスクユニットおよびディレクトリパスは適正なものとみなされます。

ディスクユニットおよびディレクトリパスに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

詳細

このコマンドはディスクファイルを開いて入出力オペレーションを実行できるようにします。オペレーションが完了したら、FCLOSE または DETACH コマンドを使用してファイルを閉じる必要があります。

関連キーワード

ATTACH

DETACH

FCLOSE

FOPEN

FOPEND

FOPENR
FOPENW
FSEEK
IOSTAT
READ
WRITE

FOPEND

読み取りのためにディスクディレクトリを開きます。

構文

FOPEND (lun, record_len, mode) **file_spec**

使用上の留意事項

file_spec パラメータのファイル名および拡張子は、読み取り作業を後で実行する場合に使用するファイル名テンプレートを準備するために使用されます。これらの読み取り作業では、ディスクディレクトリファイルからファイル名テンプレートに適合するレコードだけを返します。ディレクトリファイルへ書き込もうとするとエラーになります。ディレクトリレコードのフォーマットに関する情報については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

ファイル名および拡張子にはワイルドカード（アスタリスク、*）を含めることができます。ファイル名または拡張子内のワイルドカードは、その位置に任意の文字を受け入れることができることを示します。ファイル名または拡張子の末尾にワイルドカードがある場合、任意の後続文字を受け入れることができることを示します。ファイル名（または拡張子）の代わりにワイルドカードがある場合、任意の名称（または拡張子）を受け入れることができることを示します。ファイル名、ピリオド、およびファイル拡張子を省略することは *.* と指定したのと同じことになります。ピリオドおよびファイル拡張子を省略すると、ワイルドカード拡張子を指定したのと同じことになります。

開く操作を成功させるには、論理ユニットを割り付けておく必要があります。

システム全体でいつでも、最大 60 個のディスクファイルおよび 160 個のネットワークファイルを開くことができます。これにはすべてのプログラムタスクおよびシステムモニタ（例えば FCOPY コマンド）によって開かれたファイルが含まれます。

パラメータ

lun	アクセスするディスクデバイスの論理ユニット数を定義する実数値式です。 ユニット番号に関する説明については、ATTACH コマンドを参照してください。
record_len	読み取りされるレコードの長さを定義する、オプションの実数値式です。 レコード長を省略した場合および 0 の場合、可変長レコードが処理されます。この場合、レコードのランダムアクセスは実行できません。 レコード長が 0 以外の場合、処理される固定長レコードの長さ（文字数）を指定します。固定長レコードの場合、ランダムアクセスが許されます。
mode	読み取りアクセスの実行方法を定義する、オプションの実数値式です。指定される値は以下に示すように連続するビットフラグと解釈されます。モード値が指定されていない場合、

すべてのビットはクリアであると仮定されます。

ビット 1（最下位ビット）: 先行読み取りを無効にします（マスク値 = 1）

このビットが OFF の場合、eV+ はファイルを開いたとき（先行読み取りしたとき）、およびその後 READ リクエストが後続することが予測されるそれぞれの READ コマンドの後で、レコードを読み取ります。このビットが ON の場合、こういった先行読み取りは実行されません。

ビット 2: ランダムアクセスを有効にします（マスク値 = 2）

このビットが OFF の場合、順番にファイルにアクセスします。レコードはファイル内での作成順に読み取りされます。

このビットが ON の場合、ランダムアクセスを使用してファイルにアクセスしますが、これは固定長レコードを持つディスクファイルしか許されません。ランダムアクセスモードでは、READ コマンド内のレコード番号パラメータによってアクセスするレコードが指定されます。

ビット 4: 強制ディスク書き込み（マスク値 = 8）

ON の場合、書き込みアクセスのために開かれるディスクファイルにレコードが書き込まれる都度、物理ディスクに書き込みが行われます。書き込みの都度、ディレクトリまたはファイル割り当て情報が更新されます。このモードは書き込み後にファイルを閉じる際と同等（かそれよりも高速）です。これは通常のバッファされたモードよりも非常に低速ですが、書き込まれた情報がシステムクラッシュや停電で失われることがないことを保証します。

このモードは原則的に、長時間にわたって開いたままであり断続的に更新されるログファイルと共に使用されることを目的としています。これらのファイルタイプに対して、このモードの著しい追加負荷はその利点ほど重要ではありません。

file_spec

開かれるファイルを指定する文字列定数、変数または式です。この文字列にはオプションのディスクユニットおよびオプションのディレクトリパスを含めることが可能であり、ファイル名、ピリオド（.）およびファイル拡張子が含まれている必要があります。ファイル名および拡張子はオプションであり、両方にワイルドカードを含めることができます（上記の「使用上の留意事項」を参照してください）。

現在のデフォルトディスクユニットおよびディレクトリパスは適正なものとみなされます。

ディスクユニットおよびディレクトリパスに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

詳細

このコマンドはディスクファイルを開いて入出力オペレーションを実行できるようにします。オペレーションが完了したら、FCLOSE または DETACH コマンドを使用してファイルを閉じる必要があります。

例

以下の例では、現在のデフォルトディレクトリを開いて拡張子 DAT を持つすべてのファイルを検索します。

```
FOPEND (5) "*.dat"
```

関連キーワード

ATTACH

DETACH

FCLOSE

FOPEN

FOPENA

FOPENR

FOPENW

FSEEK

IOSTAT

READ

FOPENR

読み取りアクセスのためにファイルを開きます。

構文

FOPENR (lun, record_len, mode**) file_spec**

使用上の留意事項

開く操作を成功させるには、論理ユニットを割り付けておく必要があります。

システム全体でいつでも、最大 60 個のディスクファイルおよび 160 個のネットワークファイルを開くことができます。これにはすべてのプログラムタスクおよびシステムモニタ（例えば FCOPY コマンド）によって開かれたファイルが含まれます。

パラメータ

lun

アクセスするディスクデバイスの論理ユニット数を定義する実数値式です。

ユニット番号に関する説明については、ATTACH コマンドを参照してください。

record_len

読み取りされるレコードの長さを定義する、オプションの実数値式です。

レコード長を省略した場合および 0 の場合、可変長レコードが処理されます。この場合、レコードのランダムアクセスは実行できません。

レコード長が 0 以外の場合、処理される固定長レコードの長さ（文字数）を指定します。固定長レコードの場合、ランダムアクセスが許されます。

mode

読み取りアクセスの実行方法を定義する、オプションの実数値式です。指定される値は以下に示すように連続するビットフラグと解釈されます。モード値が指定されていない場合、すべてのビットはクリアであると仮定されます。

ビット 1（最下位ビット）: 先行読み取りを無効にします（マスク値 = 1）

このビットが OFF の場合、eV+ はファイルを開いたとき（先行読み取りしたとき）、およびその後 READ リクエストが後続することが予測されるそれぞれの READ コマンドの後で、レコードを読み取ります。このビットが ON の場合、こういった先行読み取りは実行されません。

ビット 2: ランダムアクセスの有効にします（マスク値 = 2）

このビットが OFF の場合、順番にファイルにアクセスします。レコードはファイル内での作成順に読み取りされます。

このビットが ON の場合、ランダムアクセスを使用して

ファイルにアクセスしますが、これは固定長レコードを持つディスクファイルしか許されません。ランダムアクセスモードでは、READ コマンド内のレコード番号パラメータによってアクセスするレコードが指定されます。

ビット 4: 強制ディスク書き込み（マスク値 = 8）

ON の場合、書き込みアクセスのために開かれるディスクファイルにレコードが書き込まれる都度、物理ディスクに書き込みが行われます。書き込みの都度、ディレクトリまたはファイル割り当て情報が更新されます。このモードは書き込み後にファイルを閉じる際と同等（かそれよりも高速）です。これは通常のバッファされたモードよりも非常に低速ですが、書き込まれた情報がシステムクラッシュや停電で失われることがないことを保証します。

このモードは原則的に、長時間にわたって開いたままであり断続的に更新されるログファイルと共に使用されることを目的としています。これらのファイルタイプに対して、このモードの著しい追加負荷はその利点ほど重要ではありません。

file_spec

開かれるファイルを指定する文字列定数、変数または式です。この文字列にはオプションのディスクユニットおよびオプションのディレクトリパスを含めることが可能であり、ファイル名、ピリオド（.）およびファイル拡張子が含まれている必要があります。

現在のデフォルトディスクユニットおよびディレクトリパスは適正なものとみなされます。

ディスクユニットおよびディレクトリパスに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

詳細

このコマンドはディスクファイルを開いて入出力オペレーションを実行できるようにします。オペレーションが完了したら、FCLOSE または DETACH コマンドを使用してファイルを閉じる必要があります。

例

以下の例では、可変長レコード（レコード長は省略）への読み取り専用アクセスのためのデフォルトデバイスで data.dat という名称のファイルを開きます。モードパラメータを省略しているので、先行読み取りが行われ、レコードは順番にアクセスされます（可変長レコードに必要）。

```
FOPENR (5) "data.dat"
```

関連キーワード

ATTACH

DETACH

FCLOSE
FOPEN
FOPENA
FOPEND
FOPENW
FSEEK
IOSTAT
READ

FOPENW

読み書きアクセスのためにファイルを開きます。ファイルが既に存在していると、エラーが発生します。

構文

FOPENW (lun, record_len, mode**) file_spec**

使用上の留意事項

この命令の指定におけるエラー（無効なユニットへアクセスを試みるなど）はプログラムエラーを生じ、実行プログラムを停止します。デバイスの未準備など、実際にオペレーションの実施に関連するエラーは、プログラムの通常の運用において発生するおそれのあるものであるため、実行プログラムを停止することはありません。これら通常のエラーは、IOSTAT 関数を使用して検出可能です。

開く操作を成功させるには、論理ユニットを割り付けておく必要があります。

システム全体でいつでも、最大 60 個のディスクファイルおよび 160 個のネットワークファイルを開くことができます。これにはすべてのプログラムタスクおよびシステムモニタ（例えば FCOPY コマンド）によって開かれたファイルが含まれます。

パラメータ

lun

アクセスするディスクデバイスの論理ユニット数を定義する実数値式です。

ユニット番号に関する説明については、ATTACH コマンドを参照してください。

record_len

読み書きされるレコードの長さを定義する、オプションの実数値式です。

レコード長を省略した場合および 0 の場合、可変長レコードが処理されます。この場合、レコードのランダムアクセスは実行できません。

レコード長が 0 以外の場合、処理される固定長レコードの長さ（文字数）を指定します。固定長レコードの場合、ランダムアクセスが許されます。

mode

読み取りアクセスの実行方法を定義する、オプションの実数値式です。指定される値は以下に示すように連続するビットフラグと解釈されます。モード値が指定されていない場合、すべてのビットはクリアであると仮定されます。

ビット 1（最下位ビット）：先行読み取りを無効にします（マスク値 = 1）

このビットが OFF の場合、eV+ はファイルを開いたとき（先行読み取りしたとき）、およびその後 READ リクエストが後続することが予測されるそれぞれの READ コマンドの後で、レコードを読み取ります。このビットが ON の場合、こういった先行読み取りは実行されません。

ビット 2: ランダムアクセスを有効にします（マスク値 = 2）

このビットが OFF の場合、順番にファイルにアクセスします。レコードはファイル内での作成順に読み取りまたは書き込みされます。

このビットが ON の場合、ランダムアクセスを使用してファイルにアクセスしますが、これは固定長レコードを持つディスクファイルしか許されません。ランダムアクセスモードでは、READ または WRITE コマンド内のレコード番号パラメータによってアクセスするレコードが指定されます。

ビット 4: 強制ディスク書き込み（マスク値 = 8）

ON の場合、書き込みアクセスのために開かれるディスクファイルにレコードが書き込まれる都度、物理ディスクに書き込みが行われます。書き込みの都度、ディレクトリまたはファイル割り当て情報が更新されます。このモードは書き込み後にファイルを閉じる際と同等（かそれよりも高速）です。これは通常のバッファされたモードよりも非常に低速ですが、書き込まれた情報がシステムクラッシュや停電で失われることがないことを保証します。

このモードは原則的に、長時間にわたって開いたままであり断続的に更新されるログファイルと共に使用されることを目的としています。これらのファイルタイプに対して、このモードの著しい追加負荷はその利点ほど重要ではありません。

file_spec

開かれるファイルを指定する文字列定数、変数または式です。この文字列にはオプションのディスクユニットおよびオプションのディレクトリパスを含めることが可能であり、ファイル名、ピリオド（.）およびファイル拡張子が含まれている必要があります。

現在のデフォルトディスクユニットおよびディレクトリパスは適正なものとみなされます。

ディスクユニットおよびディレクトリパスに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

詳細

このコマンドはディスクファイルを開いて入出力オペレーションを実行できるようにします。オペレーションが完了したら、FCLOSE または DETACH コマンドを使用してファイルを閉じる必要があります。

例

以下の例では、それぞれ 32 文字の固定長レコードを使用した読み書きアクセスのために、デバイス D で x.d という名称のファイルを開きます。モード値 3 はビット 1 および 2 を両方設定するので、先行読み取りは無効になり、ランダムアクセス手法が使用されます。

```
FOPENW (5, 32, 3) "D:x.d"
```

関連キーワード

ATTACH

DETACH

FCLOSE

FOPEN

FOPENA

FOPEND

FOPENR

FSEEK

IOSTAT

READ

WRITE

FOR

プログラムループを指定した回数実行します。

構文

FOR loop_var = initial TO final STEP increment

使用上の留意事項

すべての FOR コマンドには関連する END コマンドが存在している必要があります。

外部変数は、loop_var パラメータに使用できません。

パラメータ

loop_var 実数型変数であり、FOR コマンドを実行すると初期化されます。ループが実行される都度インクリメントします。値または式を指定することはできません。

> **追加情報:** 一般的には AUTO 変数として宣言されます。
>
> **注:** 外部変数は、loop_var パラメータに使用できません。

initial ループを最初に実行する際に、loop_var の値を決定する実数値です。

final ループを終了すべき時期を判断するために loop_var と比較される値を決める実数値です。

increment ループを実行する都度 loop_var に加算される値を決めるオプションの実数値です。省略した場合、increment のデフォルトは 1 となり、STEP も省略されます。

詳細

FOR ステートメントとこれに対応する END ステートメントの間にあるコマンドを繰り返し実行し、この際 loop_var は increment として指定された値だけ毎回変化します。

この構造体の処理は以下に示すようになります。

1. 最初に FOR ステートメントが入力されると、loop_var を初期値に設定します。
2. increment および final パラメータの値を決めます。
3. 以下の条件で、final の値を loop_var の値と比較します。

- increment がプラスで、loop_var が final より大きい場合、以下の項目7 までスキップします。
- increment がマイナスで、loop_var が final より小さい場合、以下の項目7 までスキップします。

4. FOR ステートメントに後続するキーワードを実行します。
5. END ステップに到達した場合、increment の値を loop_var に加えます。
6. 上記の項目 3 に戻ります。
7. END の後の最初のキーワードでプログラムの実行を継続します。loop_var は、そのループに含まれるロジックで意図的に増減されているのでない限り（または他の場所で AUTO 変数として宣言されていないのでない限り）、上記 項目 3 における判定の時点の値を保持します。

項目 3 における判定が最初に不合格であった場合、FOR 構造体のキーワードグループは実行されない可能性があります。

FOR ステートメントが最初に実行されると、initial、increment および final の値によって、FOR ループ内でキーワードグループを実行する回数が決定します。FOR ループ内でこれらのパラメータの値を変更しても、FOR 構造体の処理に影響はありません。

ループ内で loop_var を変更するとループのオペレーションに影響しますが、これは一般的プログラムでは通常発生しません。

> **注:** initial、final または increment が整数値でない場合、浮動小数点の計算における丸めによって、ループの実行回数が想定よりも多くなったり少なくなったりする可能性があります。

例

以下の例では、10 × 10 の配列のすべての要素を 0 に設定します。

```
FOR i = 1 TO 10
    FOR j = 1 TO 10
        array[i,j] = 0
    END
END
```

関連キーワード

DO

END

EXIT

NEXT

WHILE

FSEEK

ファイルをランダムアクセスのために開く位置にして、指定レコードで読み取り操作を開始します。

構文

FSEEK (logical_unit, record_number)

使用上の留意事項

指定された論理ユニットでのランダムアクセスのためにファイルを開く必要があります。詳細については FOPEN_ コマンドを参照してください)。

大部分のアプリケーションでは、効率のために、先行読み取りモード以外でファイルを開くべきです。

パラメータ

logical_unit	アクセスするデバイスを定義する実数式です。 ユニット番号に関する説明については、ATTACH コマンドを参照してください。
record_number	ランダムアクセスモードで開かれるファイル指向デバイスに対して読み取るレコードを指定するオプションの実数値式です。これを省略すると、最後の読み込みの次のレコードを仮定します。

詳細

ランダムアクセスのためにファイルを開くと、現在のデータ処理のためのディスクファイルアクセスに必要な時間をオーバーラップすることで、システム性能を向上させることができます。FSEEK コマンドを使用することで、READ コマンドを処理したすぐ後、データを処理する前に、アプリケーションプログラムがディスク検索および可能な読み取り操作を開始することができます。

このコマンドの指定におけるエラー（無効なユニットの参照など）はプログラムエラーを生じ、実行プログラムを停止します。実際の検索オペレーションの実施に関連するエラー（ファイルの末尾やデバイス準備未完了など）は、プログラムの通常の運用において発生するおそれのあるものであるため、実行プログラムを停止することはありません。これら通常のエラーは、その後の READ オペレーションを実行した後に IOSTAT 関数を使用して検出可能です。一般的に、IOSTAT からの値を試験することで、それぞれのファイルオペレーションが問題なく完了していることを常に試験するのはよい習慣です。

例

次の例では、論理ユニット 5 で開かれているファイルのレコード番号 130 を検索します。

```
FSEEK (5, 130)
```

関連キーワード

ATTACH

FOPENA

FOPEND

FOPENR

FOPENW

IOSTAT

READ

FSET

ネットワークデバイスの属性を設定または変更します。

構文

FSET (logical_unit) attribute_list

使用上の留意事項

それぞれの FSET コマンドの後で IOSTAT 関数を使用して FSET オペレーションが成功したことを確認してください。

パラメータ

logical_unit ネットワークデバイスの論理ユニット番号を定義する、整数と解釈される、実数値、変数または式です。ユニット番号に関する説明については、ATTACH プログラムコマンドを参照してください。

attribute_list ネットワークデバイスの特性を定義するために使用される文字列定数、式、実数値、変数およびフォーマット指定子のリストです。このパラメータに関する詳細については FOPEN プログラムコマンドの説明を参照してください。

詳細

TCP（伝送制御プロトコル）ネットワークデバイスを FSET コマンドを使用して参照することができます。

これらのデバイスに FSET コマンドを使用してアクセスする場合、以下の表の属性を使用することができます。

表3-8. ネットワークのための FSET 属性

属性	説明
/ADDRESS	IP アドレス（TCP デバイスにのみ適用）
/NODE	ノード名。

TCP デバイスに割り付けられている論理ユニットにアクセスするために、FSET コマンドを使用してネットワークに新しいノードを定義することができます。FSET コマンドと共に使用される文字列は、eV+ 構成ファイルにおいて NODE ステートメントを使用したものと同じフォーマットを持ちます。

例

次の例では、「SERVER2」という名前の新しいノードを IP アドレス 172.16.200.102 で定義します。

```
ATTACH (lun, 4) "TCP"
FSET (lun) "/NODE 'SERVER2' /ADDRESS '172.16.200.102'"
```

関連キーワード

FOPEN

IOSTAT

GLOBAL

変数をグローバルとして宣言し、その変数のタイプを指定します。

構文

GLOBAL type **variable,** ..., variable

使用上の留意事項

GLOBAL ステートメントはプログラム内の実行可能なステートメントの前に必ず現れなければなりません。

パラメータ

type

type パラメータを指定すると、すべての変数がそのタイプ（REAL、DOUBLE または LOC）に一致する必要があります。配列変数には、インデックスを明確に指定してその配列の最高次の有効なインデックスを示す必要があります。

このキーワードを省略すると、それぞれの変数のタイプはプログラム内での使用によって決まります。使用によってタイプを決定できない場合、エラーが発生します。

LOC　ロケーション変数 (変換またはプレシジョンポイント) です。

REAL　単精度実数型変数です。

DOUBLE　倍精度実数型変数です。

デフォルトタイプに関する詳細については GLOBAL コマンドを参照してください。

variable

変数名 (ベルト、プレシジョンポイント、実数、文字列、変換) 。それぞれの変数は単純な変数または配列である場合があります。type パラメータを指定すると、すべての変数がその指定したタイプに一致する必要があります。配列変数にはインデックスを指定しないでください。

詳細

AUTO または LOCAL として宣言されていない変数は、デフォルトで GLOBAL となります。倍精度およびロケーショングローバルな変数の場合、宣言は不要です。

同じ名称の LOCAL または AUTO 変数を宣言していない任意のプログラムによってグローバル変数にアクセスできます。例えば、program_a が var1 を GLOBAL 変数であると宣言し、program_b が var1 を AUTO 変数であると宣言した場合、program_b が GLOBAL な var1 を使用し変更することはできません。program_b を実行する都度、program_b 専用の変数 var1 の新しいコピーが作成されます。

例

以下の例では、2 つの文字列変数および 1 つのタイプ指定されていないグローバル変数が作成されます。

```
GLOBAL $str_1, $str_2, x
```

以下の例では、1 つのグローバル精密位置グローバル変数が作成されます。

```
GLOBAL LOC #ppoint_1
```

以下の例では、2 つの倍精度実数グローバル変数が作成されます。

```
GLOBAL var_1, var_2
```

関連キーワード

AUTO

LOCAL

EXTERNAL

GOTO

指定のラベルで特定されるプログラムステップに無条件分岐を実行します。

構文

GOTO label

パラメータ

label	その実行が分岐するプログラムステップのラベルです。ステップラベルは、0～65535 の範囲である整数値です。

詳細

このコマンドによって、実行されるプログラムは、指定されたステップラベルを含む行にジャンプします。ステップラベルは行番号とは違うものです。行番号とは、プロセス編集を支援するために eV+ プログラムエディタが自動的に割り当てる番号のことです。ステップラベルはプログラム行の当該部に明確に入力する必要があります。

> **重要:**現代の構造化プログラミングにおいて、GOTO ステートメントはよいプログラミング手法とはみなされません。OMRON は、GOTO ステートメントの代わりに他の制御構造体を使用することを推奨します。

例

以下の例では、1～100 の番号を入力することが要求されます。この番号が範囲内にない場合、GOTO 10 コマンドによって実行プログラムは ステップラベル 10 へジャンプします。

```
10 PROMPT "Enter a number from 1 to 100: ", number
IF (number < 1) OR (number > 100) THEN
        TYPE /B, /C1, "*Invalid response*", /C1
GOTO 10
END
```

関連キーワード

DO

EXIT

FOR

IF ...THEN

IF ... GOTO

NEXT

WHILE

HALT

実行プログラムを停止して再開できないようにします。

構文

HALT

使用上の留意事項

HALT プログラムコマンドを使用してプログラムを停止した場合、PROCEEDモニタコマンドを使用して実行プログラムを再開することはできません。

HALT はディスク論理ユニット上で必要に応じて FCLOSE および／または DETACH オペレーションを強制します。

詳細

HALT プログラムコマンドにより中断が発生し、その後、未実施のプログラムループがあるかに関わらず、アプリケーションプログラムの実行を終了します。メッセージ（HALTED）が表示されます。

HALT コマンドを使用して終了すると、PROCEED または RETRY コマンドで実行プログラムを再開することはできません。

例

次の例では、ロボットが「safe_loc」として定義されたロケーションに到達した後、プログラムタスクの実行を停止します。

```
MOVE safe_loc
BREAK
HALT
```

関連キーワード

PAUSE

RETURN

STOP

HERE

現在のロボットのロケーションに相当する変換またはプレシジョンポイント変数の値を設定します。

構文

HERE location_var

使用上の留意事項

HERE プログラムコマンドは、このコマンドを実行するタスクが選択したロボットに関する情報を返します。

eV+ システムがロボットを制御するように設定されていない場合、HERE コマンドを実行するとエラーになります。

パラメータ

location_var	変換、プレシジョンポイントまたは変換変数で終了する複合変換です。

詳細

HERE プログラムコマンドは、現在のロボットのロケーションに相当する変換またはプレシジョンポイント変数の値を設定します。

通常、ロボットのロケーションは、読み取ったジョイントエンコーダのその瞬間の値で決定されます。ロボットにおいてバックラッシュまたは線形補償が可能である場合、命令されたロボットのロケーションが使用されます。

location_var が複合変換の場合、右端の変換だけが定義されます。その値は他の変換によって決められる基準フレームを基準として、現在のロボットのロケーションに相当するように設定されます。他の変換の中に未定義のものがあると、エラーメッセージが生じます。

例

次の例では、変換「part」を現在のロボットのロケーションに等しくなるように設定します。

```
HERE part
```

次の例では、現在のロボットのロケーションをプレシジョンポイント「#part」に割り当てます。

```
HERE #part
```

関連キーワード

HERE（モニタコマンド）

HERE（変換関数）

SELECT（プログラムコマンド）

SELECT（実数値関数）

SET

IF ... GOTO

論理式の値が TRUE （非 0）の場合に分岐して指定されたステップに飛びます。

構文

IF logical_expr GOTO label

使用上の留意事項

一般的に、このコマンドよりも IF ... THEN コマンドを使う方が好ましいプログラム習慣です。

パラメータ

logical_expr	実数式であり、その値は TRUE(0 以外) か FALSE(0) であるかを試験されます。
label	その実行が分岐するプログラムステップのラベルです。

詳細

この式の値が 0 でない場合、実行プログラムは分岐して指定したものに一致するラベルを持つキーワードの実行を開始します。この式の値が 0 の場合、通常どおり次のキーワードが実行されます。

指定されたステートメントラベルが未定義の場合、そのプログラムは実行できません。エディタを終了する際、またはディスクファイルからメモリへプログラムを読み込ませる際に、未定義のラベルへ分岐させようとしている箇所が特定されます。

IF ... GOTO が最も一般的な使用されるのは、エラーであれば終了するという条件の場合です（以下の例を参照してください）。

例

次の例では、I/O の作動を確認し、I/O エラーが生じた場合、ラベルに分岐します。

```
        ATTACH(dlun, 4) "DISK"
        IF IOSTAT(dlun) < 0 GOTO 100
        FOPENW(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
            ...
        FCLOSE(dlun) "my_file"
        IF IOSTAT(dlun) < 0 GOTO 100
        DETACH(dlun)
    100 IF IOSTAT(dlun) < 0 THEN
            TYPE $ERROR(IOSTAT(dlun))
        END
```

関連キーワード

GOTO

IF ...THEN

IF ...THEN

論理式の結果に応じてキーワードグループ（または 2 つのグループのうちの一方）を条件に合わせて実行します。

構文

IF logical_expr THEN

first keyword area

ELSE

second keyword area

END

使用上の留意事項

すべての IF ... THEN コマンドには関連する END コマンドが存在する必要があります。

上記の構文項に示す second keyword area に項目がない場合、ELSE コマンドを省略できます。

パラメータ

logical_expr	実数値式であり、その値は TRUE(0 以外) か FALSE(0) であるかを試験されます。

詳細

この制御構造体は、2 つあるキーワードグループの一方を条件に応じて実行します。この制御構造体は以下のようにして機能します（上述の「構文」項を参照してください）。

1. logical_expr を判定する。結果が TRUE の場合、以下の項目 2 へ進む。結果が FALSE （0）の場合、スキップして項目 4 へ進む。
2. first keyword area を実行する。
3. スキップして以下の項目 5 に進む。
4. ELSE コマンドがある場合、second keyword area を実行する。
5. END コマンドの次のキーワードまで実行プログラムを継続する。

上記の構文の項に示すように、ELSE および END ステップが行に含まれている必要があります。

構造体の中のいずれかのグループに入れることが可能なキーワードには制限がなく、必要に応じてネストされた IF 構造体にすることができます。

例

以下の例では、変数「row」の値が 5 よりも大きい場合、式 row > 5 は TRUE（-1.0）であり、変数「spacing」には値 10 が割り当てられます。変数「row」の値が 5 未満の場合、式 row > 5 は FALSE（0）であり、変数「spacing」には値 20 が割り当てられます。

```
21  IF row > 5 THEN
22      spacing = 10
23  ELSE
24      spacing = 20
25  END
```

次のプログラム例では、変数「input.signal」が定義されているかどうかを判定します。定義されている場合、プログラムは「input.signal」の値が示す信号を確認し、その設定に応じて異なるメッセージを入力します。外側の IF には ELSE 節は含まれません。

```
IF DEFINED(input.signal) THEN
    IF SIG(input.signal) THEN
        TYPE "The input signal is ON"
    ELSE
        TYPE "The input signal is OFF"
    END
END
```

関連キーワード

CASE

DEFINED

ELSE

IF ... GOTO

IGNORE

REACT または REACTI プログラムコマンドの効果を無効にします。

構文

IGNORE signal

使用上の留意事項

インストールされ入力として設定されたデジタルI/O 信号しか、応答をモニタするためには利用できません。

REACT IGNORE プログラムコマンドを実行する必要があります。

パラメータ

signal デジタル入力信号の番号です(以下の詳細を参照)。

詳細

IGNORE プログラムコマンドは指定された信号の連続モニタリングを無効にし、この信号に対する最後の REACT または REACTI の効果を無効にします。

次の信号を REACT プログラムコマンドと共に使用することができます。

信号タイプ	範囲	詳細
デジタル入力	1001～1999	信号は、システム内に実際に存在している場合に限って使用可能です。
ソフトウェア	2001～2999	すべての信号が使用可能です。
ホスト	4001～4999	

例

以下の例では、「test」の値で特定したデジタル入力またはソフト信号のモニタリングを停止します。

```
IGNORE test
```

関連キーワード

LOCK

REACT

REACTI

JMOVE

ジョイントの値のリストに記載された位置にすべてのロボットのジョイントを動かします。ロボットはジョイント補間モードで協調的な動作を実行します。

構文

JMOVE expression1,...,expressionn

使用上の留意事項

ロボットを動かすために、1 つ以上の式（ジョイント）を指定する必要があります。

パラメータ

expression1　ジョイント1 の値を指定するオプションの式です。

expressionn　N 番目のジョイントの値を指定するオプションの式です。

詳細

最大で 12 個の式を指定できます。

式を省略した場合、対応するジョイントは動きません。

ロボットのジョイント数よりも多くの式を指定した場合、余分な式は無視されます。

例

次の例では、ジョイント 1 とジョイント 3 を変数「j1」と「j3」で表される位置まで動かします。

```
JMOVE j1,,j3
```

JOG

ロボットの指定されたジョイントをジョグしたり、指定されたデカルト座標系の座標軸方向に沿ってロボットツールを移動します。

構文

JOG (status) **robot, mode, axis, speed,** location, appro_dist

使用上の留意事項

JOG プログラムコマンドを実行する度に、ロボットは JOG.TIME システムパラメータで指定した時間だけ動きます。

COMP 以外のモードを使用している場合、他のタスクは指定されたロボットに割り付けできません。割り付けを試みた場合、エラーメッセージ「Robot interlocked」（ロボットがインターロックしました）が表示されます。現在のプログラムでロボットを割り付けることはできますが、必ずしも割り付ける必要はありません。JOG プログラムコマンドの実行時にロボットがアタッチされていない場合、JOG プログラムコマンドが終了した後、他のモーションコマンドを実行する前に、ロボットをアタッチして下さい。

JOG コマンドでロボットを動かした後、（ペンダントで手動モードが選択されていたかのように）システムは手動モードのままとなります。JOGプログラムコマンドのmode 5（またはペンダント）を使用して、COMP モードに戻すことができます。それ以外の場合にタスクがロボットへの割り付けを実行しようとすると、エラー「COMP mode disabled」（COMP モード無効）が返されます。

ジョイントが範囲外の場合、JOG コマンドを使用してジョイントを範囲内に戻すことができます。

パラメータ

status　成功の場合1、それ以外の場合はeV+ エラーコードを返す状態変数です。オプションです。

robot　ロボットの番号を指定します。

mode　次に示すジョグモードを指定します。

-1	キープアライブモード。JOG.TIME システムパラメータを使用して指定した時間まで前のコマンドを継続します。
1	フリージョイントモード。speedの値が正なら、指定されたジョイントをフリーモードにします。speedの値が負なら、指定されたジョイントのフリーモードを解除します。
2	各ジョイント制御

3	ワールド座標系制御
4	ツール座標系制御
5	COMP モードに戻します。
6	未使用
7	指定されたロケーションに向かって指定された速度でジョグします。
8	最も近いワールド軸に対してロボットツールのZ 軸が平行になるようにジョグします。
9	指定されたロケーションによって定義された座標系に対するデカルト制御

axis 指定されたジョグモード（上記参照）に従って目的の動作をするために、ジョイント番号または直交座標（X=1、Y=2、...）を指定します。

モード 7、8 の場合、このパラメータは無視されますが値は必ず指定する必要があります。

speed 動作の速度と方向を指定します。これは、手動モードでの速度率（%）として解釈されます。100 以上の値は 100%、-100 以下の値は -100% として解釈されます。

フリーモードが指定されている場合、speedの値が正なら、指定されたジョイントをフリーモードにします。speedの値が負なら、指定されたジョイントのフリーモードを解除します。

location ロボットが移動する目標位置を指定する、変換、プレシジョンポイント、位置関数または複合変換です。オプションです。モード 7、9 以外のすべてのモードでは、このパラメータは無視されるため省略することができます。

appro_dist 距離を指定するオプションの実数値式です。指定したロケーションと実際に希望する目標位置の間の、ロボットツールZ軸に沿った距離を指定する、オプションの実数値式です。距離の値が正の場合、指定されたロケーションからツールを後方（ツールZ軸のマイナス方向）にオフセットします。距離の値が負の場合、ツールを前方（ツールZ軸のプラス方向）にオフセットします。このパラメータはモード 7 以外では使用しません。

詳細

状態変数が与えられ、エラーがある場合、JOG コマンドはプログラムの実行を停止せず、状態変数にエラーを返します。

前回の動作が完了する前に、新しい JOG コマンドを実行できます。長期間の円滑なモーションについて、次のJOGコマンドを前のJOGコマンドのJOG.TIME値の内で実行する必要があります。キープアライブモードは、JOG.TIME のタイムアウト後には効果がありません。ロボットが停止する前だけ有効となります。

コマンドが処理されるとき、次のエラー状態が報告される可能性があります。

- **Mode 1**: 指定されたジョイントにフリーモードが許可されていない場合、エラー「Illegal joint number」（-609、不正なジョイント番号）が返されます。
- **Mode 2**: ロボットがジョイント制御をサポートしていない場合、エラー「Joint control of robot not possible」（-938、ロボットのジョイント制御ができません）が返されます。
- **Mode 3、4、8、9**: ロボットが直交座標制御をサポートしていない場合、エラー「Cartesian control of robot not possible」（-635、ロボットの直交座標制御ができません）が返されます。
- **Mode 7**: ロケーションに到達できない場合、動作が可能な範囲内で動作は停止し、停止時にエラー「Location out of range」（-610、ロケーション範囲外）が返されます。動作中に他の動作エラー（障害物に遭遇など）が発生した場合、関連したエラーが報告されます。
- **Mode 7、9**: ロケーションが指定されていない場合、エラー「Missing argument」（-454、引数がありません）が返されます。モード 7 では、ロケーションが変換で指定されている場合、指定のロケーションに向かって直線動作が実行されます。ロケーションがプレシジョンポイントで指定されている場合、関節補間動作が実行されます。

ロボットのジョイントが範囲外の場合、次のいずれかの方法でジョイントを範囲内にすることができます。

- フロントパネルで MAN モードに入り、手動でジョイントを制御します。
- ペンダントを COMP モードにして、JOG コマンドを使用して範囲内にジョイントを戻します。JOG はペンダントが COMP モードの場合のみ許可されます。

> **注:** ジョイントが範囲外のとき、COMP モードで出来ることは制限されます。この場合、すべての動作コマンド（JOG 以外）はエラー「Position out of range」（位置範囲外）を返します。さらに、JOGは、ジョイントを範囲内に戻す方向のみジョイントを動かすことができます。

例

次の例ではマイナス方向にジョイント 3 をジョグします。

```
JOG 1, 2, 3, -10
```

次の例ではワールドモードでX 軸方向にジョグします。

```
JOG 1, 3, 1, 10
```

次の例ではツールモードでY 軸方向にジョグします。

```
JOG 1, 4, 2, 10
```

次の例ではロボットツールを「loc1」に向かってジョグします。

```
JOG 1, 7, 1, 10,loc1
```

次の例では「loc1」として定義されているロケーションの500 mm 上の位置に向かってジョグします。

```
JOG 1, 7, 1, 10,loc1, 50
```

関連キーワード

JOG

JOG.TIME

MOVE

KEYMODE

ペンダントのキーのグループの動作を設定します。

構文

KEYMODE first_key, last_key = mode

使用上の留意事項

KEYMODE で処理する前に、ペンダントをアタッチする必要があります。

> **追加情報:** キー番号の割り当てについては *『ティーチングペンダント T20 ユーザーズマニュアル (SBCE-392)』* を参照してください。

パラメータ

first_key	影響を受けるキーの集合の最初のキー番号を定義する実数値式です。
last_key	影響を受けるキーの集合の最後のキー番号を定義する実数値式です。
mode	指定されたキーの集合に設定されるキーモードを定義する実数値式です。mode には、次の値のいずれかを設定する必要があります（以下に示すモード）。 0：キーボードモード 1：トグルモード 2：レベルモード

詳細

以下に各キーモードについて説明します。ペンダントとの相互作用の詳細については、PENDANT 実数値関数の説明を参照してください。

0 - キーボードモード

このモードにプログラムされたキーは、ターミナルキーボードと同等の機能になります。プログラムは PENDANT(0) 関数を使用して、次に押された時のキーの番号を要求できます。プログラムは、キーボードモードにプログラムされたキーのいずれかが押されるまで待機し、キーが押されるとそのキーの番号を返します。

先行入力には対応していません。プログラムはPENDANT(0) 関数が実行中でなければ、どのキーの押下も検出しません。

1 - トグルモード

このモードのキーが押された場合、eV+ が保持する内部状態が切り替わります。必要であれば現在のキーの状態を評価できます。

2 - レベルモード

現在のキーのレベルがペンダントによって保持され、そのレベルを要求することができます。キーが押されている間、キーの状態は ON です。このモードはカーソルの制御などに役立ちます。ペンダントがアタッチされていない場合、戻り値は無効です。

アタッチ／デタッチに関する要件

プログラムが PENDANT 関数を使用してキーを読み取ったり、キーのモードを設定したり、文字を画面に送信する前に、ペンダントをアタッチする必要があります（ATTACH プログラムコマンド使用）。

デフォルト

ペンダントがアタッチされると、キーモードはデフォルトでキーボードモードになります。

例

次の例では、手動制御ソフトキーをレベルモードに設定します。

```
KEYMODE 1,5 = 2
```

関連キーワード

ATTACH

KILL

プログラム実行スタックをクリアし、割り付けられている I/O デバイスを切り離します。

構文

KILL task_number

使用上の留意事項

指定されたプログラムタスクの実行中は、KILL プログラム命令を使用できません。

指定されたタスクの実行スタックが空の場合、KILL プログラム命令に効果はありません。

パラメータ

task_number クリアするプログラムタスクを指定する、整数と解釈されるオプションの実数値、変数または式です。

詳細

KILL プログラムコマンドオペレーションは、選択したプログラム実行スタックをクリアし、開かれているファイルを閉じ、異常なプログラム終了によって割り付けられたままになっているおそれのある I/O デバイスを切り離します。

プログラムが PAUSE プログラム命令を実行した場合、プログラムが ABORT キーワードによって終了した場合、または I/O デバイスが割り付けられているもしくはファイルが開いている間にエラー状態が発生した場合、この状況が生じます。アクセスが制限されている I/O デバイスが接続されている場合、切り離されるまで他のプログラムタスクはそのデバイスを使用できません。

タスク番号を省略すると、KILL コマンドは常にタスク 0 にアクセスします。

例

次の例では、信号 2001 が TRUE の場合タスク 1 に対して KILL 操作を実行します。

```
IF (SIG(2001)==TRUE ) THEN
    KILL(1)
END
```

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

STATUS

LEFTY

次の動作に関するロボットの設定変更を要求して、スカラロボットの最初の 2 つのリンクが左アーム方向を使用するようにします。

構文

LEFTY

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがLEFTY設定をサポートしていない場合、ロボットはこの命令を無視します。

LEFTY プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによってアタッチされていない限り、任意のプログラムタスクによって実行することができます。ロボットがアタッチされていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、LEFTY コマンドを実行するとエラーになります。

以下の図は、スカラロボットの平面図におけるLEFTY/RIGHTY 設定を示したものです。

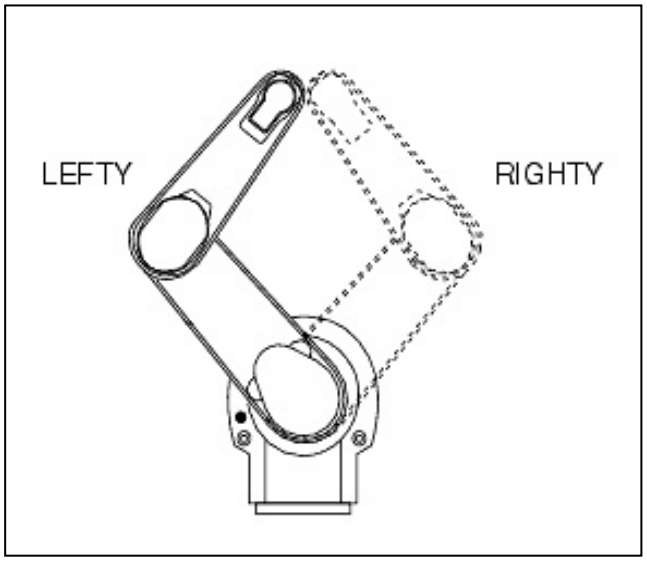


図3-9. LEFTY/RIGHTY 設定

例

次の例では、LEFTY設定で位置「point1」にロボットを移動します。

```
LEFTY
MOVE point1
```

関連キーワード

CONFIG

RIGHTY

SELECT（プログラムコマンド）

SELECT（実数値関数）

SELECT(モニタコマンド)

LOCAL

現在のプログラム内でのみ定義される恒久的変数を宣言します。

構文

LOCAL type **variable, ...,** variable

使用上の留意事項

複数のプログラムタスクがサブルーチンを同時に呼び出すこと、および 1 つのタスクがサブルーチンを繰り返し呼び出すことが可能です。こういった呼び出しを不注意に発生させると、ローカル変数およびグローバル変数が破損する可能性があります。そのため、ローカル変数の代わりに自動変数を使用することを推奨します。

LOCAL ステートメントはプログラム内の実行可能なステートメントの前に必ず現れなければなりません。

変数が LOCAL ステートメントに記載されている場合、その変数と同じ名称のグローバル変数にプログラムから直接アクセスすることはできません。

ローカル変数の値は、STORE または LOAD モニタコマンドによっては保存および修復されません。

パラメータ

type

type パラメータを指定すると、すべての変数がそのタイプ（REAL、DOUBLE または LOC）に一致する必要があります。配列変数には、インデックスを明確に指定してその配列の最高次の有効なインデックスを示す必要があります。

このキーワードを省略すると、それぞれの変数のタイプはプログラム内での使用によって決まります。使用によってタイプを決定できない場合、エラーが発生します。

LOC — ロケーション変数 (変換 またはプレシジョンポイント) です。

REAL — 単精度実数型変数です。

DOUBLE — 倍精度実数型変数です。

デフォルトタイプに関する詳細については GLOBAL コマンドを参照してください。

variable

変数名 (ベルト、プレシジョンポイント、実数、文字列、変換) です。それぞれの変数は単純な変数または配列である場合があります。type パラメータを指定すると、すべての変数がその指定したタイプに一致する必要があります。配列変数にはインデックスを指定しないでください。

詳細

このコマンドは現在のプログラム内だけで定義する変数を宣言するために使用されます。ローカル変数はそのプログラム内でのみ参照することができます。他のプロ

グラムでローカル変数の名称が定義されているかどうかには関わらず、ローカル変数の名称を選択することができます。

ローカル変数はプログラム実行中に 1 回だけ割り当てられ、その値は連続するサブルーチンの呼び出しの間で保存されます。これらの値は、複数のプログラムタスクによって同じプログラムが実行される場合も共有されます。

LOCAL（またはグローバル）変数を使用するプログラムが、いくつかの異なるプログラムタスクによって呼び出される場合や、単一タスクに繰り返し呼び出される場合、これらの変数の値は異なるプログラムインスタンスによって変更されたり、予期できないプログラムエラーを生じる可能性があります。そのため、こういったエラーの可能性を最小限に抑えるために、すべての一時的変数に対して自動変数を使用するようにしてください。詳細については AUTO コマンドを参照してください。

変数は自動、グローバルまたはローカルとして定義することができます。一旦ある変数をある分類に割り当ててしまうと、その変数を他の分類に割り当てようとすると、「Attempt to redefine variable class」（変数の分類を再定義しようとしました）のエラーメッセージが生じます。

変数は同じコンテクストの内で 1 回に限り定義することができます（自動、ローカルまたはグローバル）。異なるタイプで複数回、変数を定義しようとすると、エラーメッセージ「Attempt to redefine variable type」（変数の分類を再定義しようとしました）が表示されます。

次の例に示すように、ローカル変数はオプションのコンテキスト指定子 @ を使用することで、DELETE_、DO、HERE、LIST_、TEACH、およびTOOL などのモニタコマンドで参照することができます。

```
command @task:program command_arguments
```

例

以下の例のステートメントでは、変数「loc.a」、「$ans」および「i」を現在のプログラムに対してローカルにすることを宣言します。

```
LOCAL loc.a, $ans, i
```

関連キーワード

AUTO

LOCAL

STACK

DO

HERE

TEACH

TOOL

LOCK

プログラム応答ロックアウト優先度を与えられた値に設定します。

構文

LOCK priority

使用上の留意事項

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、LOCK 0 が仮定されます。

優先度を変更すると応答の処理方法に影響する場合があります。このコマンドを使用する前に、どの応答（およびその優先度）を有効にするかを確認してください。

パラメータ

priority 0～127 の値を持つ実数値式であり、この値が新しい応答ロックアウト優先度になります。

詳細

プログラムを実行すると、実行スタックに置かれます。プログラムのタスクがタイムスライス内で最も高い優先度になると、そのプログラムの priority は 0 に設定され、実行が開始されます。実際に実行している間、プログラムのタスクはタイムスライスの最後で中断することができ、その際、そのタスクが次にタイムスライス内で最も高い優先度になるまで待機します。LOCK コマンドはタイムスライス内のタスク優先度の値に影響を与えません。単に実行中のプログラムのプログラム優先度を変えます。

応答ルーチン（REACT、REACTE、REACTI）を呼び出す場合、プログラム優先度が重要になります。一時的プログラムの優先度を、REACT または REACTI プログラムの優先度よりも高い値に設定にすることによって、プログラムは REACT または REACTI ルーチンの実行を延期することができます。これが LOCK コマンドの機能です。例えば、LOCK コマンドが一時的プログラム優先度を 20 に変更すると、より低い優先度値の REACT または REACTI は延期されます。

> **追加情報:** REACTI オペレーションの実行が延期されている場合に LOCK コマンドを使用してその REACTI オペレーションを高優先度に変更すると、システムはそのオペレーションを実行します。これが通常の REACTI オペレーションの動作と異なる点です。システムはそのオペレーションを実行した後に REACTI オペレーションを実行します。

REACTI オペレーションの実行が延期されている場合に LOCK コマンドを使用してその REACTI オペレーションの優先度を高い優先度に設定すると、現在のコマンドが停止なしで即座に実行されます。現在のコマンドを実行した後に、REACTI コマンドが発行されます。REACTE ルーチンは優先度の考慮によって延期させることができません。

延期されている応答は無視されません。LOCK コマンドが処理される都度、延期されている応答プログラムについて、その優先度が実行する順位になっていないかチェックします。そのプログラムの優先度が低くなると、より高い優先度を持つす

べての保留されている応答ルーチンが、その相対的な優先度に応じて実行されます。

PRIORITY実数値関数を使用して、プログラムの優先度をいつでも判定することができます。

> **注:** LOCK コマンドを使用して応答プログラム内のプログラム優先度を変更することができますが、そのプログラム内で RETURN を実行すると、依然として優先度は応答前の値に戻ります。これは、RETURN を応答プログラムから実行した場合しか発生しません。

例

以下の例では、プログラムの優先度を 10 増やします。

```
LOCK PRIORITY+10
```

関連キーワード

PRIORITY

REACT

REACTI

MC

モニタコマンドプログラム 内にモニタコマンドを作成します。

構文

MC monitor_command

使用上の留意事項

MC コマンドは モニタコマンドプログラム 内にしか含めることができません。モニタコマンドプログラム には MC コマンド、空白行およびコメント行しか含めることができません。

> **追加情報:** モニタコマンドプログラム に関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

monitor_command	任意の有効な eV+ モニタコマンドキーワードです。

詳細

どれか 1 つの eV+ エディタを使用してコマンドプログラムを作成することができます。エディタに対して モニタコマンドプログラム が作成中であることを示すためには、 モニタコマンドプログラム のすべてのオペレーション行が MC の文字とそれに続く 1 つ以上のスペースで始まる必要があります。通常のアプリケーションプログラム同様、 モニタコマンドプログラム にはブランク行およびコメント行を明確化のために含めることができます。

モニタコマンドプログラム のすべての非空白行にはモニタコマンドまたはコメントが含まれる必要があります。モニタコマンドとプログラム命令を混在させることはできません。プログラムコマンドは DO コマンドを使用して含めることができます。「MC DO」の形式でラインを入力することができます。

例

次の モニタコマンドプログラム の例では、ディスクファイルを読み込んで、プログラムの実行準備をしてから、実行を開始します。DO コマンドは MOVE コマンドを含めるために使用されています。

```
1 .PROGRAM setup()
2         MC LOAD C:project
3         MC LOAD B:project.lc
4         MC SPEED 50
5         MC DO MOVE safe.loc
6         MC EXECUTE motion, -1
7 .END
```

関連キーワード

COMMANDS

DO

MCS

MCS

アプリケーションプログラムからモニタコマンドを呼び出します。

構文

MCS string

パラメータ

string	以下の eV+ モニタコマンドの 1 つの定義する文字列値、変数または式です。DELETE、DELETEL、DELETEM、DELETEP、DELETER、DELETES、FCOPY、LOAD、STORE、STOREL、STOREM、STOREP、STORER、STORES

詳細

モニタコマンドは通常、モニタウィンドウから、またはモニタコマンドだけを含むモニタコマンドプログラム から呼び出されます。MCS コマンドを使用すると次のモニタコマンドをアプリケーションプログラムから呼び出すことができます。

DELETE	DELETEL	DELETEM	DELETEP	DELETER
DELETES	FCOPY	LOAD	STORE	STOREL
STOREM	STOREP	STORER	STORES	

これらのコマンドを使用すると、アプリケーションプログラム上でプログラムをディスクとの間で保存、読み込み、およびコピーすること、ならびにメモリからプログラムを削除することが可能です。同様に、もはや不要になった変数をメモリから削除することができます。また、画像センサのプロトタイプの名称を変えることもできます。

eV+ におけるマルチタスクアクセスでは、プログラムおよびグローバル変数の読み込み、保存および削除にはインターロックはかかりません。そのため、1 つのプログラム内に複数の MCS コマンドがある場合、TAS インターロックを使用して複数のタスクでコマンドが発行されないようにする必要があります。詳細については TAS コマンドを参照してください。

> **注:** 文字列パラメータに指定されているモニタコマンドに空のプログラムコンテクスト（@ が含まれる）が含まれる場合、そのコマンドに記載されている変数は、MCS コマンドを含むプログラム内で参照されているものとして取り扱われます。プログラムコンテキストに関する詳細情報については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

このモニタコマンドを処理中にエラーが発生しても、実行プログラムは停止しません。MCS コマンドの後で ERROR 実数型関数を使用してエラーの発生をチェックすることができます。

> **注:** サブルーチン内で DELETE_ コマンドを使用して、（.PROGRAM ステートメント内の変数の 1 つである）サブルーチンのパラメータの 1 つを削除する場合、その変数は削除されず、エラー状態も記録されません。

FCOPY オプションを使用する場合、論理ユニット 5（ディスク 1）および 6（ディスク 2）が使用可能である必要があります。LOAD または STORE_ を使用する場合、論理ユニット 5 が使用可能である必要があります。

例

以下の例では、ディスクファイルを読み込んで、ファイル内のプログラムを実行し、システムメモリからプログラムを削除します。次いで、別のプログラムファイルをメモリに読み込んで実行します。

この単純な例は モニタコマンドプログラム を使用しても実行可能ですが、次では一般的プログラムにおける MCS コマンドの使用を実演しています。

```
.PROGRAM admin()
        MCS "LOAD C:setup"
        CALL setup
        MCS "DELETEP setup"
        MCS "LOAD C:demo_1"
        CALL demo_main
.END
```

関連キーワード

DELETE

DELETEL

DELETEM

DELETEP

DELETER

DELETES

ERROR

FCOPY

LOAD

MC

STORE

STOREL

STOREM

STOREP

STORER

STORES

TAS

MOVE

ジョイント補間モーションを用いて所定のロケーションによって記述されるポジションと方向に至るロボットモーションを開始します。

構文

MOVE location

使用上の留意事項

MOVE はジョイント補間モーションを発生させます。

このコマンドは、タスクがロボットをアタッチしている限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行するとエラーになります。

パラメータ

location ロボットモーションの目的のロケーションを指定する、変換、プレシジョンポイント、ロケーション関数または複合変換です。

詳細

MOVE プログラムコマンドはジョイント補間モーションを発生させます。ロボットの最初と最後のロケーションとの間の中間設定ポイントは、最初と最後のポジションを補間して算出されます。最後の MOVE オペレーション以降にプログラムが必要とする設定の変更（例えば、LEFTY コマンドの発行）は、そのモーション中に実行されます。

例

次の例では、ジョイント補間モーションを用いてプレシジョンポイント「#pick」によって記述されるロケーションに移動します。

```
MOVE #pick
```

関連キーワード

APPRO

APPROS

DEPART

DEPARTS

MOVEC

SELECT（プログラムコマンド）

SELECT（実数型関数）

MOVEC

与えられた地点によって記述される位置および方向を使用して円形／円弧経路のロボット動作を開始します。

構文

MOVEC(angle, turn) **location1, location2**

MOVEC(angle, turn) **center**

使用上の留意事項

MOVEC プログラムコマンドは、タスクがロボットに割り付けてあれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、このコマンドを実行するとエラーになります。

パラメータ

angle	円弧の角度(°)を指定する実数値式です。location2 を指定する場合、このパラメータはオプションになります。angle パラメータはプラスまたはマイナスの数値にすることができますが -360～+360 の範囲内で指定する必要があります。プラスの値は時計回りツール動作になります。マイナスの値は反時計回りツール動作になります。
turn	ツールが円弧に沿って回転するかどうかを指定するオプションのブール式です。turn が省略されるか 0 である場合、ツールの方向は MOVEC (angle) center ステートメントに対して一定になり、MOVEC location1, location2 ステートメントの location2 の方向で終了します。turn が 0 以外の場合、ツールの方向は円の中心を軸として円弧の角度だけ回転し、その軌道を基準として一定の方向を維持します。
center	円の中心を指定する、変換、プレシジョンポイント、位置関数または複合変換です。
location1	ロボットの動作軌道となる円 / 円弧上の中間位置を指定する、変換、精密位置、位置関数または複合変換です。
location2	ロボットの動作軌道となる円 / 円弧の終点を指定する、変換、精密位置、位置関数または複合変換です。このパラメータが与えられない場合は、angle を指定する必要があります。

詳細

MOVEC(angle,turn) **location1, location2**

MOVEC プログラムコマンド構文は、location2 パラメータによって定義される位置を終点とする円／円弧経路を作成するために指定されます。連続経路動作を使用する場合、この円／円弧経路を、現在のロボット位置または現在のロボット目標位置から開始することができます。以下の図で示すように、location1 パラメータが

指定する中間位置を使用して、その円が存在する平面および円弧の角度を定義します。

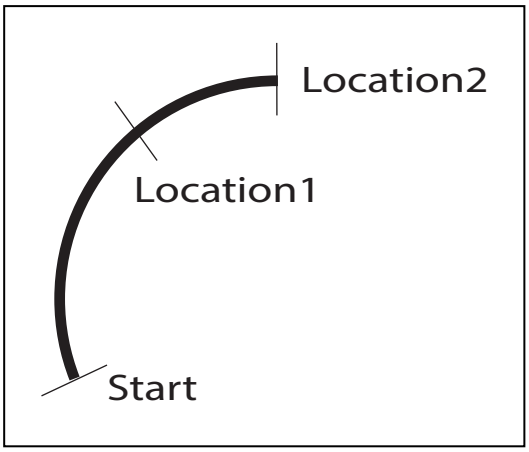


図3-10. 3 点による平面および角度の定義

3 点が一直線上にあるか、そのうちの 2 つが同一である場合、MOVEC コマンドを発行すると、円／円弧ではなく直線動作が生じます。

この構文を使用する場合、location1 の方向は使用されません

angle を指定した場合、ロボットは angle の角度だけ移動し、必ずしも location2 で動作を終了するわけではありません。終点を定義する上で、angle は location2 よりも優先されます。

angle を指定すると、location2 の方向は無視されます。終点における方向はすべて turn パラメータによって決まります。turn パラメータが省略されるか 0 である場合、終点における方向は始点における方向になります。turn パラメータが 0 以外の場合、終点における方向は、円弧の中心を軸として始点を angle だけ回転させた位置になります。

turn が 0 以外の場合、円を含む平面がツール中心点の X-Y 平面に平行でないと、MOVEC コマンドによって 4 軸ロボットに対する *invalid orientation*（無効な方向）エラーが発生します。

直線動作同様、円動作は多回転と互換性を持ちます。location2 がプレシジョンポイントの場合、多回転関節は 360° 以上回転することができます。

MOVEC(angle,turn**) center**

この MOVEC プログラムコマンド構文は、現在のロボット位置から、または連続経路動作の場合、現在のロボット目標位置から始まる円／円弧経路動作を作成するように設計されています。この経路は center 地点を中心軸とします。終点は angle の角度によって指定されます。

円を含む平面は、始点を通り center 位置の X-Y 平面に平行な平面と定義されます。center 位置の Z 軸方向が center および始点を通る直線に直交していない場合、その円の中心は center の位置ではありません。その位置は center の Z 軸とこの軸に直交し、始点を通る平面の交点になります。

円の中心を定義すると、その円の半径は始点から中心までの距離になります。

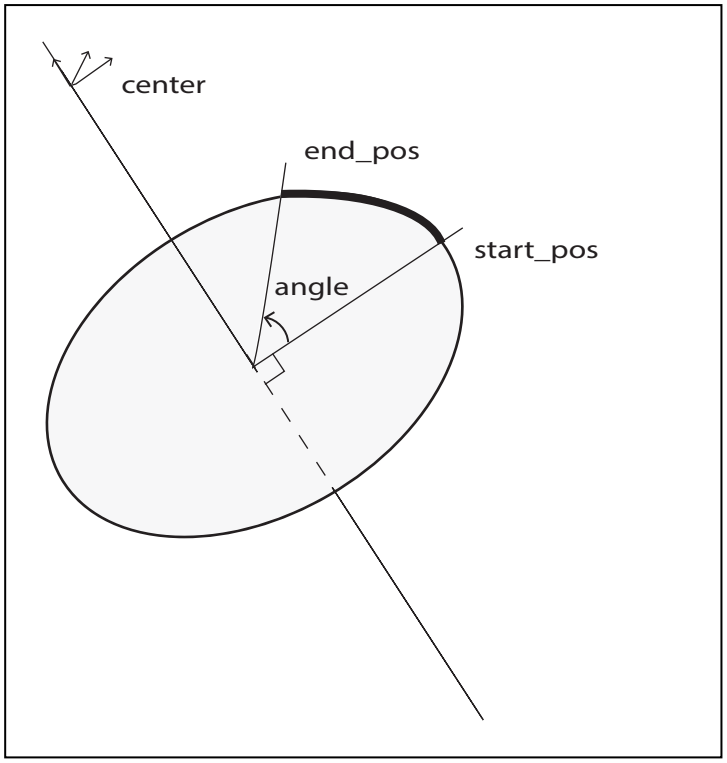


図3-11. 角度と中心の定義

例

以下の例では、MOVEC を変換と共に使用しています。

```
MOVEC loc1, loc2
```

以下の例では、以下の図に示すように、MOVEC を連続経路動作と共に使用しています。

```
MOVES p1
MOVEC p2, p3
MOVES p4
BREAK
```

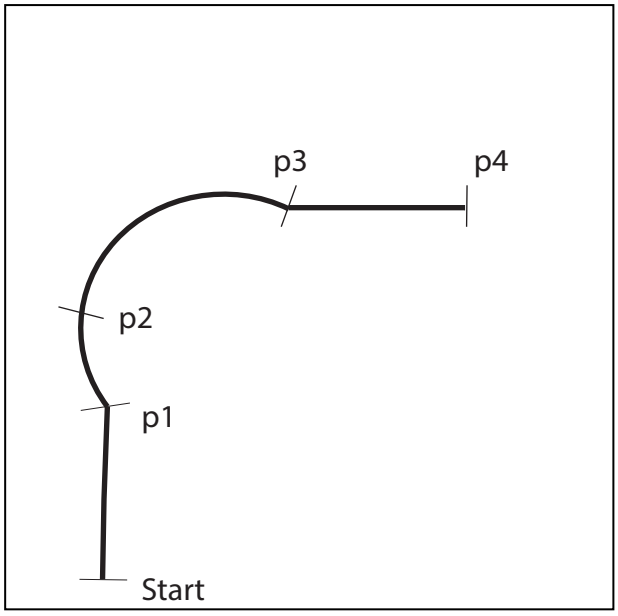


図3-12. MOVEC を使用した連続経路

以下の例では、以下の図に示すように、MOVEC を連続経路動作と共に使用しています。

```
SET center = TRANS(300,0,210,90,30,0)
SET start_pos = TRANS(420, 0, 210, 0, 180, 0)

MOVES start_pos
BREAK

MOVEC(360) center
BREAK
```

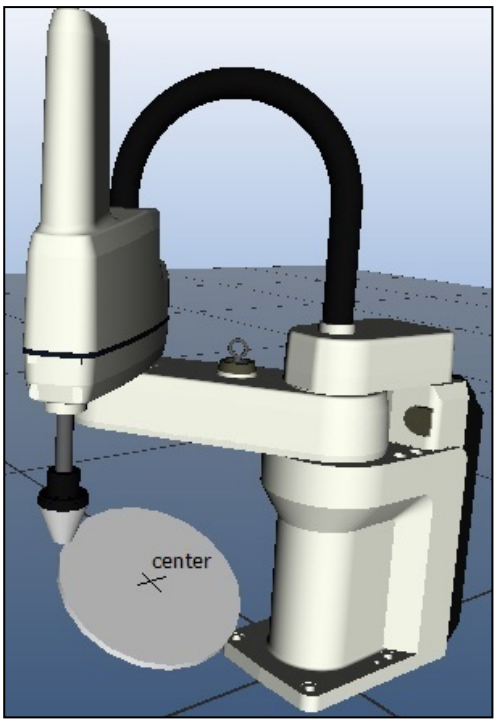


図3-13. MOVEC により完全な円を作成

注: 上図では、ロボットが水平ではない円を基準として一定の方向を維持することができないために、"MOVEC(360,1) center" は *invalid orientation* エラーを返します。

以下の例では、MOVEC を使用して、6 軸ロボットを用いた塗布のために、方向を回転させる半円を作成します。

```
SET center = TRANS(300,0,250,0,-150,0)

SET start_pos = center:TRANS(0,0,0,-90):TRANS(100,,,,-30)


MOVES start_pos

BREAK


MOVEC(180,TRUE) center

BREAK
```

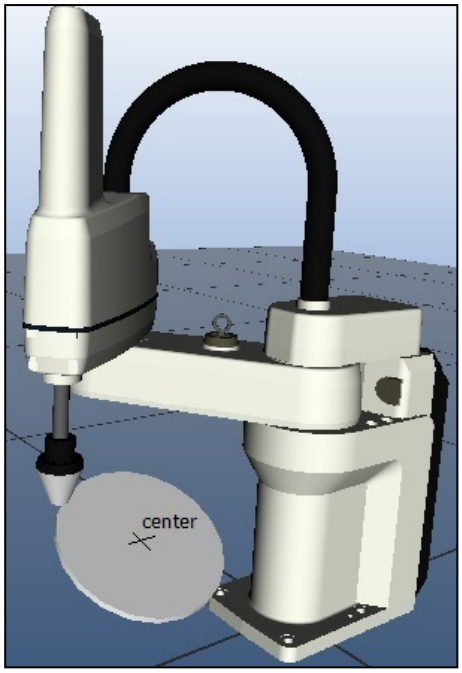


図3-14. ツール方向を回転させながらの 6 軸 MOVEC 半円

最後の 3 行を変えると、以下に示すように、上記の例において、方向を回転させないで同じ動作を作成することができます。

```
MOVEC(180) center

BREAK
```

IMPROVE IMAGE BELOW

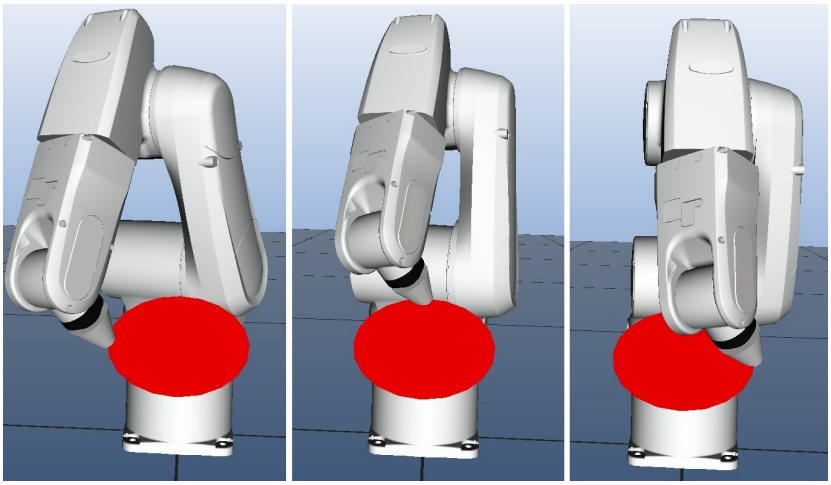


図3-15. ツール方向を回転させない 6 軸 MOVEC 半円

関連キーワード

APPRO

DEPART

MOVE

MOVES

SELECT （プログラムコマンド）

SELECT（実数値関数）

MOVES

直線モーションを用いて所定のロケーションによって記述されるポジションと方向に至るロボットモーションを開始します。

構文

MOVES location

使用上の留意事項

MOVES は直線モーションを発生させ、その間、設定の変更は許容されません。

このコマンドは、タスクがロボットをアタッチしている限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行するとエラーになります。

パラメータ

location ロボットモーションの目的のロケーションを指定する、変換、プレシジョンポイント、ロケーション関数または複合変換です。

詳細

MOVES プログラムコマンドは直線モーションを発生させます。そのようなモーション中、ツールは直線パスに沿って移動し、その最終の方向に滑らかに回転します。

例

次の例では、直線パスに沿って、複合変換「ref:place」によって記述されるロケーションに移動します。

```
MOVES ref:place
```

関連キーワード

APPRO

APPROS

DEPART

DEPARTS

MOVEC

SELECT（プログラムコマンド）

SELECT（実数値関数）

MULTIPLE

ロボット手首関節を完全に回転させることを可能にします。

構文

MULTIPLE ALWAYS

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボット動作しか影響を受けません。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、MULTIPLE ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

MULTIPLE プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+システムがロボットを制御するように設定されていない場合、MULTIPLEコマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として MULTIPLE を適用するオプションの修飾子です。ALWAYS を指定すると、SINGLEプログラムコマンドによって無効になるまで、MULTIPLEは有効のままになります。ALWAYS を指定しない場合、MULTIPLE コマンドは次のロボット動作だけに適用されます。

詳細

MULTIPLE が有効な場合、動作計画および実行の際、手首関節の完全な回転を必要に応じて使用します。

NOOVERLAP が有効な場合、MULTIPLE 設定は無視されます。

関連キーワード

NOOVERLAP

SINGLE

NEXT

分岐して n 番目にネストされたループのEND文に進み、ループテストを実行し、適切であればループします。

構文

NEXT count

使用上の留意事項

このコマンドは FOR、WHILE および DO 制御構造体と共に使用することができます。

パラメータ

count	その末尾へと分岐して進むネスト構造の番号を指定するオプションの整数です(式および変数は許容されません)。

詳細

NEXT コマンドが count = 1 で処理される場合、その実行は制御構造体の末尾まで継続します。count > 1 の場合、その実行はその count 番号のネストされている制御構造体の末尾まで継続します。

例

error = 1 の場合、分岐して最も内側の制御構造体の末尾に進みます。error = 2 の場合、分岐して最も外側の制御構造体の末尾に進みます。

```
FOR i = 1 to 20
    FOR j = 1 to 10
        FOR k = 10 to 50
            IF error == 1 THEN
                NEXT
            END
            IF error == 2 THEN
                NEXT 3
            END
        END
    END
END
```

関連キーワード

DO

END

EXIT

FOR

WHILE

NOFLIP

次の動作に関するロボットの設定変更を要求して、ロボットの手首のピッチ角がプラスの値になるようにします。

構文

NOFLIP

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがNOFLIP 設定をサポートしていない場合、ロボットはこの命令を無視します。

NOFLIP プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。ロボットが割り付けられていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、NOFLIP コマンドを実行するとエラーになります。

詳細については FLIP プログラムコマンドを参照してください。

例

次の例では、NOFLIP 設定で「point1」にロボットを移動します。

```
NOFLIP
MOVE point1
```

関連キーワード

CONFIG

FLIP

SELECT（プログラムコマンド）

SELECT（実数値関数）

SELECT(モニタコマンド)

NONULL

eV+ システムに対して、連続経路動作の末尾で位置誤差がヌル処理されるのを待たないように命令します。

構文

NONULL *ALWAYS*

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボット動作しか影響を受けません。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、NULL ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

NONULL プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、NONULL コマンドを実行するとエラーになります。

エミュレーションモードで使用した場合、このキーワードを実行してもオペレーションの違いは発生しません。エミュレーションモードにおけるロボットの軌道は現在のターゲット位置と同じとみなされます。そのため、実際のロボットを使用した場合と異なり、そのオペレーションが完了したときに位置誤差は生じません。

パラメータ

ALWAYS	デフォルト条件として NONULL を確立するオプションの修飾子です。NONULL コマンドに ALWAYS が含まれる場合、NULL プログラムコマンドによって無効になるまで NONULL は継続的に有効です。ALWAYS を指定しない場合、NONULL コマンドは次のロボット動作だけに適用されます。

詳細

NONULL が有効であり、ロボット動作に中断が発生した場合、eV+ はすべての動作する関節が指定位置に到達したことをサーボが通知するのを待つことなく、次の動作を開始します。割り当てられた時間の末尾において、eV+ はすべての関節が終点に到達したと仮定し、次の動作の命令を開始します。

COARSE モード同様、このモードでは、終点での高い精度が不要であれば、より速い動作が可能になります。

> **注:** NONULL が有効な場合、位置誤差チェックは無効であり、大きな位置誤差が発生する可能性があります。

例

次の例では、位置「point1」に対して一度だけ動作するNONULL 操作を発行します。

```
NONULL
MOVE point1
```

次の例では、位置「point2」と「point3」に対して連続動作する NONULL 操作を発行します。

```
NONULL ALWAYS
MOVE point2
MOVE point3
```

関連キーワード

COARSE

CONFIG

DELAY.IN.TOL

FINE

NULL

SELECT（プログラムコマンド）

SELECT（実数値関数）

SELECT(モニタコマンド)

NOOVERLAP

制限停止を回避するために、選択した多回転軸が ±180° 以上回転する後続モーションを計画した場合、プログラムエラーが生じます。

構文

NOOVERLAP *ALWAYS*

使用上の留意事項

NOOVERLAP は以下のロボットおよび該当するジョイントに適用されます。

- 6 軸ロボットのジョイント 1、4 および最終ジョイント
- スカラロボットのジョイント 4
- 並行ロボットのツール方向

プログラム実行が開始される場合、および新しい実行サイクルが開始される場合、OVERLAP ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

NOOVERLAP プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、NOOVERLAP コマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として NOOVERLAP を確立するオプションの修飾子です。ALWAYS を指定すると、OVERLAP プログラムコマンドによって無効になるまで、NOOVERLAP は有効のままになります。ALWAYS を指定しない場合、NOOVERLAP コマンドは次のロボットモーションだけに適用されます。

詳細

NOOVERLAP が有効で、ジョイント補間モーションまたは直線モーションの目的の変換ロケーションが制限停止を回避するために多回転軸を ±180° を超えて回転させる必要がある場合、プログラムエラーが発生して、そのモーションは実行されません。目的のロケーションがプレシジョンポイントとして指定されている場合、このチェックは行われません。

目的の変換ロケーションについては、多回転軸は通常、180° 未満のモーションしか必要としない方向に動くことで、新しいポジションに移動します。軸がより大きな変更をすることを強制する唯一の条件となるのは、SINGLE プログラムコマンドが発行された場合またはソフトウェアの制限停止を超過した場合です。

NOOVERLAP が有効な場合、SINGLE モードの設定は無視されます。

追加情報: 他のユーザプログラムエラー同様、NOOVERLAP チェックの結果として生じたエラー状態は、必要に応じて REACTE サブルーチンによって検出することができます。

関連キーワード

OVERLAP

SELECT (プログラムコマンド)

SELECT (実数値関数)

NULL

eV+ システムに対して、連続経路動作の末尾で位置誤差がヌル処理されるのを待機するように命令します。

構文

NULL *ALWAYS*

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボット動作しか影響を受けません。

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、NULL ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

NULL プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、NULL コマンドを実行するとエラーになります。

このキーワードの実行は、エミュレーションモードで使用した場合とオペレーションの違いがありません。エミュレーションモードにおけるロボットの軌道は現在のターゲット位置と同じとみなされます。そのため、実際のロボットを使用した場合と異なり、そのオペレーションが干渉したときに位置誤差は生じません。

パラメータ

ALWAYS	デフォルト条件として NULL を確立するオプションの修飾子です。ALWAYS を指定すると、NONULL プログラムコマンドによって無効になるまで、NULL は有効のままになります。ALWAYS を指定しない場合、NULL コマンドは次のロボット動作だけに適用されます。

詳細

NULL が有効であり、ロボット動作に中断が発生した場合、 eV+ はすべての動作する関節が指定位置に到達したことをサーボが通知するのを待ってから、次の動作を開始します。位置精度は COARSE または FINE プログラムコマンドによって決定します。

例

次の例では、位置「point1」に対して一度だけ動作するNULL 操作を発行します。

```
NULL
MOVE point1
```

関連キーワード

COARSE

CONFIG

DELAY.IN.TOL

FINE

NONULL

SELECT（プログラムコマンド）

SELECT（実数値関数）

SELECT(モニタコマンド)

OVERLAP

次のモーションまたは後続するすべてのモーションに対して、NOOVERLAP 制限エラーチェックを無効にします。

構文

OVERLAP *ALWAYS*

使用上の留意事項

OVERLAP は以下のロボットおよび該当するジョイントに適用されます。

- 6 軸ロボットのジョイント 1、4 および最終ジョイント
- スカラロボットのジョイント 4
- 並行ロボットのツール方向

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、OVERLAP ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

OVERLAP プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによってアタッチされていない限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、OVERLAP コマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として OVERLAP を確立するオプションの修飾子です。ALWAYS を指定すると、NOOVERLAP プログラムコマンドによって無効になるまで、OVERLAP は有効のままになります。ALWAYS を指定しない場合、OVERLAP コマンドは次のロボットモーションだけに適用されます。

詳細

OVERLAP が有効な場合、SINGLE プログラムコマンドの設定がロボットモーションに影響を与えます。

OVERLAP が有効で、ジョイント補間モーションまたは直線モーションの目的の変換ロケーションが制限停止を回避するために多回転軸を ±180° を超えて回転させる必要がある場合、そのモーションはプログラムエラーを発生することなく実行されます。

OVERLAP は NOOVERLAP の制限エラーチェックを無効にします。プログラムの実行が開始される場合、および新しい実行サイクルが開始される場合、OVERLAP 設定が適用されます。

関連キーワード

NOOVERLAP

SELECT (プログラムコマンド)

SELECT (実数値関数)

PACK

（128 文字の）文字列変数の配列の中または（配列ではない）文字列変数の中の部分文字列を置換します。

構文

PACK string_array[index]**, first_char, num_chars = string**

PACK string_var, first_char, num_chars = string

パラメータ

string_array	等号の右側にある部分文字列によって変更される文字列配列変数です。文字列配列内の各要素は 128 文字長であると仮定されます（下記参照）。
index	考慮する最初の配列要素を特定するオプションの整数値です。first_char の値はこの index が指定する要素に対して解釈されます。index を指定していない場合、要素 0 が仮定されます。
string_var	等号の右側にある部分文字列によって変更される文字列変数です。
first_char	文字列配列中の部分文字列の最初の文字の位置を指定する実数式です。値 1 は指定した文字列配列要素の最初の文字に対応します。これらの値は、0 よりも大きく指定する必要があります。 first_char の値は、128 よりも大きくすることができます。この場合、アクセスされる配列要素はこの関数呼び出しで指定される要素に続きます。例えば、値 130 は index で指定される配列要素に続く配列要素の 2 番目の文字に対応します
num_chars	この文字列から配列にコピーされる文字数を指定する実数式です。この値は 0 ～ 128 の範囲にすることができます。
string	部分文字列を抽出する文字列変数、定数または式です。この文字列は num_chars 以上の長さである必要があります。

詳細

このコマンドは文字列配列内または文字列変数内で部分文字列を置換します。文字列配列の変更中は、その部分文字列がその文字列配列の 2 つの要素にまたがることが許されます。例えば、最初の文字が要素 [3] の 127 番目の文字を置換する 10 文字の部分文字列は要素 [3] の最後の 2 文字のおよび要素 [4] の最初の 8 文字を置き換えます。

変更する配列要素を指定していない場合、要素が作成され、指定された部分文字列の始点まで ASCII NUL 文字（^H00）が入力されます。同様に、変更される配列要素が短すぎる場合、指定された部分文字列の始点まで ASCII NUL 文字が詰め込まれます。

文字列配列に効率的にアクセスするため、このコマンドでは、その配列の始点からアクセスされた要素の前の要素までのすべての配列要素が定義済みで 128 文字長であると仮定されます。多次元配列の場合、部分文字列の位置決めのために、右端の配列インデックスだけが漸増します。例えば、要素 [2,3] の次は要素 [2,4] になります。

文字列変数を変更する場合、その置換は個々の配列要素に対するものと同様にして実行されます。そのオペレーションによって文字列が 128 文字を超えると、エラーが生じます。

例

以下の例では、文字列配列「$list[ ]」の 11 文字を置換します。この置換は配列要素「$list[3]」で始まるように指定されています。しかし、置換される最初の文字は 130 番なので、この 11 文字から成る部分文字列は「$list[4]」の 2~12 番目の文字を置換します。

```
PACK $list[3], 130, 11 = $string
```

関連キーワード

$MID

$UNPACK

PANIC

外部 E-STOP ボタンを押した場合をシミュレーションし、すべてのロボットを即座に停止しますが、HIGH POWER をオフにしません。

構文

PANIC robot_num

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値と解釈されるオプションの実数値、変数、または式です。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

使用上の留意事項

robot_num パラメータでロボットを指定していない場合、すべてのロボットが停止します。

このコマンドはロボットが無いシステムには効果がありません。

詳細

このコマンドは次のアクションを実行します。

- ロボットの動作を即座に停止する
- ロボットが割り付けられていて REACTE が実行されていない場合に、ロボット制御プログラムの実行を停止して、プログラムのエラー処理を可能にします。

ペンダントの非常停止ボタンを押した場合と異なり、PANIC 操作を処理した後も高電圧は通電したままになります。

関連キーワード

ABORT（モニタコマンド）

ABORT（プログラムコマンド）

ESTOP（プログラムコマンド）

ESTOP(モニタコマンド)

PANIC（モニタコマンド）

PARAMETER

システムパラメータの値を設定します。

構文

PARAMETER parameter_name = value

PARAMETER parameter_name[index] **= value**

使用上の留意事項

指定したシステムパラメータがインデックス修飾子を受け入れ、そのインデックスが 0 であるか括弧付きまたは括弧なしで省略されている場合、そのパラメータ配列のすべての要素に与えられた値が割り当てられます。

パラメータ

parameter_name	値を変更されるパラメータの名称です。
index	あるindexによって修飾可能なパラメータに対して、対象となる特定のパラメータ要素を指定する、オプションの実数値、変数または式です。
value	システムパラメータに対して割り当てられる値を定義する実数値、変数または式です。

詳細

PARAMETER プログラムコマンドは与えられたシステムパラメータを右側の値に設定します。パラメータの名称は一意に特定する最小長さに短縮することができます。

> **注:** システムパラメータの値を設定するために通常の割り当てステートメントを使用することはできません。

システムパラメータに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

次の例では、BELT.MODE パラメータのビット 1 とビット 3 を ON に設定します（マスク値 1 + 4）。

```
PARAMETER BELT.MODE = 5
```

関連キーワード

BELT.MODE

NOT.CALIBRATED

PARAMETER（モニタコマンド）

PAUSE

実行プログラムを停止しますがプログラムの再開は可能にします。

構文

PAUSE

使用上の留意事項

HALT や STOP とは異なり、PAUSE はディスク論理ユニット上で FCLOSE または DETACH オペレーションを強制するものではありません。プログラムに開いているファイルがあり、現在の実行プログラムを継続しないことを決断した場合、該当するタスク番号を持つ KILL プログラムコマンドを発行して、すべてのファイルを閉じ、すべての論理ユニットのデタッチする必要があります。

詳細

PAUSE プログラムコマンドは中断を発生させ、アプリケーションプログラムの実行を停止しメッセージ（PAUSED）を表示します。該当するタスク番号を持つ PROCEED モニタコマンドを発行することで、実行をその後継続することが可能です。

プログラムをデバッグするとき、PAUSE コマンドを挿入してプログラムを一時停止し、変数の値を確認することができます。

> **注:** PAUSE コマンドの処理時に進行中のロボットモーションは、正常に完了します。

関連キーワード

HALT

KILL(monitor command)

KILL (program command)

PROCEED

STOP

PDNT.CLEAR

該当する場合に、T20 ペンダントの現在の告知ウィンドウまたはカスタムメッセージウィンドウをクリアして T20 ペンダントをホーム画面に戻します。

構文

PDNT.CLEAR

使用上の留意事項

対象となるペンダントは現在選択中のロボットに接続されているものになります。

T20 ペンダントのプログラミングについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

次の例では、現在選択されているロボットに接続している T20 ペンダントの画面をクリアします。

```
PDNT.CLEAR
```

関連キーワード

PDNT.NOTIFY

PDNT.WRITE

PDNT.NOTIFY

ペンダント告知を作成します。

構文

PDNT.NOTIFY $title, $msg

使用上の留意事項

このオペレーションを使用する前に ATTACH コマンドを使用してペンダントを割り付ける必要はありません。

対象となるペンダントは現在選択中のロボットに接続されているものになります。

T20 ペンダントのプログラミングについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

$title	ペンダント告知のタイトルを含むオプションの文字列定数、変数、式です。
$msg	ペンダント告知のメッセージを含むオプションの文字列定数、変数、式です。

詳細

PDNT.NOTIFY を使用することで T20 ペンダントの画面にシンプルな告知ボックスを作成します。この告知はペンダントで [**OK**] または [**Cancel**] ボタンを押すか、PDNT.CLEAR オペレーションを使用することでクリアできます。

例

次の例では、現在選択されているロボットに接続している T20 ペンダントに、タイトルとメッセージを持つ告知ボックスを作ります。

```
PDNT.NOTIFY “Manual Mode”, “To enable power, press and
hold the enable switch.”
```

関連キーワード

PDNT.CLEAR

PDNT.WRITE

PDNT.WRITE

ペンダントのカスタムメッセージ画面を設定します。

構文

PDNT.WRITE (msgsize) $title, $msg, $f1, $f2, $f3, $f4

使用上の留意事項

このオペレーションを使用する前に ATTACH コマンドを使用してペンダントを割り付ける必要はありません。

対象となるペンダントは現在選択中のロボットに接続されているものになります。

T20 ペンダントのプログラミングについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

msgsize	$msg の配列サイズを示すオプションの実数値、変数または式です。
$title	ペンダントのカスタムメッセージ画面のタイトルを含むオプションの文字列定数、変数または式です。
$msg	ペンダントのカスタムメッセージ画面の本文を含むオプションの文字列定数、変数または式です。これは html タグを受け入れて html フォーマットのテキストボックスを作成することができます。$msg が配列であり、かつmsgsize > 1 である場合、配列のすべての要素を連結します。
$f1	カスタムメッセージ画面の [F1] キーのラベルを含むオプションの文字列定数、変数または式です。
$f2	カスタムメッセージ画面の [F2] キーのラベルを含むオプションの文字列定数、変数または式です。
$f3	カスタムメッセージ画面の [F3] キーのラベルを含むオプションの文字列定数、変数または式です。
$f4	カスタムメッセージ画面の [F4] キーのラベルを含むオプションの文字列定数、変数または式です。

詳細

PDNT.WRITE は T20 ペンダントの画面を設定するために使用されます。これを使用することで、eV+ を介してペンダントをプログラミングして、ユーザインターフェースを作成できます。この画面は、その後 PDNT.WRITE を使用して更新するか、PDNT.CLEAR を使用してクリアすることができます。この画面が表示されている間、すべての緑色のキーおよび [Select Robot] は有効であり、ロボットを常にジョグ作動させることができます。その他のすべてのキーは無効です。

例

次の例では、現在選択されているロボットに接続している T20 ペンダントのタイトル、メッセージ、ファンクションキーに対して指定されたテキストを持つ画面を作ります。

```
$p.title = "Operator Control"

$p.msg[0] = "Select Options from buttons below"

$p.f[1] = "Apps"

$p.f[2] = "Status"

$p.f[3] = ""

$p.f[4] = ""

PDNT.WRITE $p.title, $p.msg[], $p.f[1], $p.f[2], $p.f[3], $p.f[4]
```

関連キーワード

KEYMODE

PENDANT

PDNT.CLEAR

PDNT.NOTIFY

PENDANT

手動制御ペンダントからの入力を返します。

構文

PENDANT (select) key

使用上の留意事項

この関数を使用する前に ATTACH コマンドを使用してペンダントを割り付ける必要があります。

T20 ペンダントのプログラミングについては、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

パラメータ

select 返されるペンダント情報の種類を選択する実数値式です。

key select によって決まるペンダントからの戻り値です。

詳細

戻り値は選択したパラメータに応じて次のように変わります。

select> 0

オペレーションがが呼び出された時点における所定のキー番号を持つキーの実際の状態を反映した値を即座に返します。キーの状態はそのキーのモード設定によって異なります。キーのモード設定については KEYMODE プログラムコマンドを参照してください。ペンダントが接続されている場合に限って、戻り値は有効です。オペレーションのこのモードでは、ペンダントの論理ユニットを割り付ける必要はありません。

キーがキーボードモードにある場合、値 ON（-1）はそのキーが押されていることを意味します。値 OFF（0）はキーが押されていないことを意味します。

キーがレベルモードにある場合、値 ON（-1）は、ペンダントが接続されていて、そのキーが押されていることを意味します。値 OFF（0）はキーが押されていないことを意味します。

キーがトグルモードにある場合、値 ON（-1）はそのキーがオンであることを、値 OFF（0）はキーがオフであることを意味します。

> **注:** select が 36（SLOW キーを表します）の場合、戻り値は、キーの キーのLED によって示されている、スローモードの現在の状態を示します。SLOW キーは常にトグルモードであり、KEYMODE コマンドの影響を受けません。

select = 0

次に押されたキーボードモードのキーのキー番号を返します。キーボードモードのキーが押されるまでプログラムの実行が中断します。このモードでキーをプログラ

ミングしていない場合、エラーが発生します。オペレーションのこのモードでは、ペンダントの論理ユニットを割り付ける必要があります。

select = -2

スピードラベルからの現在値を、0～100（10 進数）の範囲で返します。ペンダントジョグモードが COMP の場合、モニタ速度を返します。その他のジョグモードではジョグ速度を返します。オペレーションのこのモードでは、ペンダントの論理ユニットを割り付ける必要はありません。

select = -3

ペンダント上で有効な現在の表示画面を返します。これは、書き込みをする前に手動制御の状態を確認をする場合などに使用することができます。オペレーションのこのモードでは、ペンダントの論理ユニットを割り付ける必要はありません。

表示モードは次のように解釈されます。

表示モード	解釈
1	ホーム画面
2	その他の画面
3	エラー画面
4	USER（カスタム）画面

select = -4

手動制御ソフトウェアのバージョン番号を返します。これは実数値関数 ID(1,2) の戻り値と同じです。ペンダントがシステムに接続されていない場合、値 -1 を返します。オペレーションのこのモードでは、ペンダントの論理ユニットを割り付ける必要はありません。

例

次の例では手動制御ソフトキーをキーボードモードに設定し、そのどれか一つが押されるまで待機します。

```
ATTACH (1)
KEYMODE 1,5 = 0
key = PENDANT(0)
TYPE "Soft key #", key, " pressed"
DETACH (1)
```

次の例では、DONE キーをレベルモードに設定し、キーが押されるまで待機します。

```
ATTACH (1)
KEYMODE 8 = 2
WAIT PENDANT(8)
DETACH (1)
```

関連キーワード

ATTACH

KEYMODE

PROCEED

アプリケーションプログラムの実行を再開します。

構文

PROCEED task

使用上の留意事項

実行プログラムが正常に終了した場合または HALTプログラムコマンドによって停止した場合、プログラムを再開することはできません。

パラメータ

task	実行するプログラムタスクを指定する、整数として解釈される実数値、変数、または式です。タスク番号を指定していない場合、タスク番号 0 とみなされます。

詳細

PROCEED プログラムコマンドは、PAUSE プログラムコマンド、ABORTプログラムコマンド、ブレークポイント、シングルステップ実行またはランタイムエラーにより実行停止したステップの次のステップから、指定したプログラムタスクの実行を再開します。

中断されたプログラムの実行を継続する他に、このコマンドを使用して PRIME モニタコマンドで実行されるように作成されたプログラムの実行を開始することができます。

指定されたタスクが実行中で、そのプログラムが WAIT または WAIT.EVENT ステートメントにある（例えば、外部信号条件が満たされるのを待っている）場合、PROCEED コマンドを発行すると、その WAIT またはWAIT.EVENTをスキップします。

指定されたタスクが実行中で、プログラムが WAIT または WAIT.EVENT ステートメントにない場合、このコマンドには効果がありません。

PROCEED は RETRY コマンドとは異なります。プログラムステートメントがエラーを生成した場合、RETRY コマンドはそのステートメントを再実行しようとします。PROCEED コマンドは次のプログラムステートメントで実行を再開させます。プログラム停止時にロボット動作が進行中であった場合、RETRY ではその動作を完了しようとしますが、PROCEED では次の動作に進みます。

例

次の例では PAUSE またはブレークポイント操作によって停止している場合、タスク 2 の実行を再開します。

```
IF (TASK(1,2)==5)THEN
    PROCEED 2
END
```

関連キーワード

ABORT

ABORT（プログラムコマンド）

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

PROCEED

RETRY(モニタコマンド)

RETRY （プログラムコマンド）

STATUS

SSTEP

XSTEP

.PROGRAM

実行する際にプログラムに渡される引数を定義します。

構文

.PROGRAM program_name(argument_list) ;comment

使用上の留意事項

このコマンドは新しいプログラムが編集された際に eV+ エディタによって自動的に挿入されます。

この特別なコマンドはすべてのプログラムの最初の行にある必要があり、プログラムから省くことはできません。

> **追加情報:** 引数や変数の V+ プログラムとの間での引き渡しに関する詳細については、『eV+ 3 ユーザーズマニュアル (SBCE-486)』を参照してください。

パラメータ

program_name	この命令が含まれているプログラム名です。
argument_list	オプションの変数名をカンマで区切ったリストです。それぞれの変数は eV+ で使用可能な 1 つのデータタイプ（ベルト、プレシジョンポイント、実数値、文字列、変換）にできます。それぞれの変数は、単純な変数またはそのすべてのインデックスを空欄にした配列にすることができます。
;comment	ディスクファイルからプログラムが読み込まれた際と DIRECTORY モニタコマンドが処理された際に表示されるオプションコマンドです。コメントが含まれない場合、セミコロン [;] を省く必要ができます。

詳細

新しいプログラムを編集すると、eV+ エディタが自動的に .PROGRAM ラインを自動的に入力します。また、このラインを削除することやプログラム名を変更することもできなくします。このラインを編集して引数リストを追加、削除または変更することができます。プログラム名を変更するためには RENAME モニタコマンドを使用する必要があります。

引数リスト内の変数は、命名されたプログラムの自動変数とみなされます。詳細については AUTO キーワードを参照してください。

プログラムの実行が始まると（例えば、EXECUTE キーワードまたはCALL プログラムコマンドにより）、.PROGRAM コマンド内の引数は EXECUTE または CALL キーワード内の引数に関連付けられます。この関連付けにより、プログラムと呼び出し元の間で値を渡すことができます。

プログラム引数が呼び出しプログラムから値を受け取る方法、およびそれらの値を呼び出しプログラムに返す方法については、CALL プログラムコマンドの項を参照

してください。そのプログラムが実行される場合、省略される任意のプログラム引数には次のルールが適用されます。

- 省略された場合、プログラム内で実数型スカラーパラメータに値を割り当てることができます。
- 省略された場合、プログラム内で位置、文字列、およびベルト（スカラーまたは配列）パラメータ、および実数型配列パラメータを値として割り当てることはできません。AUTO 変数を使用すると、以下の例に示すように、この制限を回避することができます。未定義パラメータをプログラム引数として渡し、値として割り当てることができます。

  注: プログラムにおいてある値を省略された変数の一つに割り当てると、*Undefined Value*（未定義値）のエラーになります。その場合、エラーは割り当てステートメントの左側にある変数を照会しています。

- 未定義または省略されたパラメータがその後の CALL プログラムコマンドを介して他のプログラムに渡され、変数タイプが多義的である場合（タイプが実数値または位置である場合など）、そのパラメータは実数値であると仮定されます。
- 省略された配列パラメータの要素を、その後の CALL プログラムコマンドにおいて参照によって渡すことはできません。

DEFINED 実数型関数をプログラム内で使用して、プログラムパラメータが定義済み（引数として、また前に値を割り当てた引数として渡されたことの両方を意味します）であることを確認します。以下の例では、未定義または省略されたパラメータに対応するためのプログラム作成方法について示します。

.PROGRAM ラインにはコメントを追加することができ、このコメントはプログラムがディスクからまたは DIRECTORY モニタコマンドによって読み込まれる際に表示されます。

例

次の例は、引数を渡さないことを期待するプログラムを定義しています。

```
.PROGRAM get()
```

次の例は文字列型引数および位置または実数型引数を期待するプログラムを定義しています。2 番目の引数のタイプはプログラム内での使われ方によって異なります。

```
.PROGRAM test($n, dx)
```

未定義または省略されたパラメータを処理するためのプログラム作成方法について以下の例で示します。この例では、実数値パラメータと文字列パラメータを持つプログラムの例の一部を示します。

```
.PROGRAM example(real, $string)
        AUTO $internal.var
        IF NOT DEFINED(real) THEN
            real = 1
        END
        IF DEFINED($string) THEN
            $internal.var = $string
        ELSE
            $internal.var = "default"
```

```
        END
.END
```

関連キーワード

CALL

CALLS

EXECUTE（モニタコマンド）

EXECUTE（プログラムコマンド）

PRIME

SSTEP

XSTEP

PROMPT

モニタウィンドウに文字列を表示して、オペレータの入力を待ちます。

構文

PROMPT output_string**, variable_list**

使用上の留意事項

PROMPT キーワードを モニタコマンドプログラム と共に使用しないでください。またモニタウィンドウへのアクセスができない時も使用しないでください。

variable_list パラメータに対して外部配列変数を使用しないでください。

パラメータ

output_string	モニタウィンドウに出力されるオプションの文字列式です。そのカーソルは文字列の末尾に残されます。
variable_list	データを受け取る実数型変数のリストまたは単独の文字列変数です。 **注:** 外部配列変数はサポートされていません。

詳細

出力文字列のテキストをモニタウィンドウに表示し、リターンキーの押下により終了する一行の入力を待ちます。

入力行は以下に示す 2 つの方法のいずれかで処理されます。

1. 実数型変数リストが変数リストとして指定されている場合、行にはスペース文字および／またはカンマで区切られた数値のリストが含まれると仮定されます。それぞれの数値はテキストから内部表現に変換され、その値はその変数リストに含まれる次の変数に保存されます。指定した変数の数よりも読み取った値の数の方が多い場合、余分な値は無視されます。読み取った値の方が少ない場合、残っている変数には 0 が設定されます。数値ではない値を読み取った場合、エラーが生じてプログラムの実行は停止します。混乱を避け、エラーの可能性を減少させるために、個々の PROMPT コマンドが 1 つの値だけを要求するようにすべきです。
2. 変数リストとして 1 つの文字列変数を指定する場合、その入力行全体が文字列変数に保存されます。その後その文字列を、プログラムによって適切に処理する必要があります。

リターンキーを押すか CTRL + C を押すと、空の行が読み取られます。その場合、すべての実数型変数は 0 に設定されるか、文字列変数に空の文字列が割り当てられます。

CTRL + Z を押すとファイル末尾のエラー状態になります。REACTE コマンドが有効でない場合、プログラムの実行は終了し、エラーメッセージが表示されます。そのため、CTRL + Z は PROMPT においてプログラムを停止する有用な方法になります。

例

以下の例では、次のステートメントを考察します。

```
PROMPT "Enter the number of parts: ", part.count
```

以下のメッセージがモニタウィンドウに表示されます。

```
Enter the number of parts(部品数を入力してください):
```

数値を入力してリターンキーを押すと、変数「part.count」は入力した数値と同じ値に設定され、プログラムの実行が再開します。

以下の例では、次のステートメントを考察します。

```
PROMPT "Enter the number of parts: ", $input
```

数値入力として有効ではない文字を入力しても、eV+ はエラーメッセージを出力しません。このアプリケーションプログラムはさまざまな文字列関数を使用して入力文字列から数値を抽出します。

端末への文字列出力にフォーマット仕様（行をスキップするための /Cn など）を含める場合、以下に示すように $ENCODE 関数または TYPE コマンドのどちらかを使用することができます。

```
PROMPT $ENCODE(/B,/C1,/X10)+"Enter the number of parts:
", $input
```

上記のステートメントでは、端末が鳴り、1 行下がり、スペースが 10 個入り、文字列を出力し、入力を待機します。output_string パラメータ全体が単一の文字列式でなければならないので、$ENCODE 関数と参照される文字列の間には + 記号を用いる必要があります。

以下のステートメントは上記の例と同等です。

```
TYPE /B, /C1, /X10, /S
PROMPT "Enter the number of parts: ", $input


TYPE $ENCODE(/B,/C1,/X10)+"Enter the number of parts: ",
/S
PROMPT , $input
```

プロンプト文字列出力を 1 行にして、カーソルをその行に残すためには、図示するように TYPE コマンドには /S を含める必要があります。

関連キーワード

GETC

READ

TYPE

REACT

指定したデジタル信号の連続モニタリングを開始し、信号が変化するとサブルーチンを自動的に引き起こします。

構文

REACT signal_num, program, priority

使用上の留意事項

REACT プログラムコマンドは任意のプログラムタスクが実行できます。それぞれのタスクには個別の REACT 定義を持たせることができます。

REACTプログラムコマンドでは、以下に示す信号が使用可能です。

信号タイプ	範囲	詳細
ディジタル入力	1001 to 1999	信号は、システムに物理的に存在する場合のみ可用できます。
ソフトウェア	2001 to 2999	すべての信号は可用です。
ホスト	4001 to 4999	

応答は信号レベルではなく信号の遷移によってトリガされます。信号の OFF から ON への遷移をモニタしていて、REACT コマンドを実行したときにその信号がすでに ON である場合、その信号が OFF になってから再び ON になるまで応答は発生しません。

遷移を確実に検出するためには、信号がその状態を 18 ms 以上保つ必要があります。

> **注:** ソフトウェア信号を使用して応答をトリガしている場合、引数を持たない WAIT プログラムコマンドを必要に応じて使用することで、必要な時間にわたって信号の状態が一定になるようにしてください。

要求された信号モニタリングは、REACT コマンドを実行すると即座に有効になります。これは、そのプログラム内で REACT コマンドに先行する動作コマンドによって開始されたモーションに影響を与える可能性があります。

パラメータ

signal_num モニタリングする信号を示す実数式です。信号番号は上で述べた範囲内である必要があります。ソフトウェア信号は、あるプログラムタスクが他のタスクを中断するために使用することができます。signal_num がプラスである場合、eV+ は OFF から ON の遷移に応答します。signal_num がマイナスである場合、eV+ は ON から OFF の遷移に応答します。

program 信号が正しく遷移した場合に呼び出されるサブルーチン名です。

priority	次に示すようにして、この応答の相対的重要度を示す、オプションの実数値式です。この式の値は整数値と解釈され、1～127 の範囲になります。priority 値の詳細については、LOCK プログラムコマンドを参照してください。デフォルト値は 1 です。

詳細

指定した信号遷移を検出すると、eV+ はその時点でのプログラム優先度設定に照らして REACT コマンドで指定した優先度をチェックして応答します。実行を開始するとき、プログラムの優先度は常に 0 に設定されますが、LOCK プログラムコマンドを使用して変更することができます。REACT の優先度がプログラムの優先度よりも大きい場合、通常のプログラム実行シーケンスを中断して、CALL と同等のオペレーションが実行されます。

プログラムの優先度は REACT の優先度まで一時的に上がり、重要度が同等または以下の応答をロックアウトします。RETURN プログラムコマンドを応答サブルーチン内で実行する場合、プログラムの優先度は、その応答プログラムが呼び出される前の値で復元されます。

信号遷移が検出されたときに REACT の優先度がプログラムの優先度以下である場合、その応答はキューされてそのプログラムの優先度が低くなるまで発生しません。相対的な優先度に応じて、信号遷移が eV+ によって検出されてから応答プログラムが呼び出されるまでに相当な遅延が生じる可能性があります。

優先度によるロックアウトによって複数の応答が保留されている場合、ロックしている側の優先度が下がると、最も優先度の高い応答が最初に実行されます。複数の保留されている応答が同じ優先度である場合、高位の信号番号に関連するものが最初に処理されます。

プログラムを呼び出すサブルーチンの実行においては、RETURN プログラムコマンドに遭遇すると、実行される次のステートメントは、その応答プログラムが開始される直前に処理されたステートメントの次のものになります。応答プログラムによって中断したくないステートメントシーケンスがある場合、LOCK プログラムコマンドを使用してそのシーケンス中のプログラムの優先度を高めるようにしてください。

信号モニタリングは以下のいずれかが発生するまで継続します。

- その信号に対して IGNORE プログラムコマンドが実行される。
- IGNORE signal_num を自動的に実行する応答が発生する。
- 同じ信号を参照する REACT プログラムコマンドが実行される。REACT コマンドにおいて指定される信号がすでに以前の REACT または REACTI コマンドによってモニタされている場合、新しい REACT コマンドを実行すると古いコマンドはキャンセルされます。

例

次の例では、変数「test」の値によって指定される外部入力信号をモニタします。「test」の値の符号によって指定される信号遷移が発生すると、このステートメントにおいて優先度が指定されていないために、プログラムの優先度は 0 に低下すると同時に、プログラムは分岐してプログラム「delay」に進みます。このサブルーチンが呼び出されるとプログラムの優先度は 1（デフォルト）に上がり、プログラムが復帰すると、プログラム優先度は 0 に戻ります。

```
REACT test, delay
```

関連キーワード

IGNORE

LOCK

PRIORITY

REACTE

REACTI

SIG.INS

REACTE

現在のプログラムタスク実行中に発生するシステムメッセージのモニタリングを開始します。

構文

REACTE program_name

使用上の留意事項

REACTE プログラムコマンドは任意のプログラムタスクで実行できます。

それぞれのタスクには個別の REACTE ステートメントを付けることができます。タスクは他のタスクに生じたシステムメッセージを直接検出することはできませんが、グローバル変数またはソフトウェア信号を使用して他のタスクに伝達することができます。

> **重要:**予期せぬエラーが発生した際のエラールーチンを、事前にプログラミングする以外の目的でREACTE コマンドを使用することには、特に注意が必要であり、予測不能な挙動を生じる可能性があります。

その他の留意事項については以下を参照してください。

パラメータ

program_name	プログラムシステムメッセージが発生したときに呼び出される、オプションのプログラム名です。プログラムを指定しない場合、以前の REACTE オペレーションはキャンセルされ、保留中のシステムメッセージは廃棄されます。

詳細

REACTE プログラムコマンドの主な目的は、予期せぬシステムメッセージが発生したときに、あらかじめプログラミングされているエラールーチンを実行できるようにすることです。ロボットのハードウェアエラーが発生した場合、プログラムを実行して外部出力信号ラインを設定することで、外部機器を遮断することなどができます。

REACTE コマンドを実行した後にシステムメッセージが発生すると、通常のプログラムの実行を停止するのではなく、指定されたプログラムを呼び出します。このプログラムは、CALL プログラム命令に類似する機能を使用して呼び出されます。ERROR 関数をエラー処理プログラム内で使用すると、そのプログラムを呼び出す原因となったシステムメッセージを特定することができます。

REACTE コマンドを使用する際は以下の点に留意してください。

- エラー処理プログラムが呼び出されると、プログラムの優先度は 254 に上昇し、すべての応答プログラムをロックアウトします。
- システムが以前のシステムメッセージを処理しているときにエラーが発生するとプログラムタスクの実行が停止します。
- ユーザープログラムスタックには、さらにもう 1 つのサブルーチンを実行す

るために十分な使用可能メモリがある必要があります。エラー *Too many subroutine calls*を処理することはできません。詳細については STACK モニタコマンドを参照してください。

- エラー処理プログラムには RETURN ステートメントを含めることができます。このプログラムを実行すると、プログラムはそのシステムメッセージの原因となったコマンドを再実行しようとします。そのためシステムメッセージが引き続き発生すると、ループ状態になる可能性があります。
- エラー処理プログラムに入る前に、当該ロボット（論理ユニット番号 0）に DETACH コマンドを実行します。ロボットのプログラム制御を再開する前に、ATTACH コマンドをロボットに対して実行する必要があります。
- エラー処理プログラム内で STOP、HALT または PAUSE コマンドを実行する場合、そのエラー処理プログラムに引数のない REACTE コマンドが含まれる場合を除き、当初のシステムメッセージが出力されます。
- REACT および REACTI コマンドとは異なり、REACTE エラー処理プログラムの実行は、優先度の考慮によって延期されることがありません。

例

次の例では、システムメッセージのモニタリングを開始し、現在のプログラムタスク実施中にエラーが発生すると、プログラム「error.trap」を実行します。

```
REACTE error.trap
```

関連キーワード

ERROR

REACT

REACTI

RETURN

REACTI

指定したデジタル信号の継続的モニタリングを開始します。信号が正しく遷移すると現在のロボット動作を自動的に停止し、オプションとしてサブルーチンの呼び出しをトリガします。

構文

REACTI signal_num, program**,** priority

使用上の留意事項

ほとんどアプリケーションでは、ロボット制御プログラムだけに REACTI コマンドを使用するようにしてください。

REACTI が起動すると、その REACTI コマンドが実行された時点でどのロボットが選択されていたかに関わらず、停止するロボットは引き起こされた時点でそのタスクが選択しているロボットになります。

REACT コマンドプログラムのために記載されている留意事項を参照してください。

信号タイプ	範囲	詳細
ディジタル入力	1001 to 1999	信号は、システムに物理的に存在する場合のみ可用できます。
ソフトウェア	2001 to 2999	すべての信号は可用です。
ホスト	4001 to 4999	

パラメータ

REACTIプログラムコマンドでは、以下に示す信号が使用可能です。

signal_num モニタリングする信号を示す実数式です。信号番号は上で述べた範囲内である必要があります。これらのソフトウェア信号を 2 次プログラムで使用することで、ロボット制御プログラムを中断することができます。

signal_num がプラスである場合、eV+ は OFF から ON の遷移に応答します。信号がマイナスである場合、eV+ は OFF から ON の遷移に応答します。

program 信号が遷移した場合に呼び出される、オプションのサブルーチン名です。

priority 次に示すように、この応答の相対的重要度を示す、オプションの実数値式です。この式の値は整数値と解釈され、1～127 の範囲になります。引数を省略すると、デフォルトは 1 になります。priority 値の詳細については、LOCK プログラムコマンドを参照してください。

詳細

指定した信号遷移を検出すると、eV+ は現在のロボットモーションを即座に停止して応答します。プログラムを指定している場合、eV+ は REACT プログラムコマ

ンドに対する場合と同様に応答の処理を継続します。この処理に関する全解説については REACT プログラムコマンドの説明を参照してください。

ロボットを制御していないプログラムタスクがREACTI を使用している場合、応答サブルーチンが意図する効果により、ロボット制御プログラムが無効にならないように注意する必要があります。アプリケーションに信号をモニタするプログラムタスクとロボットを制御する他のプログラムタスクがある場合、応答を処理するための計画をする際は以下の点を考慮してください。

- 応答の際、処理中のロボットモーションは BRAKE コマンドが実行されたのと同じ様に停止しますが、ロボット制御プログラムの実行は直接的には影響を受けません。
- 応答サブルーチンを指定している場合、そのルーチンは応答をモニタしているタスクによって実行され、ロボットをを制御しているタスクによっては実行されません。

信号モニタリングは以下のいずれかが発生するまで継続します。

- その信号に対して IGNORE プログラムコマンドが実行される。
- IGNORE signal_num を自動的に実行する応答が発生する。
- 同じ信号を参照する REACTI コマンドが実行される。REACTI コマンドにおいて指定される信号がすでに以前の REACTI または REACT コマンドによってモニタされている場合、新しい REACTI コマンドを実行すると古いコマンドはキャンセルされます。

応答プログラムが呼び出されるまでロボットモーションを停止したくない場合、REACT プログラムコマンドを使用し、応答プログラムに BRAKE ステートメントを配置します。

例

次の例では、外部入力信号 1001 のモニタリングを開始します。この信号はマイナス値で指定されているので、信号が ON から OFF に変わると、直ちにロボットモーションは停止します。プログラムの優先度が 10 未満に低下すると、まだこのレベル以下に低下していなければ、プログラム「alarm」に至る分岐が発生します。

```
REACTI -1001, alarm, 10
```

関連キーワード

ERROR

REACT

REACTE

RETURN

READ

開かれたファイルまたはファイル指向ではない割り付けられたデバイスからレコードを読み取ります。ネットワークデバイスの場合、割り付けられた開いている TCP 接続から文字列を読み取ります。

構文

READ (lun, record_num, mode**) var_list**

使用上の留意事項

このコマンドで参照される論理ユニットは事前に割り付けられている必要があります。

ファイル指向デバイスの場合、ファイルを FOPEN_ キーワードを使用して開いておく必要があります。

var_list パラメータに対して外部配列変数を使用しないでください。

パラメータ

lun — アクセスするデバイスを定義する実数値式です。ユニット番号に関する説明については、ATTACH プログラムコマンドを参照してください。

record_num — ランダムアクセスモードで開かれるファイル指向デバイスに対して読み取るレコードを指定するオプションの実数値式です。ファイル指向ではないデバイスの場合やファイルにシーケンシャルアクセスをする場合、このパラメータを 0 にするか省略するようにしてください。レコードは 1～16,777,216 の番号になります。

TCP デバイスにサーバプログラムを使用してアクセスする場合、このパラメータはクライアントハンドル番号を返すオプションの実数型変数です。このハンドルを使用すると、複数クライアントのサーバにアクセスするクライアントを特定することができます。

mode — 読み取りオペレーションのモードを指定するオプションの実数値式です。このモードはモニタウィンドウ論理ユニットに対してのみ使用されます。その値は以下に示すように連続するビットフラグと解釈されます。mode の値が指定されていない場合、すべてのビットはクリアであると仮定されます。

- **Bit 1 (LSB, mask value = 1):** 待機 (0) 対非待機 (1)

  このビットが OFF の場合、プログラムの実行は読み取り作業が終了するまで中断されます。このビットが ON で要求されたデータを入手できない場合、プログラムの実行は即座に継続され、IOSTAT 関数がエラーコード *No data received*(受信データなし) (-526) を返します。

- **ビット 2 (マスク値 = 2)**: エコー (0) 対非エコー (1)

  このビットが OFF の場合、端末からの入力はソースにエコーバックされます。このビットが ON の場合、文字はソースにエ

コーバックされ ません。.

var_list データを受け取る実数型入力変数のリストまたは文字列変数のリストです。

> **注:** 外部配列変数はサポートされていません。

詳細

READ プログラムコマンドは、指定した論理ユニットからレコードを読み取る汎用のデータ入力オペレーションです。レコードには任意の文字リストを含めることができますが、512 文字長を超えることはできません。固定長レコードモードで開かれているファイルの場合、このコマンドでは、対応する FOPEN_ キーワードオペレーションで指定されている文字数を読み取るまで、文字の読み取りを継続します。

可変長レコードモードの場合、ほとんどのデバイスと同じように、このコマンドは最初のキャリッジリターン（CR）およびラインフィード（LF）文字シーケンスに遭遇するか、Ctrl + Z のキー入力をするまで、文字の読み取りを継続します。例えば、ディスクから可変長レコードモードでの読み取りを行う場合、CR または LF に遭遇するまで、すべての文字を受けます。

特殊文字 Ctrl + Z（10 進数の 26）はファイルの論理終点を示し、IOSTAT 関数によってエラーとして報告されます。この位置以降は文字を読み取ることができません。

モニタウィンドウおよびペンダントからの READ オペレーションは常に、可変長レコードモードであると仮定されます。以下に記載されている場合を除き、レコードは CR および LF で終わっていて、これらがレコードの一部として返されることはありません。これらのデバイスからの READ オペレーションは CR または LF を入力として受けるまで、完了しません。例えば、モニタウィンドウから READ オペレーションを実行する場合、RETURN キーを押すまですべての文字を受けます。

> **注:** モニタウィンドウから CR を受けると、eV+ は自動的に LF を追加します。
>
> レコード末尾の CR および LF 文字を受けたい場合は、READ プログラムコマンドの代わりに GETC関数を使用することができます。

mode パラメータの値でビット 1 が ON の場合、未完了の読み取りオペレーションがプログラムを待機させることはなく、エラー *No data received*（受信データなし）（エラーコード -526）を伴って速やかにリターンします。次に、変数リストのデータを得るために、1 つの READ コマンドが完了するまでに別の READ コマンドを実行する必要があります。IOSTAT 関数を使用して、こういった READ オペレーションか完了したときを特定することができます。

読み取られたレコードは次のいずれかの方法で処理されます。

- var_list パラメータが実数型変数のリストである場合、そのレコードにはスペースおよび／またはカンマで区切られた数字のリストが含まれると仮定されます。それぞれの数値はテキストから内部表現に変換され、その値はその変数リストに含まれる次の変数に保存されます。指定した変数の数よりも読

み取った値の数の方が多い場合、余分な値は無視されます。読み取った値の方が少ない場合、残っている変数には 0 が設定されます。数値ではない値を読み取った場合、エラーが生じて実行プログラムは停止します（またはエラーに対する応答が発生します）。

- var_list パラメータが文字列変数のリストである場合、レコード全体がその文字列変数内に次のようにして保存されます。レコードの最初の 128 バイトは最初の文字列変数にコピーされます。レコードが 128 バイトを超える場合、2 番目の文字列変数には次の 128 バイトが格納されます。そのレコード全体が処理されるか、すべての文字列変数が格納されるまで、これを繰り返します。

  すべての文字列変数を満たす十分なデータがない場合、未使用の文字列変数は空の文字列（""）になります。指定した文字列変数の数に対してデータ量が大きすぎる場合、IOSTAT 関数がエラーを報告します。

  可変長レコードモードで READ オペレーションを実行すると、その文字列には、末尾の CR および LF までに至るすべての文字が含まれます。ただし CR および LF は含まれず、削除されます。

このコマンドの仕様におけるエラー（無効なユニットからの読み取りしようとするなど）はプログラムエラーを生じ、プログラムの実行を停止します。実際の読み取りオペレーションの実施に関連するエラー（ファイルの末尾やデバイス準備未完了など）は、プログラムの通常の運用において発生するおそれのあるものであるため、実行プログラムを停止することはありません。これら通常のエラーは、読み出しを実行した後に IOSTAT 関数を使用して検出可能です。

> **注:** 一般的に、IOSTAT 関数からの値を試験することで、それぞれのファイルオペレーションが問題なく完了していることを常にチェックするのはよい習慣です。

ネットワークデバイスにアクセスする場合、record_num パラメータによって、サーバが 1 つの論理ユニット上で複数のクライアントと通信することが可能になります。このパラメータは READ データを送信するクライアントを識別するために使用するハンドル番号を与えます。クライアントがサーバに接続するとハンドルが割り当てられ、クライアントが接続を切ると割り当てが解除されます。クライアントが接続および切断を行った時期を特定するためには、READ オペレーションの後に IOSTAT 関数を使用する必要があります。

TCP/IP 通信を使用した READ コマンドでは、入力文字列が一杯になるかバッファが空になるまでデータを読み取り、そこでコマンドがリターンします。TCP/IP を使用した READ オペレーションでは固定長レコードは使用することはできず、区切り文字が来てもオペレーションは終了しません。

例

以下の例では、ディスクからテキスト行を読み取り、そのレコードを文字列変数「$disk.input」に保存します。

```
READ (5) $disk.input
```

関連キーワード

ATTACH

FOPEN

FOPENA
FOPEND
FOPENR
FOPENW
FSEEK
GETC
IOSTAT
PROMPT

READY

ロボットをレディ位置に移動します。

構文

READY

使用上の留意事項

このコマンドを実行する前に、ロボットがレディ位置に移動している間に障害物と衝突しないことを確認してください。

ロボットをレディ位置に移動すると、ロボットは標準設定に変わります。

READY プログラムコマンドは、タスクがロボットに割り付けてあれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、READY コマンドを実行するとエラーになります。

詳細

コマンドの実行時にロボットがどこにいても READY プログラムコマンドは必ず成功します。

レディ位置にいるときの各ロボットのジョイント位置を次の表に示します。

ジョイント	eCobra 600	eCobra 800	Viper 650/850	iX3 565	iX4 600/850
1	-43.5°	-42.9°	0°	0°	0°
2	96.8°	93.4°	-90°	0°	0°
3	10.0 mm	10.0 mm	180°	0°	0°
4	53.8°	50.5°	0°	0°	0°
5	N/A	N/A	0°	N/A	N/A
6	N/A	N/A	0°	N/A	N/A

例

次の例では robot 1 を選択し、現在のタスクを割り付けてからレディ位置まで移動させます。

```
SELECT ROBOT=1
ATTACH
READY
```

関連キーワード

SELECT

SELECT（実数値関数）

SELECT（モニタコマンド）

RELEASE

利用可能な次のプログラムタスクの実行を可能にします。

構文

RELEASE

詳細

このコマンドは実行準備の完了している別のタスクに制御を解放します。

このコマンドは、他のタスクに実行機会を与える必要があるものの、次の軌道サイクルまでの遅延が望ましくない場合に、引数のない WAIT コマンドの代わりに使用することができます。

関連キーワード

WAIT

WAIT.EVENT

RESET

外部出力信号をすべて OFF にします。

構文

RESET

使用上の留意事項

このプログラムコマンドは信号番号 4001～4009 のホストI/O（外部）信号には効果がありません。

詳細

RESET プログラムコマンドは、プログラムの初期化処理の一部として、すべての外部出力信号を既知の状態にする場合に役立ちます。



危険：このコマンドを発行する前に、出力信号に接続されているすべてのデバイスが安全に OFF できることを確認してください。

例

次の例では、「reset.sig」が TRUE の場合にすべての外部出力信号を OFF にします。

```
IF reset.sig==TRUE
    RESET
END
```

関連キーワード

BITS

BITS (プログラムコマンド)

BITS

IO

RESET(モニタコマンド)

SIG

SIGNAL(モニタコマンド)

SIGNAL (プログラムコマンド)

RETRY

最後に中断されたプログラムコマンドを再度実行し、実行プログラムを継続します。

構文

RETRY task

使用上の留意事項

指定されたプログラムタスクの実行中は、RETRY プログラムコマンドを処理できません。

実行プログラムが正常に終了した場合または HALT プログラムコマンドによって停止した場合、プログラムを再開することはできません。

パラメータ

task	実行するプログラムタスクを指定する、整数として解釈される実数値、変数、または式です。タスク番号を指定していない場合、タスク番号 0 とみなされます。

詳細

RETRY プログラムコマンドは、PROCEED プログラムコマンド同様、指定したタスクの実行を再始動します。RETRY コマンドの後、どの地点から実行が再開されるかは、その実行が中断されたときの状況によって変わります。プログラムステップまたはロボット動作が完了前に中断された場合、RETRY コマンドを使用すると、中断されたオペレーションを完了してから、実行が通常通り継続します。そのため、中断されたステップまたはエラーの原因となったステップを再試行することができます。

プログラムステップやロボット動作を中断していない場合、RETRY コマンドの効果は PROCEED プログラムコマンドのものと同じになります。

> **注:** RETRY コマンドを使用して中断した動作を再開すると、すべての動作パラメータはその動作を中断する前に有効であった設定に戻されます。

例

次の例では、（タスクが実行中ではない、または HALT 操作で停止されていない場合）「task.num」でタスク番号が定義されたタスクを、中断されたところから再開します。

```
IF TASK (task.num)<>4 THEN
   RETRY task.num
END
```

関連キーワード

PROCEED（プログラムコマンド）

PROCEED（モニタコマンド）

SSTEP

STATUS

XSTEP

RETURN

現在のサブルーチンの実行を終了して、中断された実行プログラムを次のステップで再開します。

構文

RETURN

詳細

メインプログラム内の RETURN プログラムコマンドには、STOP プログラムコマンドと同じ効果があります。

実行プログラムがサブルーチンの最後のステップに到達すると、RETURN オペレーションが仮定されます。RETURN ステートメントを省略するのはよいプログラミング手法ではありません。RETURN ステートメントをそれぞれのサブルーチンの最終行に含めるようにしてください。

エラー応答サブルーチンにおいて、応答サブルーチンが呼び出された原因が、サーボの不同期エラーや PANIC ボタンを押したことではなく、プログラムエラーである場合、そのエラーの原因となったステートメントが再度実行されます。その際、そのエラーがすぐに再発する可能性があります。こういった状況を回避するためには、エラー応答ルーチンにおいて RETURNE プログラムコマンドを使用してください。

応答ルーチンから脱出するために RETURN コマンドを使用すると、プログラム応答の優先度は、その応答ルーチンが実行を開始する前の値に戻ってしまいます。

関連キーワード

CALL

CALLS

LOCK

REACT

REACTE

REACTI

RETURNE

RETURNE

エラー応答サブルーチンの実行を終了し、サブルーチンを呼び出したステートメントの次のステップにおいて最後に中断した実行プログラムを再開します。

構文

RETURNE

詳細

RETURNE プログラムコマンドは、REACTE キーワードによって呼び出されたエラー応答ルーチンで使用することを目的としています。

エラー応答ルーチンから脱出するために RETURNE コマンドを使用すると、プログラム応答の優先度は、その応答ルーチンが実行を開始する前の値に戻ってしまいます。

エラー応答サブルーチンにおいて RETURNE コマンドを実行すると、エラーが発生したときに実行していたステートメントの次のステートメントで実行を継続します。この場合、RETURN コマンドを使用すると、エラーを生じたステートメントを再度実行することになり、エラーが再発する可能性があります。

> **注:** eV+ の先読み処理機能のせいで、エラーの原因となったステートメントはエラーを登録した際に実行したものではない可能性があります。例えば、MOVE プログラムコマンドを処理する場合、ロボットが動き始めても、その動作中にいくつかの追加ステートメントが処理されている可能性があります。この先読み処理の後にエンベロープエラーまたは類似するエラーが発生した場合、その RETURNE オペレーションはエラーが発生したときのステートメントの処理に基づき、MOVE プログラム命令に基づくものにはなりません。

エラーに応答して実行されなかった プログラム内の RETURNE コマンドには RETURN オペレーションと同様の効果があります。RETURNE オペレーションにかかる時間は RETURN オペレーションよりも若干長くなっています。

関連キーワード

REACTE

RETURN

RIGHTY

次の動作に関するロボットの設定変更を要求して、スカラロボットの最初の 2 つのリンクが右アーム方向を使用するようにします。

構文

RIGHTY

使用上の留意事項

直線動作中は設定の変更をすることができません。

選択したロボットがRIGHTY設定をサポートしていない場合、ロボットはこの命令を無視します。

RIGHTY コマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。ロボットが割り付けられていない場合、このコマンドは効果がありません。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように構成されていない場合、RIGHTY コマンドを実行するとエラーになります。

以下の図は、スカラロボットの平面図におけるLEFTY/RIGHTY 設定を示したものです。

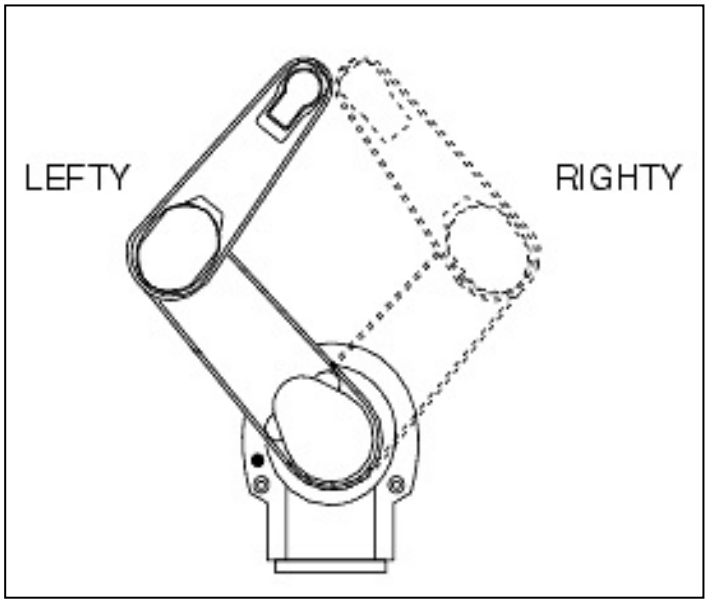


図3-16. LEFTY/RIGHTY 設定

例

次の例では、RIGHTY 設定で位置「point1」にロボットを移動します。

```
RIGHTY
MOVE point1
```

関連キーワード

CONFIG

LEFTY

SELECT（プログラムコマンド）

SELECT（実数値関数）

SELECT(モニタコマンド)

RUNSIG

呼び出しプログラムタスクの実行が継続している間は指定したデジタル信号を ON または OFF にします。

構文

RUNSIG signal_num

使用上の留意事項

それぞれのプログラムタスクに対する RUNSIG プログラムコマンドには、1 つの信号しか関連付けすることができません。

パラメータ

signal_num	制御されるデジタル出力信号の 1 つまたは内部ソフトウェア信号を指定する、オプションの実数式です。 この値がプラスである場合、プログラム実行中、信号は ON に設定されます。マイナス値の場合、プログラム実行中、信号は OFF に設定され、実行を停止すると、ON になります。 信号を指定しない場合、そのタスクに有効な RUNSIG はキャンセルされます。

詳細

RUNSIG プログラムコマンドは、このコマンドを実行すると即座に指定したデジタル信号を ON または OFF にします。呼び出しプログラムタスクの実行が停止するか、STOP プログラムコマンドが実行されると、信号の状態が保存されます。

このコマンドは通常、プログラム実行中にエラーが発生した場合、補助装置を停止する必要があるアプリケーションで使用されます。

RUNSIG コマンドは、それぞれのプログラムタスクに対して、一度に 1 つの信号しか有効にすることができません。プログラムが最初の RUNSIG オペレーションをキャンセルしてから 2 番目の RUNSIG オペレーションを開始しようとしない限り、エラー状態が発生します。

RUNSIG コマンドを実行した後にプログラムの実行が中断した場合、PROCEED または RETRY コマンドを発行すると、指定された信号は指定された状態に戻ります。SSTEP または XSTEP モニタコマンドが発行されると、呼び出されたコマンドの実行中、信号は指定された状態に戻ります。同様に、DO モニタコマンドを処理すると、対応するプログラムタスクの RUNSIG 信号が一時的に有効になります。EXECUTE キーワードは、指定したプログラムタスクに対する以前の RUNSIG オペレーションをキャンセルします。

例

次の例では、その値がプラスであると仮定される変数「run.signal」の値によって特定されるデジタル信号を ON にします。この信号は現在のプログラムを実行している間を通じて ON のままになります。実行が終了するとこの信号は OFF になります。

```
RUNSIG run.signal
```

関連キーワード

IO

RESET

SIG

SIG.INS

SIGNAL

SIGNAL（モニタコマンド）

SELECT

現在のタスクがアクセスするために、命名されたデバイスのユニットを選択します。

構文

SELECT device_type = unit

使用上の留意事項

SELECT プログラムコマンドは、同じタイプの複数のデバイスが eV+ システムに接続している場合に限って、使用するようにしてください。

SELECT プログラムコマンドは、そのコマンドが実行されているタスクだけに効果があります。

ステートメント SELECT ROBOT は、現在のタスクに割り付けられたロボットがない場合に限って実行することができます。ロボットが割り付けられているかいないか不確実である場合は、DETACH プログラムコマンドを実行してから SELECT コマンドを実行するようにしてください。

ロボットへの EtherCAT 接続が有効ではなく、COMM_STOP エラーが存在する場合、依然としてこれを選択することができます。SELECT 実数値関数を使用してロボットの状態を取得してから SELECT プログラムコマンドを発行することを推奨します。

パラメータ

device_type	選択するデバイスのタイプを特定する指定子です。唯一有効な device_type 指定子は、ROBOT です。
unit	選択するユニットを示す、整数値と解釈される実数値、変数または式です。使用できる値は、システムの設定によって異なります。

詳細

複数ロボットシステムで SELECT プログラムコマンドを使用すると、現在のタスクで通信しているロボットを選択します。SELECT モニタコマンドは、eV+ モニタがアクセスするロボットを指定します。SELECT プログラムコマンドは、動作コマンドを受け取り、ロボット関連情報（HERE 関数など）を返すロボットを指定します。

プログラムタスクが実行を開始するたびに、ロボット 1 を自動的に選択します。ロボットを選択すると、そのロボットに関する情報にアクセスできます。プログラムによってロボットを動かすためには、ロボットを選択して、ATTACH プログラムコマンドを使用して割り付ける必要があります。

例えば、SELECT コマンドを使用して ロボット 2 を選択すると、別の SELECT コマンドを発行するまで、現在のタスクが実行するすべての動作オペレーションは、そのロボットに対して命令されます。すべてのロボット関連関数はロボット 2 に関する情報を返します。

> **注:** タスク 0 を実行すると、プログラムの実行を開始する際に、ロボット 1 を自動的に選択して割り付けます。

任意のタスクにおいて選択しているロボットを変更するためには、ロボットがそのタスクによって割り付けられていてはいけません。複数のタスクで特定の1台のロボットを選択することはできますが、1 台のロボットを割り付けることができるのは 1 つのタスクだけです。ロボットがすでに別のタスクに割り付けられている場合、その ATTACH コマンドの mode パラメータに応じて ATTACH オペレーションは待機するかエラーを発生します。

unit パラメータに指定されたロボット番号が無効な場合、エラー「Invalid argument」（-407、無効な引数）が発生します。

unit パラメータに対して指定されたロボット番号は有効であるが、ロボットがシステム内で設定されていない場合、エラー「No robot connected to system」（-622、システムに接続しているロボットがありません）が発生します。

unit パラメータに対して指定されたロボット番号が有効で、ロボットがシステム内で設定されている場合、「Success」（成功しました）情報メッセージが発生します。

例

以下の例ではロボット 3 を選択して動かします。タスク 0 は デフォルトでロボット 1 に割り付けられているので、このプログラムは通常タスク 0 によって実行されません。

```
.PROGRAM test()
        SELECT ROBOT = 3
        ATTACH (0,1)
        IF IOSTAT(0) < 0 THEN
            TYPE /B, "Error attaching robot: ", $ERROR(IOSTAT(0))
            PAUSE
         END
         MOVE x
         MOVE y
         DETACH
.END
```

関連キーワード

ATTACH

OBSTACLE

SELECT

SET.EVENT

指定されたタスクに関連するイベントを設定します。

構文

SET.EVENT task, flag

パラメータ

task	設定されるイベントのタスクを指定する、整数と解釈されるオプションの実数値、変数または式です。有効範囲は 0～27 です。このパラメータを省略した場合、現在のタスクの番号が使用されます。
flag	使用しません。デフォルトは 1 です。

詳細

SET.EVENT プログラムコマンドは指定されたタスクに関連するイベントを設定します。例えば、あるタスクが WAIT.EVENT 1 ステートメントによって中断されている場合、そのタスクに対して SET.EVENT コマンドを実行すると、適切な次の時機にそのタスクの実行を再開します。

関連キーワード

CLEAR.EVENT

GET.EVENT

WAIT.EVENT

SET

等号の左側にあるロケーション変数の値を、右側のロケーション値に等しくなるように設定します。

構文

SET location_var = location_value

パラメータ

location_var	変換変数で終了する単一のロケーション変数または複合変換です。
location_value	等号の左側にあるロケーション変数と同じタイプのロケーション値であり、変数、関数、複合変換で定義されます。

詳細

等号の右側が定義されていない場合または同じタイプのロケーション表現（変換またはプレシジョンポイント）ではない場合、プログラムエラーが発生します。

等号の左側に複合変換を指定すると、右端の変換だけが定義されます。複合変換中の他の変換の中に未定義のものがあると、エラー状態が生じます。

等号の左側に変換変数を指定すると、右側には、変換、複合変換または変換関数を入れることができます。

例

次の例では、変換「pick」の値を「corner」のロケーションに「corner」を基準とした shift のロケーションを加えたものと等しくなるように設定します。

```
SET pick = corner:shift
```

次の例ではプレシジョンポイント「#place」の値をプレシジョンポイント「#post」の値を等しくなるように設定します。

```
SET #place = #post
```

次の例では、変換 part の値を、変換 pallet を基準とした現在のロボットのロケーションに設定します。

```
SET pallet:part = HERE
```

次の例では「loc1」の値を X = 550、Y = 450、Z = 750、y = 0、p = 180、r = 45 に設定します。

```
SET loc1 = TRANS(550, 450, 750, 0, 180, 45)
```

関連キーワード

HERE（プログラムコマンド）

HERE（モニタコマンド）

SETBELT

指定したベルト変数のエンコーダオフセットを式の値と等しくなるように設定します。

構文

SETBELT %belt_var = expression

使用上の留意事項

このコマンドを 実行するためには BELT システムスイッチを有効にする必要があります。

ロボットが指定したベルト変数に対して移動している間は、SETBELT プログラム命令を実行することはできません。

DEFBELT プログラムコマンドを使用して、参照するベルト変数を事前に定義する必要があります。

パラメータ

%belt_var 設定するエンコーダオフセットに関連するベルト変数名です。

expression 符号付の 24 ビットエンコーダオフセット値を指定する実数式です。

詳細

ベルト変数に関連するベルトの位置を計算する場合、eV+ はそのオフセット値を現在のベルト位置の値から引き、その差、16,777,216の剰余を使用します。

式の値は通常、-8,388,608～8,388,607の範囲の符号付きの数値です。この数値が範囲外の場合、値の16,777,216 の剰余を使用します。

SETBELT プログラムコマンドは通常、BELT 関数と一緒に有効ベルトポジションを 0 に設定するために使用されます。これは、ロボットがベルトに対してモーションシーケンスを実行するたびに、そのシーケンスの最初のモーションをする前に実行する必要があります。



警告: ロボットがベルトをトラッキングする必要が生じたらそのたびに SETBELT を実行して、有効なベルトトラッキングの間、BELT 関数が返す現在のベルトポジションと指定したベルト変数のベルトポジションの差が 8,388,607(^H7FFFFF) を超えないことを確実にすることが重要です。ベルトのトラッキング中に差がこの値を超えると、予期せぬロボットモーションが生じる可能性があります。

SETBELT プログラムコマンドを使用して、ロボットモーションを、外部信号またはビジョンシステムによってラッチされたエンコーダの値に同期させることができます。詳細については LATCHED 関数または DEVICE 関数を参照してください。

例

次の例では、デジタル信号を待って、ベルトポジションを 0 に設定します。これはベルトオフセットを現在のベルトポジショに等しくなるように設定することで実施

されます。その後、ロボットはベルト上に移動します。

```
WAIT sig(1001)
SETBELT %belt1 = BELT(%belt1)
MOVES %belt1:pickup
```

関連キーワード

BELT

DEFBELT

LATCHED

WINDOW

WINDOW(実数値関数)

SETDEVICE

デバイスを初期化するか、デバイスパラメータを設定します。実行されるオペレーションは参照されるデバイスによって異なります。

構文

SETDEVICE (type, unit, error**, command)** p1, p2, ...

使用上の留意事項

この構文には特定のデバイスタイプおよびコマンドだけに適用されるオプションのパラメータが含まれています。

パラメータ

type 参照されるデバイスのタイプを示すオプションの実数値、変数または式です。整数値と解釈されます。以下のタイプを使用できます。

- 0: ベルトエンコーダ
- 1: 今後の使用のための予備
- 2: フォースプロセッサ基板
- 3: ロボットデバイス(OMRON サーボのみ)
- 4: 画像センサ
- 5: 今後の使用のための予備

unit デバイスユニット番号を示す、整数値と解釈される実数値、変数または式です。この値は 0 から(max - 1) の範囲にある必要があります。ここで max は指定したタイプのデバイスの最大数です。指定したタイプのデバイスが 1 つしかない場合、この値を 0 にします。

error このコマンドが成功したか失敗したかを示す、標準システムエラー番号を受けるオプションの実数型変数です。このパラメータが省略されている場合、デバイスエラーが発生すると、プログラムの実行は停止されます。エラー変数を指定した場合、プログラムはエラーを検出するために明確なチェックを実行する必要があります。

command このコマンドで設定するデバイスコマンドまたはパラメータを指定する実数値、変数または式です。一部のコマンドは標準であり、すべてのデバイスが認識します。その他のコマンドは特定のデバイスタイプだけに適用されます。

p1, p2, ... オプションの実数値、変数または式であり、この値はコマンドのデータとしてデバイスに送られます。指定されるパラメータの数およびその意味はアクセスされている特定のデバイスタイプによって異なります。

詳細

SETDEVICE は外部デバイスを初期化する汎用コマンドです。ソフトウェアを初期化して、そのデバイスに関連付けられたさまざまなパラメータを設定することを可能にします。

以下に示すようにすべてのデバイスは 2 つの標準 SETDEVICE コマンドを認識します。

command = 0 **デバイスの初期化**
他の任意のコマンドでこのデバイスにアクセスする前に、このコマンド

を一度発行するようにしてください。通常、追加パラメータは不要ですが、一部のデバイスには追加パラメータを許容するものがあります。

command = 1　**デバイスのリセット**
このコマンドはデバイスをリセットします。通常、追加パラメータは不要ですが、一部のデバイスには追加パラメータを許容するものがあります。

関連キーワード

DEVICE

SIGNAL

外部デジタル出力信号または内部ソフトウェア信号を ON または OFF にします。

構文

SIGNAL signal_num, ..., signal_num

パラメータ

signal_num	デジタル出力信号、または内部信号の番号を評価する実数式です。プラスの値は ON にすることを意味します。マイナスの値は OFF にすることを意味します。SIGNAL コマンドは値 0 のパラメータを無視します。

詳細

signal_num パラメータの値は、どのデジタル信号または内部信号を操作するかを特定します。

インストールされ出力として設定されたデジタル信号だけが使用することができます。現在のデジタル I/O 設定をチェックするには IO モニタコマンドを使用します。信号 3001 および 3002 は現在のタスクで選択されているロボットを意味します。信号 3001 はハンド閉ソレノイドの状態です。信号 3002 はハンド開ソレノイドの状態です。

signal_num パラメータ値の符号が + である場合、信号は ON になります。signal_num パラメータ値の符号が - である場合、信号は OFF になります。

> **注:** すべての eV+ デジタル出力キーワードが 16 ms の eV+ サイクルを待機するわけではなく、すぐに ON になります。デジタル信号は eV+ オペレーティングシステムによって 16 ms 毎に確認されるので、システムが出力を読み取ることができるようになる前に、信号を ON および OFF にすることができます。

例

以下の例では、「reset」の値がプラスであると仮定することで、変数「reset」の値によって指定される外部出力信号を OFF にして外部出力信号 4 を ON にします。

```
SIGNAL -reset, 4
```

以下の例では外部出力信号 1 を OFF にし、外部出力信号 4 を ON にし、内部ソフトウェア信号 2010 を ON にします。

```
SIGNAL -1, 4, 2010
```

関連キーワード

BITS

BITS（プログラムコマンド）

BITS（実数値関数）

IO

NOOVERLAP

OVERLAP

RESET

RUNSIG

SIG

SIG.INS

SIGNAL

SINGLE

ロボットの手首のジョイントの回転を -180° ～+180° の範囲に限定します。

構文

SINGLE ALWAYS

使用上の留意事項

ALWAYS パラメータが指定されていない場合、次のロボットモーションしか影響を受けません。

SINGLE プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによってアタッチされていない限り、任意のプログラムタスクによって実行することができます。このプログラム命令はタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、SINGLE コマンドを実行するとエラーになります。

パラメータ

ALWAYS	デフォルト条件として SINGLE を確立するオプションの修飾子です。ALWAYSを指定すると、SINGLEは継続して有効なままになります。ALWAYS を指定しない場合、SINGLE コマンドは次のロボットモーションだけに適用されます。

詳細

変換で指定したロケーションまで移動する場合、ロボットは通常、所要の方向を達成するために必要となる最小距離だけ手首のジョイントを動かします。場合によっては、このモーションによって手首が限界の停止位置に近づくように移動し、その後の直線モーションによってこの停止位置に当たる可能性があります。

SINGLE プログラムコマンドを実行すると、その後のモーションで手首を可動範囲の中心付近に戻して、直線モーションがこのようにして失敗しないようにすることができます。

APPRO オペレーションにおいてロボットのプログラミング時点でそのポジションと方向が分からないオブジェクトをピックアップする場合、一般的に SINGLE コマンドが指定されます。オブジェクトを獲得した後は、手首のモーションを最小限に保つことができます。

NOOVERLAP が有効な場合、SINGLE 設定は無視されます。

関連キーワード

CONFIG

NOOVERLAP

OVERLAP

SELECT（プログラムコマンド）

SELECT（実数値関数）

SOLVE.ANGLES

現在のロボットに対して、指定した変換に相当するロボットジョイント位置を計算します。

構文

SOLVE.ANGLES o.jts[*o.idx* **], o.flags, error = trans, i.jts[** *i.idx* **],i.flags**

使用上の留意事項

このコマンドが実行する計算はリンクの寸法に基づくロボット幾何形状、軸数、ツールのオフセット、および基準フレームのオフセットの関数なので、異なる幾何パラメータを持つロボットでは結果が異なります。一般的なタイプが同じロボットであってもその寸法は若干異なる可能性があり、同じタイプの 2 つの異なるロボットシステムでこのプログラムを実行しても、若干異なる結果を返す可能性があります。

SOLVE.ANGLES プログラムコマンドは、このコマンドを実行するタスクが選択したロボットに関する情報を返します。

eV+ システムがロボットを制御するように設定されていない場合、このコマンドを実行しても、ロボットが存在しないので、エラーを生じません。しかし、返される情報に意味がなくなる可能性があります。

パラメータ

o.jts

計算したジョイント角度を返す実数配列です。配列の 1 番目の要素にはジョイント 1 の位置、2 番目の要素にはジョイント 2 の位置のように、各要素に対応するジョイントの位置が含まれます。回転ジョイントの場合は、ジョイント位置は ° になります。直動ジョイントの場合、ジョイントの位置は mm になります。

計算したジョイント位置がそのジョイントの可動域外である場合、i.jts[ ] で示されるその初期ジョイント位置に最も近い制限ストップを返します。

o.idx

ジョイント 1 の位置を受ける配列要素を特定する、整数値と解釈されるオプションの実数値、変数または式です。インデックスを指定しない場合、配列要素 0 にはジョイント 1 の位置が入ります。

o.flags

計算したジョイント位置に対応するロボット設定を示すビットフラグ値を受ける実数型変数です。このビットフラグは以下に示すように解釈されます。

ビットフラグ	説明
ビット 1（LSB） RIGHTY（マスク値 = 1）	このビットが ON の場合、その位置には右側設定のロボットがあります。それ以外の場合、その位置には左側設定のロボットがあります。

ビットフラグ	説明
ビット 2 BELOW （マスク値 = 2）	このビットが ON の場合、この位置には肩から手首の線よりも下に肘のある設定のロボットがあります。それ以外の場合、ロボットの肘は肩-手首ラインよりも上にあります。スカラロボットを使用する場合このビットは常に 0 になります。
ビット 3 FLIP （マスク値 = 4）	このビットが ON の場合、この位置には手首のピッチ軸がマイナスの角度になるように設定されたロボットがあります。それ以外の場合、ロボットの手首のピッチ角はプラスの値です。そのロボットがスカラロボットのように 3 軸手首を持たない場合、この値は常に 0 になります。

error 任意のジョイント位置が作業域外であると計算されているかどうか、または目標位置の XYZ 位置がロボットの作業エンベロープ外であるかどうかを示すビットフラグ値を受ける実数型変数です。このビットフラグは以下に示すように解釈されます。

ビットフラグ	説明		
ビット 1～12 ジョイント／モータ範囲外	以下に示すように、ON の場合、ジョイントまたはモータに対する計算値が制限ストップの範囲外であることが判明しています。		
	ビット	ジョイント／モータ番	マスク値

ビットフラグ	説明		
		号	
	1	1	^H1
	2	2	^H2
	3	3	^H4
	4	4	^H8
	5	5	^H10
	6	6	^H20
	7	7	^H40
	8	8	^H80
	9	9	^H100
	10	10	^H200
	11	11	^H400
	12	12	^H800
ビット 13 衝突 （マスク値 = ^H1000）	このビットが ON の場合、衝突が検出されています。		
ビット 14 近すぎる （マスク値 = ^H2000）	ロボットのコラムに近すぎるために目標位置の XYZ 位置に到達できません。		
ビット 15 遠すぎる （マスク値 = ^H4000）	ロボットから遠すぎるために目標位置の XYZ 位置に到達できません。		
ビット 16 ジョイント対モータ （マスク値 = ^H8000）	ON の場合モータを制限します。それ以外はジョイントを制限します。		

trans 対象のロボットロケーションを定義する変換変数、関数または複合変換です。

i.jts ロボットの始動位置を示すジョイント位置を含む実数配列です。これらの値は、多回転関節がジョイントの回転を最小限にするため、および計算したジョイント位置が範囲外である場合にどの制限ストップを返すかを決めるために、参照されます。

配列の最初の指定要素はジョイント1の位置を含んでいる必要があります。第2要素にはジョイント2の値などが含まれる必要があります。回転ジョイントの場合、ジョイントの位置は°で設定されます。直動ジョイントの場合、ジョイントの位置はmmで設定されます。

i.idx 関節1の位置の値を含む配列要素を特定する、整数値と解釈されるオプションの実数値、変数または式です。インデックスを指定しない場合、要素0には関節1の位置が入ります。

i.flags 実数値、変数または式であり、その値はロボットの初期設定、実施された設定の変更および特殊な運用モードを示すビットフラグと解釈されます。このビットフラグは以下に示すように解釈されます。

ビットフラグ	説明
ビット1:（LSB）RIGHTY（マスク値＝1）	このビットがONの場合、そのロボットは当初、右腕設定のロボットであると仮定されます。それ以外の場合、そのロボットは左腕設定であると仮定されます。
ビット2: BELOW（マスク値＝^H2）	このビットがONの場合、ロボットは当初、肩から手首の線よりも下に肘のあると仮定されます。それ以外の場合、ロボットの肘はこの線よりも上にあると仮定されます。スカラ構成のように、垂直面上で動く肘を持たないロボットでは、このビットは無視されます。
ビット3: FLIP（マスク値＝^H4）	このビットがONの場合、このロボットは手首のピッチ軸がマイナスの角度になるように設定されていると仮定され

ビットフラグ	説明
	ます。それ以外の場合、ロボットの手首のピッチ角はプラスの値に設定されていると仮定されます。3 軸手首を持たないロボットの場合、このビットは無視されます。
ビット 9: RIGHTY/LEFTY の変更（マスク値 = ^H100）	このビットが ON の場合、このコマンドはビット 10 で指定される RIGHTY/LEFTY 設定に対応するジョイント位置の組を計算しようとします。
ビット 10: RIGHTY への変更（マスク値 = ^H200）	ビット 9 が ON でこのビットも ON の場合、このコマンドは右腕設定に対するジョイント位置を計算しようとします。ビット 9 が ON でこのビットが 0 の場合、このコマンドは左腕設定に対するジョイント位置を計算しようとします。
ビット 11: BELOW/ABOVE の変更（マスク値 = ^H400）	このビットが ON の場合、このコマンドはビット 12 で指定される BELOW/ABOVE 設定に対応するジョイント位置の組を計算しようとします。スカラ構成のように、垂直面上で動く肘を持たないロボットでは、このビットは無視されます。
ビット 12: BELOW への変更（マスク値 = ^H800）	ビット 11 が ON でこのビットも ON の場合、このコマンドは肘下向き設定に対するジョイント位置を計算しようとします。ビット 11 が ON

ビットフラグ	説明
	でこのビットが 0 の場合、このコマンドは肘上向き設定に対するジョイント位置を計算しようとします。スカラ構成のように、垂直面上で動く肘を持たないロボットでは、このビットは無視されます。
ビット 13: FLIP/NOFLIP の変更 （マスク値 = ^H1000）	このビットが ON の場合、このコマンドはビット 14 で指定される FLIP/NOFLIP 設定に対応するジョイント位置の組を計算しようとします。3 軸手首を持たないロボットの場合、このビットは無視されます。
ビット 14: FLIP への変更 （マスク値 = ^H2000）	ビット 13 が ON でこのビットも ON の場合、このコマンドは FLIP 手首設定に対するジョイント位置を計算しようとします。ビット 13 が ON でこのビットが 0 の場合、このコマンドは NOFLIP 手首設定に対するジョイント位置を計算しようとします。3 軸手首を持たないロボットの場合、このビットは無視されます。
ビット 21: 縮退回避 （マスク値 = ^H10000）	このビットが ON の場合、ジョイント 2 の計算値が外側リンクを完全に伸長させた場合の 10° 以内である場合（値がジョイント 2 ±10°）、ジョイント 2 に対する範囲外エラーが発生します。

ビットフラグ	説明
ビット 22: 単回転ジョイント 4（マスク値 = ^H200000）	このビットが ON の場合、ジョイント 4 の計算値は -180° ～+180° の範囲に制限されます。
ビット 23: 直線動作（マスク値 = ^H400000）	このビットが ON の場合、返されるジョイント位置は、最初に指定されたものと同じ設定に対応している必要があります。ロボット設定の変更は許されません。

詳細

SOLVE.ANGLES プログラムコマンドは、システムに接続しているロボットの幾何的データを使用して、指定された変換値に相当する関節位置を計算します。指定された変換は、ワールド座標系においてロボットツールの端部の位置およびロケーションと解釈され、現在の TOOL 変換および BASE オフセットを考慮します。

例

以下の例は有効な機能を発揮しませんが、SOLVE.ANGLES コマンドの作用を説明することを目的としています。これらのコマンドを実行すると、jts2 および jts 配列の両方にほぼ同じ値が入ります。値の違いは、計算時の丸め誤差によるものです。

```
HERE #cpos
DECOMPOSE jts[] = #cpos
SOLVE.TRANS new.t, error = jts[]
SOLVE.ANGLES jts2[], flags, error = new.t, jts[], SOLVE.FLAGS(jts[])
```

関連キーワード

DECOMPOSE

SELECT（プログラムコマンド）

SELECT（実数値関数）

SOLVE.FLAGS

SOLVE.TRANS

SOLVE.TRANS

現在のロボットに対して与えられたジョイントポジションの組に相当する変換を計算します。

構文

SOLVE.TRANS transform, error = joints[index]

使用上の留意事項

このコマンドが実行する計算はリンクの寸法に基づくロボット幾何形状、軸数、ツールのオフセット、および基準フレームのオフセットの関数なので、異なる幾何パラメータを持つロボットでは結果が異なります。一般的なタイプが同じロボットであってもその寸法は若干異なる可能性があり、同じタイプの 2 つの異なるロボットシステムでこのコマンドを実行しても、若干異なる結果を返す可能性があります。

SOLVE.TRANS プログラムコマンドは、このコマンドを実行するタスクが選択したロボットを参照します。

eV+ システムがロボットを制御するように設定されていない場合、SOLVE.TRANS コマンドを実行しても、ロボットが存在しないので、エラーを生じません。しかし、返される情報に意味がなくなる可能性があります。

パラメータ

transform	結果を保存する変換変数または変換配列要素です。
error	コマンドの処理中に計算エラーが発生した場合に eV+ エラーコードに設定される実数値変数です。エラーが発生しない場合、この変数は 0 に設定されます。報告される唯一のエラーは、算術桁あふれ [-409] であるため、このパラメータを TRUE または FALSE 値を返すと見なすことができます。
joints	ジョイントポジションを含む実数配列であり、相当する変換に変換されます。この配列の最初の指定要素にはジョイント 1 の位置が入る必要があり、2 番目の要素にはジョイント 2 の値が入る必要があり、以下同様です。回転ジョイントの場合、ジョイントの位置は ° 単位で設定されます。直動ジョイントの場合、ジョイントの位置は mm 単位で設定されます。
index	ジョイント 1 のポジションの値を含む配列要素を特定する、オプションの整数値です。インデックスを指定しない場合、要素 0 にはジョイント 1 のポジションが入ります。

詳細

SOLVE.TRANS プログラムコマンドは、システムに接続しているロボットの幾何的データを使用して、ジョイントポジションの組を相当する変換値に変換します。計算された変換は、ワールド座標系においてツール端部のポジションと方向と解釈され、現在の TOOL 変換および BASE オフセットを考慮します。

例

次の例では、回転ジョイント 1 によって現在のロケーションを 10° 変える場合に、ロボットの移動するポジションと方向を計算します。

```
HERE #cpos
DECOMPOSE joints[1] = #cpos
joints[1] = joints[1]+10
SOLVE.TRANS new.trans, error = joints[1]
```

関連キーワード

DECOMPOSE

SELECT（プログラムコマンド）

SELECT（実数値関数）

SOLVE.ANGLES

SOLVE.TRANS

SPEED

後続するロボット動作の定格速度を設定します。

構文

SPEED speed_factor, r_speed_factor units *ALWAYS*

使用上の留意事項

プログラム実行が開始される場合および新しい実行サイクルが開始される場合、SPEED 100,100 ALWAYS が仮定されます。これは eV+ システムのデフォルト状態です。

関節補間動作と直線動作では動作速度の意味は異なります。

ロボットの動作速度はプログラム速度設定とモニタ速度設定の組み合わせによって決まります。

SPEED プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによって割り付けられていないのであれば、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、SPEED コマンドを実行するとエラーになります。

パラメータ

speed_factor	実数値、変数または式であり、その値は新しい速度係数として使用されます。値 100 は通常の最高速度であり、50 は最高速度の 1/2 の速度です。IPS または MMPS を単位として指定している場合、この値はツール先端の線形速度です。 フルスピード設定に関する詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPU ユニット編 (SBCA-464)』* を参照してください。
r_speed_factor	オプション実数値、変数または式であり、その値は新しい直線動作の回転数係数として使用されます。値 100 は通常の最高速度であり、50 は最高速度の 1/2 の速度です。
units	speed_factor パラメータの解釈の仕方を決定するオプションのキーワードです。IPS
ALWAYS	次の SPEED コマンドがプログラム速度を変更するまでプログラムの speed_factor を有効にすることを指定するオプションの修飾子です。これを指定しない場合、（APPROS および DEPART を含め）次の動作オペレーションだけに対して有効になります。

詳細

units パラメータを省略すると、モニタ速度係数が 100% であると仮定して、プログラム速度（定格ロボット作動速度）を決定します。

units に MONITOR を指定すると、モニタ速度が設定されます。この場合、パラメータ r_speed_factor を無視して、ALWAYS を仮定します。動作の実行速度はこのコマンドで指定する値と現在のプログラム速度設定を組み合わせることで決定されます。モニタ速度の変更は速やかに行われ、これには現在実行中の動作の残りの部分が含まれます。

IPS または MMPS を units パラメータに指定すると、speed_factor は直線動作のツール先端の絶対速度として解釈されます。この場合、speed_factor パラメータには、関節補間動作に対する直接的な意味はありません。

プログラム速度の変更とモニタ速度の変更では、連続経路動作の場合、その影響に若干の違いがあります。ロボットが連続する点に沿って移動するとき、ロボットはプログラム速度と指定された加速度を維持しながら、その点にできるだけ近づくようにやって来ます。プログラム速度を増加した場合、ロボットはプログラム速度と加速度を維持するために、実際の地点に対してよりおおまかな近似をします。

モニタ速度を増加した場合、命じられた目標位置を基準としたロボットの経路は変わりませんが、加速度が増加します。経路に沿うことが重要なアプリケーションの場合、モニタ速度を低い値に設定して経路を定義してから、より高いモニタ速度で正確に再生することができます。

モニタ速度を設定すると、その値は最大 100% までに制限されます。より高い速度設定値を指定しても、エラーは報告されません。速度を 0.000001（1.0E-6）未満にすることはできません。

プログラム速度を設定すると、その値は制御するロボットに応じてその最大値までに制限されます。より高い速度設定値を指定しても、エラーは報告されません。現在のロボットの最大速度値は、実数型関数SPEED (8) によって返されます。

大きなオフセットを持つツールをロボットに取り付けており、直線動作を実行する場合、ツール先端周りの回転を行うと、ロボット関節およびフランジの速度が非常に大きくなることがあります。r_speed_factor パラメータは直線動作中の最大ツール回転速度の制御を行います。

回転速度係数（r_speed_factor）を指定すると、それは、直線動作中に使用される最大直交座標系回転速度のパーセンテージと解釈されます。r_speed_factor パラメータを指定しないと、以下に示す結果のどれかが生じます。

- units パラメータを省略すると、回転速度は speed_factor の値に設定されます。
- units パラメータを指定した場合、回転速度は変更されません。

IPS または MMPS を指定すると、speed_factor は対応する公称速度に内部で変換されます。SPEED 関数を使用してプログラム速度を読み取った場合、返される値は速度係数のパーセンテージであり、絶対速度設定ではありません。

> **追加情報:** 最終的なロボット速度はモニタ速度（SPEED モニタコマンド）、プログラム速度（SPEED プログラムコマンド）および加減速（ACCEL プログラムコマンド）を組み合わせたものになります。

例

以下の例では、モニタ速度を 10) と仮定して、プログラム速度を次の動作の 50% に設定します。

```
SPEED 50
```

以下の例では、直線動作に対してモニタ速度を 100 と仮定して、公称ツール先端速度を 20 in/sec に設定します。ツール先端周りの回転は最大値の 40% までに制限されます。別の SPEED プログラムコマンドによって変更されるまで、この設定は有効のままになります。

```
SPEED 20, 40 IPS ALWAYS
```

以下においてはモニタ速度を公称値の 50% に設定します。

```
SPEED 50 MONITOR
```

関連キーワード

ACCEL

DURATION

IPS

MMPS

SCALE.ACCEL

SELECT

SELECT（実数値関数）

SPEED（モニタコマンド）

SPEED（実数値関数）

STOP

現在のプログラムタスクの実行を終了します。

構文

STOP

使用上の留意事項

さらに実行するプログラムサイクルがある場合、STOP はプログラムの実行を停止しません。

STOP コマンドによってプログラムを停止した場合、PROCEED プログラムコマンドを使用して実行プログラムを再開することはできません。

プログラムの実行が STOP コマンドによって停止した場合、すべてのアタッチされている I/O デバイスで FCLOSE および／または DETACH オペレーションが強制されます。

詳細

STOP プログラムコマンドは現在のプログラムの実行が終了させます。ただし、さらにプログラムループを行う予定の場合、そのプログラムを最初のステップから継続します（EXECUTE キーワードを参照してください）。プログラム実行パスの末尾をマークするために STOP コマンドを使用します。

> **注:** HALT プログラムコマンドは残りのすべてのサイクルをキャンセルするので、効果が異なります。
>
> **注:** メインプログラムにある RETURN プログラム命令は STOP コマンドを同じ効果があります。メインプログラムとは、EXECUTE キーワードまたは PRIME もしくは XSTEP モニタコマンドで呼び出されるプログラムです。サブルーチンとは、他のプログラムの中で、CALL または CALLS プログラムコマンド（または応答）によって呼び出されるプログラムです。

STOP プログラムコマンドは完了したものについてはプログラムサイクルのカウントを 1 つ増やし、残っているものについてはカウントを 1 つ減らします。結果として残っているサイクルがなくなると、プログラムの実行が停止します。

さらにサイクルが残っている場合、内部のロボット動作パラメータが再度初期化され、たとえ STOP オペレーションがサブルーチンまたは応答プログラム内で生じても、メインプログラムの最初のステップを使用してプログラムの実行が続きます。

関連キーワード

ABORT

ABORT(プログラムコマンド)

EXECUTE

HALT

PAUSE

SWITCH

値に基づいてシステムスイッチを有効または無効にします。

構文

SWITCH switch_name = value

SWITCH switch_name[index] **= value**

使用上の留意事項

指定したスイッチがインデックス修飾子を受け入れ、そのインデックスが 0 であるか括弧付きまたは括弧なしで省略されている場合、そのスイッチ配列のすべての要素は与えられた値に応じて設定されます。

パラメータ

switch_name	設定を変更されるシステムスイッチの名称です。スイッチの名称は一意に特定する最小長さに短縮することができます。
index	あるindexによって修飾可能なシステムスイッチに対して、対象となる特定のスイッチ要素を指定する、オプションの実数値、変数または式です。
value	システムスイッチが有効であるか無効であるかを決定する実数値、変数または式です。値がTRUE(0以外)であればスイッチは有効です。値がFALSE(0)であればスイッチは無効です。

詳細

SWITCH プログラムコマンドは、所定のスイッチを等号の右側の数値によって示される設定にします。

スイッチの名称は一意に特定する最小長さに短縮することができます。

システムスイッチに関する詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

例

以下の例では、SWITCH 関数およびコマンドを使用してシステムスイッチの設定を保存し、復旧する方法を示します。

```
old.upper = SWITCH(UPPER)
          .
          .
          .
SWITCH UPPER = old.upper
```

関連キーワード

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

SWITCH（モニタコマンド）

SWITCH（実数値関数）

TIMER

指定したシステムタイマーを所定の時間値に設定します。

構文

TIMER timer_number = time_value

使用上の留意事項

eV+ によって測定された時刻は わずか 1 ms（0.001 s）以内の精度であり、これよりも短い時間を測定することはできません。

0 以下のタイマーは読み取り専用であり、このコマンドでは設定できません。

パラメータ

timer_number	設定するタイマーの整数値の番号と解釈される実数値式です。この値は 1～15 の範囲にする必要があります。
time_value	タイマーに設定する時間（s）と解釈される実数値式です。このパラメータは、小数点以下の秒を指定することやマイナス値にすることができます。

詳細

以下の例に示すようにして使用した場合、タイマーを使用して TIMER コマンドを使用して設定してから 596 時間を測定することができます。タイマーの分解能は 1 ms であり、最大カウント数は 2.E+009 です。

> **追加情報:** TIMER 関数を使用してシステムタイマーの瞬間値を読み取ります。

例

以下の例では、TIMER コマンドおよび関数を使用して一定時間待機する 2 つの方法を提供します。それぞれの例で、タイマーを設定し、タイマー値が遅延時間だけ変化するまで待機します。

```
TIMER 1 = 0
WAIT TIMER(1) > delay
TIMER 1 = -delay
WAIT TIMER(1) > 0
```

関連キーワード

TIMER（実数値関数）

TOOL

ロボットのツール取り付けフランジを基準としたツール先端のロケーションと方向を表現するために使用する内部変換を設定します。

構文

TOOLtransformation_value

使用上の留意事項

TOOL プログラム命令は連続パスモーションにおいて中断を発生させます。

TOOL プログラムコマンドは、そのタスクで選択されたロボットが他のタスクによってアタッチされていない限り、任意のプログラムタスクによって実行することができます。このコマンドはタスクで選択されているロボットに適用されます。

eV+ システムがロボットを制御するように設定されていない場合、TOOL コマンドを実行するとエラーになります。

パラメータ

transformation_value	新しいツール変換であるオプションの変換変数もしくは関数、または複合変換式です。変換値が省略されている場合、ツールは NULL に設定されます。

詳細

TOOL プログラムコマンドはロボットの連続パスモーションに中断を発生させ、与えられた変換値に相当するツール変換の値を設定します。

このコマンドの使い方に関する詳細な説明については TOOL モニタコマンドを参照してください。ツール変換を定義するため、詳細については *『eV+ 3 ユーザーズマニュアル (SBCE-486)』* を参照してください。

関連キーワード

SELECT (プログラムコマンド)

SELECT(real-valued function)

TOOL(モニタコマンド)

TOOL(実数値関数)

TYPE

出力指定によって記述される情報をモニタウィンドウに表示します。

構文

TYPE output_specification, ..., output_specification

使用上の留意事項

MESSAGES システムスイッチが無効な場合、出力は発生しません。

パラメータが与えられない場合、空白行が出力されます。

通常、実行プログラムは出力の完了を待ってから次に進みます。後述する出力指定を、実行の遅延が望ましくない場合に待機を防止するために使用することができます。

1 つの TYPE コマンドの出力は 512 文字を超えることができません。以下に示す /S フォーマット指定子を使用するとこれよりも長いメッセージを出力することができます。

パラメータ

output_specification は、カンマで区切られた任意の順番の、次に示す任意の構成要素で設定することができます。

- 文字列式。
- 表示する値を決定するために判定される実数値式。
- 出力されるメッセージのフォーマットを決定するフォーマット制御指定子（以下の詳細参照）。

詳細

以下のフォーマット制御指定子を使用して、数値の表示方式を制御することができます。これらの設定は、その効果を変更するために別の指定子を使用しない限り、このコマンドの残りに対して有効なままになります。

これらの表示モードに対して、値が大きすぎて所定のフィールド幅に表示できない場合、そのフィールドにはアスタリスク文字（*）が入ります。

/D ではデフォルトフォーマットを使用して、1 つの先頭スペースを使用して値を完全な精度で表示します。1,000,000 以上の数値には指数表記を使用します。

> **注:** 以下のフォーマット規格ではフィールド幅（n）として 0 を受容します。これにより、実際のフィールドサイズは値に合わせるために変化し、すべての先頭スペースは抑圧されます。これは、値がテキスト行内または行の末尾に表示される場合に有用です。

/En.m　　指数表記の出力値であり、スペース n 個分の幅で、小数部が m 桁です。n が 0 ではない場合、(表示される値が負である場合の) マイナス記号、小数点の左側の 1 桁、(m が 0 でない場合の) 小数点、m 桁、および指数表示のための 4、5 桁のためのスペースを含めるために十分な大きさが必要です。

/Fn.m	固定小数点表記(-123.4 など)における出力値であり、スペースn 個分の幅で、小数部が m 桁です。
/Gn.m	値が 0.01 よりも大きい場合に、小数部 m 桁の F フォーマットにおける出力値であり、n 個のスペース幅のフィールドに適合します。そうではない場合、/Em.n フォーマットを使用します。
/Hn	n 個のスペース幅のフィールドにおける整数 16 進数としての出力値です。
/In	n 個のスペース幅のフィールドにおける整数 10 進数としての出力値です。
/On	n 個のスペース幅のフィールドにおける整数 8 進数としての出力値です。

出力の表示形式を制御するために以下の指定子をすることができます。

/Cn	キャリッジリターン(CR)およびラインフィード(LF)の文字 n 回分を出力します。これにより、フォーマット制御指定子が出力指定の最初または最後にある場合、n 行の空白行が生じます。それ以外の場合 n-1 行の空白行になります。
/S	現在の行を表示した後、キャリッジリターン(CR)もラインフィード(LF)も出力しません。
/Un	カーソルを上に n 行分移動します。
/Xn	n 個のスペースを出力します。

制御機能を実行するために以下の指定子を使用することができます。

/N	実行プログラムがその完了を待つことなく出力を開始します。2 番目の出力リクエストは実行プログラムに対して、最初の出力が完了していないのであれば、出力を待つことを強制します。

例

以下の例では、メッセージ「Point 5 = 12.67」を表示します。この例では、実数型変数「i」が値 5 になる、配列要素「point[5]」が値 12.666666 になります。

```
TYPE, "Point", i, " = " /F5.2, point[i]
```

point[5] の値が 1000 の場合、その値が大きすぎて指定されたフォーマット（/F5.2）で表示できないために、上記のステートメントは「Point 5 = *****」を表示します。フォーマット指定が /F0.5 であった場合、このコマンドは point[5] に任意の値を表示することができます。

関連キーワード

$ENCODE

MESSAGES

PROMPT

WRITE

UNTIL

DO ... UNTIL 制御構造体の末尾を示すと共に、そのループを脱出する時機を判定するために評価される式を指定します。この式の値が 0 以外になるまで、ループの実行を続けます。

構文

UNTIL expression

使用上の留意事項

UNTIL は必ず DO 制御構造体と共に使用します。詳細については DO コマンドを参照してください。

パラメータ

expression	TRUE(0 以外) または FALSE(0) のどちらかに解釈される実数値式、定数または関係です。

詳細

UNTIL ステートメント内の式が 0 の場合、実行プログラムは該当する DO コマンドに続くステートメントを続けます。式の値が 0 以外の場合、実行プログラムは該当する UNTIL コマンドに続くステートメントを続けます。

例

以下の例は 0 以上の数字を入力するまで数字の入力を求めるループです。

```
DO
    PROMPT "Enter a positive number: ", number
UNTIL number >= 0
```

関連キーワード

DO

EXIT

NEXT

WHILE

VALUE

直後のプログラムステートメントを実行するために、CASE ステートメントの式が一致する必要がある値を示します。

構文

VALUE expression_list:

使用上の留意事項

VALUE は CASE 制御構造体の一部である必要があります。詳細については CASE コマンドを参照してください。

パラメータ

expression_list	どのキーワードを実行するかを判断するために CASE 構造体において一致すべき値を定義する、コンマで区切られた、実数値、変数または式です。

関連キーワード

ANY

CASE

VPARAMETER

画像センサツールパラメータの現在の値を設定します。

構文

VPARAMETER(sequence_id, tool_id, parameter_id, index_id, object_id) **$ip** = **value**

使用上の留意事項

詳細については、*『ロボット用画像センサソフトウェアACE Sightリファレンスマニュアル (SBCE-400)』*を参照してください。

パラメータ

sequence_id	画像センサシーケンスのインデックスです。オプションです。最初のシーケンスは 1 です。
tool_id	ビジョンシーケンスでのツールインデックスです。オプションです。最初のツールは1 です。
parameter_id	オプションのパラメータ識別子(ID)です。
index_id	内部使用のための予備値は常に 1 です。
object_id	一部のパラメータは、配列の特定の値にアクセスするためにオブジェクトのインデックスが必要になります。
$ip	画像センササーバの IP アドレスです。画像センササーバは、Robot Vision Manager 画像センサソフトウェアが動作しているPC であり、標準の IP アドレス形式(例 192.168.1.120)を使用します。
value	実数値式です。

詳細

オプションのパラメータに値が指定されていない場合、デフォルトの値は 1 とみなされます。

例

次の例では、MaximumInstanceCountが - 519 であるオブジェクトインスタンスを、最大で4 つ検索するロケータを設定します。

```
VPARAMETER(1, 2, 519) $ip = 4
```

関連キーワード

VLOCATION

VPARAMETER（実数値関数）

VRESULT
VRUN
VSTATE
VTIMEOUT
VWAITI

VRUN

画像センサシーケンスの実行を開始します。

構文

VRUN **$ip**, **sequence_id**

パラメータ

$ip	画像センササーバの IP アドレスです。画像センササーバとは、Robot Vision Manager が動作している PC であり、標準の IP アドレス形式 (192.168.1.120 など) を使用します。
sequence_id	画像センサシーケンスのインデックスです。最初のシーケンスは 1 です。

例

次の例では最初のシーケンスを実行します。

```
VRUN $ip, 1
```

関連キーワード

VLOCATION

VPARAMETER（実数値関数）

VPARAMETER（プログラムコマンド）

VRESULT

VSTATE

VTIMEOUT

VWAITI

VWAITI

指定した画像センサシーケンスが type パラメータで指定する状態になるまで待機します。

構文

VWAITI (sequence_id) **$ip**, type

パラメータ

sequence_id	画像センサシーケンスのインデックスです。オプションです。最初のシーケンスは 1 です。
$ip	画像センササーバの IP アドレスです。画像センササーバとは、Robot Vision Manager が動作している PC であり、標準の IP アドレス形式(192.168.1.120など)を使用します。
type	待機を完了させる画像センサシーケンスの状態です。オプションです。値を次に示します。 • 0: 完全に完了するまで待機します(デフォルト)。 • 1: 撮影が完了するまで待機します。

詳細

VWAITI プログラムコマンドは VRUN プログラムの後に使用します。コンベア追跡アプリケーションにおいて、特定の VWAITI コマンドがないと、バーチャルカメラツールと通信ツールが干渉してアプリケーションの実行が遅れる可能性があります。

> **注:** VWAITI コマンドを発行することで、他のタスクが他の eV+ Robot Vision Manager キーワードを実行することを阻害してしまうことがあります。アプリケーションにRobot Vision Manager シーケンスと相互作用する複数の eV+ タスクがある場合、VSTATE の使用を検討してください。

例

次の例では、最初のシーケンスを実行し、完全に終了するのを待ちます。

```
VRUN $ip, 1
VWAITI (1) $ip, 0
```

関連キーワード

VLOCATION

VPARAMETER（実数値関数）

VPARAMETER（プログラムコマンド）

VRESULT

VRUN

VSTATE

VTIMEOUT

WAIT

1軌道サイクル分、プログラムを待機ループに入れます。条件を指定した場合、その条件が TRUE になるまで待機します。

構文

WAIT condition

使用上の留意事項

指定した期間待機させるには、WAIT.EVENT プログラムコマンドを使用してください。

実行中 PROCEED コマンドを使用すると、アプリケーションプログラムの WAIT コマンドをキャンセルします。

パラメータ

condition	TRUE(0 以外) または FALSE(0) かを試験されるオプションの実数値、変数または式です。

詳細

condition の指定がない WAIT プログラムコマンドは、それぞれの軌道サイクルであるオペレーションを 1 回だけ実行する必要があるプログラムで役立ちます。

condition が WAIT プログラムコマンドに与えられない場合、次の軌道サイクルまでプログラムの実行が中断されます。軌道サイクルはシステム設定に応じて 16、8、または 4 ms の間隔で発生します。

1 軌道サイクル分の遅延が必要な場合（例えば、操作信号が REACT または REACTI プログラムコマンドによってモニタされている時）は、2 つの連続する、引数のない WAIT コマンドを実行します。

condition が指定されている場合、WAIT コマンドによって、そのcondition が発生するまで、プログラムの実行が中断します。例えば、1 つ以上のデジタル信号の状態を継続のための condition として使用することができます。

例

次の例では、外部入力信号 1001 が ON で 1003 が OFF である間、プログラムの実行を停止して、それぞれの eV+ 軌道サイクルで呼び出しを行います。

```
WHILE SIG(1000, -1003) DO
    WAIT
END
```

関連キーワード

RELEASE

WAIT.EVENT

WAIT.START

WAIT.EVENT

指定したイベントが発生するまで、または指定した時間が経過するまで、プログラムの実行を中断します。

構文

WAIT.EVENT mask, timeout

使用上の留意事項

アプリケーションプログラム中の WAIT.EVENT プログラムコマンドがプログラムの実行を中断している場合、PROCEED モニタコマンドを使用してその WAIT.EVENT をキャンセルすることができます。

パラメータ

mask	待つイベントを指定するオプションの実数値、変数または式です。その値は以下に示すように連続するビットフラグと解釈されます。mask 値が指定されていない場合、すべてのビットはクリアであると仮定されます。 **Bit 1 (LSB, mask value = 1):** I/O を待つ このビットが ON の場合、現在のタスクによる入出力オペレーションの完了が希望するイベントになります。
timeout	待機秒数を指定するオプションの実数値、変数または式です。このパラメータを省略するか、その値が 0 またはマイナスの場合、タイムアウト処理は実行されません。

詳細

WAIT.EVENT プログラムコマンドは、指定したイベントが発生するまで、または指定した時間が経過するまで、プログラムの実行を中断するために使用されます。

WAIT.EVENT コマンドの後にプログラムが実行を再開する場合、GET.EVENT 関数を使用して希望するイベントが実際に発生したのかを確認することができます。これはタイムアウトを指定している場合に、イベントの発生とタイムアウトを区別する唯一の方法になります。

mask パラメータの値が 0 であるか、省略されている場合、このコマンドは、実行プログラムを timeout パラメータで指定する時間中断するための非常に効率的な方法になります。

timeout パラメータが省略されているか、0 またはマイナスの値である場合、このコマンドは指定したイベントが発生するまで無期限にプログラムの実行を中断します。

mask と timeout の両方が 0 であるか省略されている場合、このコマンドはシステムの挙動に影響を与えません。

WAIT.EVENT 1 ステートメントは、あるタスクに対してあるイベントが知らされるのを待ちます。イベントの伝達は SET.EVENT プログラムコマンドによるか、または保留中の待機なしI/OプログラムコマンドのI/Oオペレーション完了時になります。

イベントがセットされた理由を確認する方法はありません。I/O オペレーションによって設定された場合、SET.EVENT プログラムコマンドによる場合、または内部システム処理による場合が考えられます。そのため、WAIT.EVENT オペレーションを実行した後にその希望する条件を判定することが必要になります。I/O の場合、該当する待機なし I/O オペレーションを繰り返すか、IOSTAT 関数を使用してください。他のタスクによって SET.EVENT オペレーションが発行されたのであれば、グローバル変数を定義してチェックしてください。

判定と待機の間にイベントをセットまたはクリアするプログラムロジックの問題を回避するために、待機タスク中に以下のループを使用してください。

注: このループ内ではステートメントの順序が重要です。

1. CLEAR.EVENT
2. 該当する場合、待機なし I/O を発行します。
3. I/O の状態をチェックするか、グローバル変数をチェックします。
4. オペレーションが完了したら、ループを抜けます。
5. WAIT.EVENT 1
6. GOTO step 1

SET.EVENT を使用して他のタスクに信号を送っている場合、以下の手順を使用してください（ステートメントの順序が重要です）。

1. グローバル変数を設定する。
2. 該当するタスクのため SET.EVENT を実行する。

例

以下の例では、プログラムの実行を 5.5 秒間中断します。

```
WAIT.EVENT , 5.5
```

以下の例では、任意のシステム入出力が完了するまで、またはほかのプログラムタスクが SET.EVENT コマンドを使用してイベントをセットするまで、プログラムの実行を中断します。

```
WAIT.EVENT 1
```

以下の例では、任意のシステム入出力が完了するまで、またはほかのプログラムタスクが SET.EVENT コマンドを使用してイベントをセットするまで、5 秒間プログラムの実行を中断します。現在のプログラムで GET.EVENT 関数を使用して、イベントが発生したのか、5 秒経過したのかを判断してください。

```
WAIT.EVENT 1, 5
```

関連キーワード

CLEAR.EVENT

GET.EVENT

SET.EVENT

WHILE

条件がTRUE の場合WHILE 構造体の処理を開始し、その条件が最初からFALSE の場合は、WHILE 構造体をスキップします。

構文

WHILE condition DO

使用上の留意事項

このコマンドは完全な WHILE ... DO ... END 構造体の一部である必要があります。

パラメータ

condition TRUE(0 以外) または FALSE(0) 値であるかを試験されるオプションの実数値式です。

詳細

この構造体は、（DO コマンド同様に）ある制御条件が満たされるまでステートメントのグループを実行するもう一つの方法を提供します。WHILE 構造体の完全な構文を以下に示します。

```
WHILE condition DO
    group_of_steps
END
```

WHILE 構造体の処理は以下のように説明することができます。

1. 条件を判定する。結果が FALSE の場合、項目 4 に進む。
2. group_of_steps を実行する。
3. 項目 1 に戻る。
4. END コマンドの後の最初のステートメントからプログラムの実行を継続します。

他で説明した DO 構造体とは異なり、WHILE 構造体内のステートメントグループは全く実行されない可能性があります。WHILE を最初に実行するときに、条件が FALSE 値である場合、ステートメントグループは実行されません。

この構造体を使用すると、収容されているステートメントグループ内である作用が発生し、この構造体を抜けるべきときに論理式の結果が TRUE から FALSE に変化すると仮定されます。

例

次の例では、WHILE 構造体を使用して入力信号の組み合わせをモニタし、一連の動作を停止すべき時機を判断します。この例においては、どれかのパーツフィーダからの信号が 0（フィーダが空になったことを示す）になると、ロボットの繰り返し動作は停止し、プログラムは継続します。

WHILE 構造体が最初に実行されるときにどれかのフィーダが空になっている場合、実行により END の後にあるステートメントを直ちにスキップします。

```
feeder.1 = 1037
feeder.2 = 1038

WHILE SIG(feeder.1, feeder.2) DO
    CALL move.part.1()
      CALL move.part.2()
END
```

関連キーワード

DO

EXIT

NEXT

UNTIL

WINDOW

コンベアのトラッキングのため、指定したベルト変数の動作範囲の境界を設定します。

構文

WINDOW %belt_var = location, location, program, priority

使用上の留意事項

DEFBELT プログラムコマンドを使用して、参照されるベルト変数を事前に定義しておく必要があります。

パラメータ

%belt_var	ウィンドウが作成されるベルト変数の名称です。
location	ベルトの方向と一緒に、ベルトに沿った操作ウィンドウの 1 つの境界を定義する複合変換です。 ウィンドウの境界はベルトの移動方向に直交する面であり、2 つの変換によって指定される位置を含みます。変換の順序は重要ではありません。このコマンドは上流側の境界を示す変換と下流側の境界を示す変換を自動的に決定します。
program	ベルトのトラッキング中にウィンドウ違反が発生した場合に呼び出されるオプションのプログラムです。指定された優先度レベルおよびシステムの現在の優先度レベルに従います。
priority	オプションであるウィンドウ違反プログラムの優先度レベルです。priority を指定しない場合、priority は 1 に設定されます。

詳細

このプログラムコマンドが定義する操作ウィンドウは、モーション計画時とモーション実行時の両方において、モーションの目的のロケーションが許容限度内にあるかどうかを判断するために使用されます。

モーションの計画中、モーションの目的のロケーションは操作ウィンドウに対して比較されます。ウィンドウ違反が発生すると、ウィンドウ違反プログラムが無視され、BELT.MODE システムパラメータ設定およびそのエラーの特性に応じてプログラムエラーが発生します。

ベルトを基準とするモーションが実行中の場合、またはモーション完了後にロボットが目的のロケーションのトラッキングを継続している場合、すべての eV+ 軌道サイクルで目的のロケーションはこのウィンドウと比較されます。ウィンドウ違反が発生し、プログラムが指定されていると、優先度レベルに従ってプログラムが自動で呼び出され、ロボットはベルトのトラッキングを継続して、その連続パスモーションに従います。

> **注:** 指定されたプログラムが必要に応じてウィンドウ違反からの復帰を命令すると仮定しています。

プログラムを指定していない場合、ロボットは直ちに停止し、ウィンドウ違反プログラムエラーが送信されます。REACTE オペレーションを実行している場合、その REACTE オペレーションは有効化されます。そうではない場合、実行プログラムを終了します。

例

以下の例では、ロボットがベルトをトラッキングしているときにウィンドウ違反が発生すると、サブルーチン「belt.error」を実行します。ベルト変数「%belt1」の作業ウィンドウが location「win1」および「win2」によって定義されます。

```
WINDOW %belt1 = win1, win2, belt.error
```

関連キーワード

BELT（実数値関数）

BELT.MODE

BSTATUS

DEFBELT

SETBELT

WINDOW（実数値関数）

WRITE

開いているファイルまたは任意の I/O デバイスにレコードを書き込みます。ネットワークデバイスの場合、アタッチされたデバイスに文字列を書き込み、TCP 接続を開きます。

構文

WRITE (lun,record_num**) format_list**

使用上の留意事項

出力を受け取るデバイスをアタッチする必要があります。ディスクファイルに出力する場合、FOPENA または FOPENW プログラムコマンドを使用してそのファイルを開いておく必要があります。

フォーマットリストに /N 指定子がある場合を除き、実行プログラムは書き込みオペレーションの完了を待ちます。

パラメータ

lun — アクセスするデバイスを定義する実数式です。ユニット番号に関する説明については、ATTACH プログラムコマンドを参照してください。

record_num — 書き込まれるレコードの数を定義するオプションの実数式です。次のシーケンシャルレコードを書き込むためには、この値を0 (デフォルト) にする必要があります。この値が 0 ではない場合、レコードはランダムアクセスモードで書き込まれますが、そのためにはすべてのレコードが同じ長さである必要があります。ランダムアクセスモードでは、レコードは 1~16,777,216 の番号になります。

TCP デバイスにサーバプログラムを使用してアクセスする場合、このパラメータはクライアントハンドルを定義するオプションの実数、変数または式です。詳細については READ プログラム命令を参照してください。

format_list — 出力レコードを作成するために使用される出力変数、文字列式およびフォーマット指定子のリストを構成します。このフォーマットリストは、TYPE プログラムコマンドの出力仕様とまったく同様にして処理されます。

TCP デバイスにアクセスする場合、eV+ システムが書き込み承認を待たないようにするために、/N 指定子を含めることができます。

詳細

これはレコードを指定した論理ユニットに書き込む汎用のデータ出力コマンドです。レコードには任意の文字リストを含めることができますが、512 文字長を超えることはできません。

固定長レコードモードで開かれたファイルの場合、出力レコードがファイルレコードよりも短いとこのコマンドが NULL 文字を追加します。

TCP/IP デバイスにアクセスする場合、record_num パラメータによって、サーバが 1 つの論理ユニット上で複数のクライアントと通信することが可能になります。クライアントがサーバに接続するとハンドルが割り当てられ、クライアントが接続を切断すると割り当てが解除されます。接続中、TCP 論理ユニットからデータを受け取る READ プログラムコマンドはクライアントハンドルを返します。これにより WRITE プログラムコマンドはハンドル値を使用してデータを対応するクライアントに送信することができます。

例

次の例では、変数「$message」に保存されたメッセージを手動制御ペンダントに書き込みます。

```
ATTACH (1)
WRITE (1) $message
DETACH (1)
```

次の例では、可変長レコードを持つファイルをシステムディスクドライブに書き込みます。

```
ATTACH (dlun, 4) "DISK"
FOPENW (dlun) "A:testfile.dat"
FOR i = 0 TO LAST($lines[])
     WRITE (dlun) $lines[i]
END
FCLOSE (dlun)
DETACH (dlun)
```

次の例では、文字列「$str」をハンドルが指定するクライアントに書き込みます。このハンドルはメッセージを受け取った時に事前に定義しておく必要があります。これは承認を待ちません。

```
WRITE (lun, handle) $str, /N
```

関連キーワード

ATTACH

DETACH

FCLOSE

FEMPTY

FOPEN

FOPENA

FOPEND

FOPENR

FOPENW

IOSTAT

PROMPT

READ

TYPE

3.5 システムパラメータキーワード

本項の情報を参考にして、システムパラメータキーワードと eV+ システムでのそれらの使用方法について理解してください。

BELT.MODE

eV+ システムのコンベアトラッキング機能の特性を設定するシステムパラメータです。

構文

...BELT.MODE[robot_num]

使用上の留意事項

BELT.MODE パラメータの現在の値は、PARAMETER モニタコマンド、または関数キーワードで判断できます。

BELT.MODE パラメータの値は、PARAMETER モニタコマンド、またはプログラムコマンドキーワードでのみ修正できます。

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値として解釈されるオプションの実数値、変数、式です。パラメータが省略、または 0 の場合、すべてのロボットの設定が変更されます。それ以外の場合は、指定されたロボットの設定のみ影響されます。

詳細

このパラメータは、ビットフラグワードとして解釈されます。このパラメータの初期値は 0（すべてのビットが0）です。目的のビットマスク値をともに加算した値を割り当てることで、ビットを設定できます。

ビットフラグの意味を次に示します。

ビット 1 (LSB)	上流／下流の定義（マスク値 = 1） このビットが 1 に設定されている場合、ベルトのその瞬間の走行方向が内部モーションプランナーとWINDOW実数値関数のウィンドウテストルーチンで、上流と下流を定義するために使用されます。 このビットが 0 に設定されている場合、上流から下流への移動は、常にノミナル変換のX 軸の正方向への移動に対応します。
ビット 2	停止ベルト処理（マスク値 = 2） このビットが 1 に設定された状態で、目的のロケーションがベルトウィンドウの外側にあり、ベルトが停止している場合、モーションプランニング中にプログラムエラーが発生します。

このビットが 0 に設定された状態で、モーションプランニング中にベルトが停止した場合、ノミナル変換の X 軸の正方向が下流方向の定義に使用されます。この場合、通常のウィンドウエラーの基準が適用されます（以下参照）。

ビット 3　ウィンドウエラーの定義（マスク値 = 4）

このビットが 1 に設定されている場合、目的のロケーションがベルトウィンドウの下流または上流にあるとき、プランニング中のモーション命令は失敗します。

このビットが 0 に設定されている場合、上流のウィンドウ違反が発生すると、ロケーションがウィンドウに入るまでプランニングは待機状態になります。目的のロケーションがベルトウィンドウの下流にある場合、ウィンドウエラーが発生します。

ビット 4　ウィンドウエラーの影響（マスク値 = 8）

このビットが 1 に設定されている場合、ウィンドウエラーのためプランニング中に失敗したモーション動作は無視（スキップ）されますが、プログラムの実行は正常に継続します。このオプションが選択されている場合、各ベルト関連モーション動作の後に、BSTATUS 関数を使用したプランニングエラーに対する明示的なテストが必要です。

このビットが 0 の場合、モーションプランニング中にウィンドウエラーが発生すると、ププログラムステップの実行がエラーになり、プログラムの実行が停止する、または REACTEルーチンがトリガーされます。

このビットの設定にかかわらず、ロボットが実際にベルトをトラッキングしている間にウィンドウエラーが発生した場合、最新の WINDOW 演算で指定されたプログラムが実行されます。このようなプログラムが指定されていない場合、プログラムの実行は停止します。

例

次の例では、パラメータのビット 1 とビット 3 を 1 に設定しています（マスク値 1 + 4）。

```
PARAMETER BELT.MODE = 5
```

関連キーワード

BELT実数値関数

BSTATUS 実数値関数

PARAMETER（モニタコマンド）

PARAMETER（プログラムコマンド）

PARAMETER （実数値関数）

WINDOW（プログラムコマンド）

WINDOW 実数値関数

DEVIATION

ロボットが特異点にあるとき、特異点領域のモーションに 1～100% のパス偏差を追加します。

構文

DEVIATION[robot_num] = value

使用上の留意事項

valueを0に設定すると、特異点処理は無効になります。これを1から100の値に設定すると、経路から逸脱します。

追加情報: スカラロボットは特異点領域の影響を受けません。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略されているか0の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

このパラメータは特異点に関連する問題を処理します。ロボットが特異点にある場合を検出し、特異点領域における動作に少量の偏差を加えます。ロボットが特異点から出ても、当初の経路にはもどりません。

例

次の例ではロボット 1 に対して2%の偏差を設定します。

```
PARAMETER DEVIATION[1] = 2
```

関連キーワード

PARAMETER（実数値関数）

PARAMETER（モニタコマンド）

PARAMETER（プログラムコマンド）

JOG.TIME

ジョグオペレーションのキープアライブタイムを設定するシステムパラメータです。

構文

...JOG.TIME

使用上の留意事項

JOG.TIME パラメータの値は、PARAMETER モニタコマンド、またはプログラムコマンドキーワードでのみ修正できます。

eV+ システムが複数のロボットを制御している場合、JOG.TIME パラメータはすべてのロボットに対してジョグオペレーションのキープアライブ時間を制御します。

JOG.TIME パラメータの現在の値は、PARAMETER モニタコマンド、または関数キーワードで確認できます。

詳細

JOG プログラムコマンドおよびモニタコマンドのキーワードはジョグ動作を実行するために使用されます。

JOG.TIME パラメータ設定によってジョグオペレーションのキープアライブタイムが決まります。ジョグ運転が実行されるたびに、このパラメータ設定によって軸またはジョイントの動作時間が決まります。

このパラメータ値は 0.1～5.0 秒の範囲にすることができます。このパラメータは eV+ システムが初期化される場合、0.3 秒に設定されます。

例

次の例ではキープアライブ時間を 0.5 秒に設定します。

```
PARAMETER JOG.TIME = 0.5
```

関連キーワード

JOG（プログラムコマンド）

JOG（モニタコマンド）

PARAMETER（プログラムコマンド）

PARAMETER（実数値関数）

PARAMETER（モニタコマンド）

NOT.CALIBRATED

システムに接続されているロボットのキャリブレーション状態を示すシステムパラメータです。

構文

...NOT.CALIBRATED

使用上の留意事項

NOT.CALIBRATED システムパラメータの現在の値は、PARAMETER モニタコマンド、または関数で判断できます。

その結果は、キャリブレーションオペレーションによってランタイムキャリブレーションが実施されたことを示します。キャリブレーションのオフセットが事前に保存されていることを示すものではありません。

パラメータ名は短縮できます。

詳細

このパラメータの値の範囲は 0～255 で、ビットマスクとして解釈する必要があります。ビット 1～8 は、ロボットの 1～8 に対応します。値は次に示すように解釈されます。

パラメータ値	意味
0	すべてのロボットがキャリブレーション済み。
1	ロボット 1 がキャリブレーションされていない。
3	ロボット 1 とロボット 2 がキャリブレーションされていない。
7	ロボット 1 からロボット 3 までがキャリブレーションされていない。

電源ON 時、すべての設置済みロボットがキャリブレーションされていないことを示すように、このパラメータはセットされます。ロボットが接続されていない、または定義されていない場合、NOT.CALIBRATED ビットは常に OFF です。

CALIBRATE モニタコマンドとプログラムコマンドは、NOT.CALIBRATED ビットが ON になっている有効なロボットのキャリブレーションを試みます。キャリブレーション動作が完了した時点で、NOT.CALIBRATED ビットは更新されます。

1 台のロボットのみが設置されているシステムを例にすると、CALIBRATEモニタコマンドが発行され、成功した場合、NOT.CALIBRATED は 0 に設定されます。3 台のロボットが接続されている場合で、CALIBRATE モニタコマンドがロボット1 と 2 のキャリブレーションに成功したが、ロボット 3 は成功していない場合、

NOT.CALIBRATED は 4（2進数で 100: ロボット 1、2 はキャリブレーション完了、ロボット 3 は未完了）に設定されます。

このパラメータの目的は、キャリブレーションされていないロボットに対応するビットをON にして、次回、CALIBRATE モニタコマンドまたはプログラム命令でキャリブレーションを実行できるようにすることです。また、このパラメータは、ロボットのキャリブレーション状態の判断にも使用できます。

パラメータ値は常に変更されえます。新しい値にどう影響を与えるかを、次のルールで説明します。

- 新しい値がロボットがキャリブレーションされていないことを示す場合、サーボソフトウェアが、ロボットがキャリブレーションされていると認識しているかどうかにかかわらず、eV+ システムはロボットがキャリブレーションされていないかのように動作します。
- 新しい値がロボットがキャリブレーションされていることを示す場合、eV+ はそのロボットのキャリブレーション完了または未完了状態を追跡します。

  注: 通常、PARAMETER NOT.CALIBRATED ステートメントを使用して、ビットをOFFにすることには意味がありません。

例

次の例では、キャリブレーションされていないすべてのロボットに対してランタイムキャリブレーションを実行します。

```
IF PARAMETER(NOT.CALIBRATED) <> 0 THEN
    CALIBRATE
END
```

関連キーワード

CALIBRATE

CALIBRATE （プログラムコマンド）

PARAMETER（モニタコマンド）

PARAMETER （プログラムコマンド）

PARAMETER（実数値関数）

ROBOT

VTIMEOUT

後続するビジョンコマンドの応答が無い場合に、エラーメッセージを返すまでのタイムアウト値を設定するシステムパラメータです。

構文

PARAMETER VTIMEOUT = value

詳細

ステートメント VTIMEOUT = 0 はタイムアウト値 16 ms を設定します。これは使用可能な最小のタイムアウト値です。

タイムアウト値は秒で表されます（0.15 は 150 ms を意味します）。

例

次の例ではタイムアウト値を設定し、200 ms 後に応答がないとエラーメッセージが発生します。

```
PARAMETER VTIMEOUT = 0.20
```

関連キーワード

PARAMETER（プログラムコマンド）

PARAMETER（実数値関数）

VLOCATION

VPARAMETER（実数値関数）

VPARAMETER（プログラムコマンド）

VRESULT

VRUN

VSTATE

VWAITI

3.6 システムスイッチキーワード

本項の情報を参考にして、システムスイッチキーワードと eV+ システムでのそれらの使用方法について理解してください。

AUTO.POWER.OFF

このシステムスイッチは特定の動作エラーが生じた際にハイパワーを無効にします。

構文

...AUTO.POWER.OFF[robot_num]

使用上の留意事項

このシステムスイッチは自動モード中は機能しますが、マニュアルモード中は機能しません。これは、一般的なヌリングタイムアウトやエンベロープエラーの際にオペレータによる介入の必要性を減少させる上で特に役立ちます。

パラメータを指定せずにこのスイッチをモニタコマンドと共に使用すると、システム内のすべてのロボットが対象になります。

このシステムスイッチは、デフォルトでは無効になっています。

パラメータ

robot_num	影響を受けるロボットの番号を示す必須の実数値、変数、または式です。整数として解釈されます。

詳細

このスイッチが有効になると、robot_num で指定したロボットに関連動作エラーが発生した場合、eV+ は常にハイパワーを無効にします。 関連動作エラーは次に示すものです。

- ソフトエンベロープエラー
- ソフトオーバースピードエラー
- フォース保護制限超過

例

次の例では、関連動作エラーが発生した場合にロボット1 へのハイパワーを無効にするよう、eV+ に指示します。

```
ENABLE AUTO.POWER.OFF[1]
```

関連キーワード

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE (プログラムコマンド)

SWITCH(モニタコマンド)

SWITCH (プログラムコマンド)

SWITCH（実数値関数）

CP

ロボットの連続パス機能を制御するシステムスイッチです。

構文

...CP[robot_num]

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値として解釈されるオプションの実数値、変数、式です。パラメータが省略、または 0 の場合、すべてのロボットの設定は変更されます。それ以外の場合は、指定されたロボットの設定のみ影響されます。

詳細

連続パスモーション処理の OFF に CP スイッチを使用できます。

eV+ システムが初期化されると、このスイッチは ON になります。

連続パス軌道の詳細については、*『eV+ 3 ユーザーズマニュアル (SBCE-486)』*を参照してください。

例

次の例では、システムのすべてのロボットに対して、連続パスモーション処理を OFF にしています。

```
DISABLE CP
```

関連キーワード

BREAK（プログラムコマンド）

CPOFF（プログラムコマンド）

CPON（プログラムコマンド）

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

SWITCH（モニタコマンド）

SWITCH（プログラムコマンド）

SWITCH 実数値関数

DECEL.100

ACCEL プログラムコマンドキーワードに対して 100% の最大減速を有効または無効にするシステムスイッチです。

構文

...DECEL.100[robot_num]

使用上の留意事項

選択したロボットに対して DECEL.100 を有効にすると、ACCEL プログラムコマンドキーワードで指定された値が制限され、最大減速率が使用されます。

DECEL.100 はデフォルトではすべてのロボットに対して無効になっています。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数、式です。整数として解釈されます。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

例

次の例では、ACCEL プログラムコマンドキーワードにおいて、ロボット2 の最大減速度として 100%が使用されます。

```
DECEL.100[2]
```

関連キーワード

ACCEL（プログラムコマンド）

ACCEL（実数値関数）

SPEED（モニタコマンド）

SPEED（プログラムコマンド）

DELAY.IN.TOL

eV+ システムが動作区間を完了した後、粗ヌリングまたは微ヌリングのタイミングを制御するシステムスイッチです。

構文

...DELAY.IN.TOL [robot_num]

使用上の留意事項

多くのアプリケーションで、このスイッチを有効にすると、最良のヌリング動作を実現できます。しかし、以前の eV+ システムとの後方互換性のために、このスイッチを無効にする必要があります。

このスイッチを有効にしても、エミュレーションモードでは、動作に違いはありません。エミュレーションモード中のロボットの軌道は、現在の目標位置と同じとみなされ、動作が完了したとき、位置エラーは発生します。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数、式です。整数として解釈されます。ENABLE、またはDISABLE キーワードで、この値が省略または 0である場合、すべてのロボットの設定が変更されます。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

DELAY.IN.TOL システムスイッチはすべてのスカラロボットに対してデフォルトで無効になっています。

このスイッチが有効になっている場合、ロボットが到達する最終的に指令された位置はより正確になりますが、その動作を完了するための全体の時間が増加します。ロボットが指定されたFINE / COARSE 許容範囲内で動作区間内の終点に到達すると、システムにより遅延が追加されます（通常2〜3 ミリ秒）。この時間中、システムは指令された速度が0 に到達するまで待機し、ロボットは次の動作を開始できるようになるまで指定された許容範囲内に留まります。この機能により、指定されたFINE / COARSE 許容範囲内で希望するロケーションに到達し、完全に停止してから次の動作区間に進みます。

このスイッチが無効になっている場合、ロボットが指定されたFINE / COARSE 許容範囲内で最終的に命令されたロケーションに到達すると、遅延なく直ちに次の動作が命令されます。

動作中にこのスイッチの状態を変えても、その時点のロボット動作に影響はありません。

例

次の例ではロボット 2 に対して DELAY.IN.TOL システムスイッチを有効にします。

```
ENABLE DELAY.IN.TOL[2]
```

関連キーワード

COARSE

FINE

NULL

NONULL

DRY.RUN

eV+ がロボットと通信するかしないかを制御するシステムスイッチです。

構文

...DRY.RUN[robot_num]

使用上の留意事項

DRY.RUN システムスイッチは、アプリケーションプログラムによって有効、または無効にできますが、このスイッチの新しい設定は、以下に示すイベントが次に発生するまで、有効になりません。

- タスク 0 に対して EXECUTE キーワードが処理される
- ロボットが ATTACH キーワードを使用して割り付けられる
- CALIBRATE キーワードが実行される

アプリケーションプログラムが DRY.RUN スイッチの設定を変更する前に、プログラムでロボットを切り離す必要があります。そうしない場合、スイッチ設定を変更しようとするとエラー（-602）「Robot already attached to program」（ロボットはプログラムに割付済）が生じます。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数、式です。整数として解釈されます。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

このシステムスイッチを使用すると、eV+ がロボットへ動作コマンドを送信することやロボットから位置情報を受信することを停止することができます。システムが DRY.RUN モードの場合、ロボット衝突の懸念なしに、アプリケーションを実行して、適切な論理フローや正しい外部通信といった事項をテストすることができます。

システムが DRY.RUN モードの間は、ロボットの制御にペンダントを依然使用できます。

ロボットを割り付けると、DRY.RUN スイッチが確認されます。タスク 0 は、プログラムの実行が開始または再開すると、ロボットの割り付けを試みます。あるタスクに対する DRY.RUN 設定は、実行中にロボットを割り付け解除して DRY.RUN を変更したうえでロボットを割り付けることで変更することができます。

> **注:** DRY.RUN の状態が予想とは異なる可能性があるため、複数のタスクが同時に DRY.RUN を変更することはできません。この場合、プログラムでソフトウェアインターロックを使用する必要があります。

DRY.RUN スイッチは、デフォルトでは無効になっています。



警告: デジタル I/O とアナログ I/O は、DRY.RUN の影響を受けないため、アナログまたはデジタル出力コマンドによって駆動される外部デバイスは、動作を続けます。

例

次の例ではロボット 2 に対して DRY.RUN システムスイッチを有効にします。

```
ENABLE DRY.RUN[2]
```

関連キーワード

DISABLE

ENABLE

SWITCH（モニタコマンド）

SWITCH（プログラムコマンド）

SWITCH（実数値関数）

MESSAGES

モニタウィンドウに対する TYPE コマンドプログラムからの出力を有効/無効にするシステムスイッチです。

構文

...MESSAGES

詳細

このシステムスイッチを有効にした場合、TYPE プログラムコマンドからの出力が、モニタウィンドウに表示されます。それ以外の場合は、出力は表示されません。

このスイッチは、デフォルトでは有効になっています。

例

次の例では、MESSAGES システムスイッチが無効な場合に有効にして、モニタウィンドウに "Messages are now enabled"を表示します。

```
IF (SWITCH(MESSAGES)==FALSE)THEN
    ENABLE MESSAGES
    TYPE "Messages now enabled"
END
```

関連キーワード

DISABLE(モニタコマンド)

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE (プログラムコマンド)

SWITCH(モニタコマンド)

SWITCH (プログラムコマンド)

SWITCH（実数値関数）

TYPE

OBSTACLE

選択されたロボットに対して最大で 4 つの障害を有効、または無効にするシステムスイッチです。

構文

...OBSTACLE[switch_ID]

使用上の留意事項

SELECT プログラムコマンドを使用してロボットを指定してから OBSTACLE システムスイッチを操作する必要があります。

OBSTACLE ステートメントに switch_ID パラメータ値が含まれていない場合、「Invalid argument」（-407、無効な引数）エラーが発生します。

障害は、Sysmac Studio ソフトウェアを使用して設定する必要があります。詳細については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』*を参照してください。

パラメータ

switch_ID	有効または無効にしたい障害を示す整数値として解釈される必須の実数値、変数、式です。 1～4 の値のみが許可されます。

詳細

システムを構成する各ロボットに対して、最大 4 つの障害を使用できます。各障害は、OBSTACLE システムスイッチパラメータ値の switch_ID で有効、または無効にできます。次の例では、システムの各ロボットに関連する障害を有効、無効にする方法を示しています。

選択したロボットの障害の状態を読み取るために、SWITCH キーワードを使用できます。次の例では、システムの各ロボットに関連する障害を読み取る方法を示しています。

> **追加情報:** 障害の設定については、*『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

例

次のプログラムコマンドの例では、ロボット 2 に対して障害 1 を有効にします。

```
SELECT ROBOT = 2
SWITCH OBSTACLE [1] = TRUE
```

次のプログラムコマンドの例では、ロボット 2 に対して障害 1 と障害 2 を有効にします。

```
SELECT ROBOT = 2
ENABLE OBSTACLE [1], OBSTACLE [2]
```

次のプログラムコマンドの例では、ロボット 1 に対して障害 1 を有効にし、ロボット 2 に対して障害 1 を無効にします。

```
SELECT ROBOT = 1
ENABLE OBSTACLE [1]
SELECT ROBOT = 2
DISABLE OBSTACLE [1]
```

次の実数値関数の例では、ロボット 2 に対して障害 2 の変数「VAR」の状態を返します。

```
SELECT ROBOT = 2
VAR = SWITCH(OBSTACLE [2])
```

次のモニタコマンドの例では、モニタウィンドウにすべてのロボットの障害の状態をすべて一覧にします。

```
SWITCH OBSTACLE
```

次のモニタコマンドの例では、モニタウィンドウにロボット 2 の障害 1の状態を一覧にします。

```
SELECT ROBOT = 2
SWITCH OBSTACLE [1]
```

関連キーワード

ENABLE（プログラムコマンド）

ENABLE（モニタコマンド）

DISABLE（プログラムコマンド）

DISABLE（モニタコマンド）

SELECT（プログラムコマンド）

SELECT（モニタコマンド）

SWITCH（プログラムコマンド）

SWITCH（モニタコマンド）

POWER

ロボットハイパワーの状態を制御または監視するシステムスイッチです。

構文

...POWER[robot_num]

使用上の留意事項

アプリケーションプログラムが POWER スイッチの設定を変更する前に、プログラムでロボットを切り離す必要があります。そうしない場合、スイッチ設定を変更しようとするとエラー（-314）「Switch can't be enabled」（スイッチを有効化できません）が生じます。



危険: システムが欧州認証に準拠しない限り、プログラムから電源を有効にするために、POWER スイッチを使用しないでください。欧州認証に準拠している場合、警告なしにロボットが起動しないように、特別な安全機能がシステムに組み込まれています。詳しくは、次の「詳細」項を参照してください。



危険: プログラムによってこのスイッチを使用してハイパワーを ON にすると、オペレータの直接的なアクションなしでロボットを作動させることができるので、潜在的な危険があります。ロボットワークスペースを明瞭に見渡せない状態や非常停止ボタンをすぐに押せない状態にある場合、端末からハイパワーをオンにすることは危険である可能性があります。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

このスイッチを有効にすることは、ペンダントの [COMP/PWR] ボタンを押して、ハイパワーをオンにすることと同じです。電力をオンにすることを阻害するエラー状態がないのであれば、この有効化プロセスは次に示す第 2 ステップに移行しますので、フロントパネルの [HIGH POWER] ボタンを押す必要があります。欧州認証に準拠していないシステムの場合、次のステップは必要ありません。

このスイッチを無効にするには、ロボットの減速を制御し、パワーダウンシーケンスを実行する必要があります。このシーケンスは、次のステップで構成されます。

1. ユーザー指定のパラメータに従って、すべてのロボットを減速します。以下の注意を参照してください。
2. ブレーキをオンにします。

3. ユーザー指定のブレーキ遅延間隔の間、待機します。以下の注意を参照してください。
4. アンプおよび電源をオフにします。
5. バックプレーンの非常停止信号をアサートし、ハイパワー有効（HPE）信号をデアサートします。

注: DISABLE POWER は、減速時間が長く、ブレーキオン遅延が長いために、不定的に長時間を要する可能性があります。速やかにシャットダウンする必要がある場合は ESTOP キーワードを使用してください。ハイパワーがオンであるかオフであるかを確認するために、SWITCH 実数値関数を使用してこのスイッチの値をいつでも確認することができます。

エラーを発生させずに、ロボットプログラムから電源を無効にするためには、プログラムを DRY.RUN モードにするか、ロボットをプログラム制御から切り離す必要があります。詳細については DRY.RUN システムスイッチまたは DETACH プログラムコマンドを参照してください。

例

次の例ではロボットを切り離してハイパワーをオフにします。

```
DETAC
DISABLE POWER[2]
```

関連キーワード

DETACH

DISABLE（モニタコマンド）

DISABLE（プログラムコマンド）

DRY.RUN

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

ESTOP（プログラムコマンド）

SWITCH（モニタコマンド）

SWITCH（プログラムコマンド）

SWITCH（実数値関数）

ROBOT

1 台のロボットまたはすべてのロボットを有効／無効にするシステムスイッチです。

構文

...ROBOT [robot_num]

使用上の留意事項

次の条件を両方満たす場合のみ、ROBOT システムスイッチを修正できます。

1. POWER システムスイッチは OFF になっている。
2. eV+ システムがディスクから起動したとき、少なくとも 1 台のロボットが致命的なエラーを報告せずに起動している。

ROBOT スイッチは、致命的なエラーなしに起動したロボットに対してのみ修正できます。

ロボットが有効かを判断するために、SWITCH キーワードモニタコマンド、または実数値関数を使用して、いつでもこれらのスイッチの設定を確認できます。

パラメータ

robot_num	影響を受けるロボットの番号を示す整数値と解釈されるオプションの実数値、変数、または式です。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

ディスクから起動した後に eV+ システムが起動したとき、致命的なエラーを報告せずに起動したすべてのロボットはデフォルトで有効になり、該当するすべての ROBOT システムスイッチは有効になります。起動後、ROBOT スイッチを使用して、ロボットを選択的に無効にできます。これは各ロボットのトラブルシューティングに役立ちます。

ROBOT スイッチで無効にされたロボットは、次の場合でも無視されます。

- ペンダントの COMP/PWR ボタンまたはPOWER システムスイッチですべてのロボットの電源が有効になっている。
- すべてのロボットが CALIBRATE キーワードで校正されている。

注: 無効になっているロボットに対して動作操作を実行しないでください。

例

次の例では、ロボット 2 を無効にしています。

```
DISABLE ROBOT[2]
```

関連キーワード

DISABLE(モニタコマンド)

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE（プログラムコマンド）

SWITCH(モニタコマンド)

SWITCH（プログラムコマンド）

SWITCH（実数値関数）

SCALE.ACCEL

プログラム速度の関数として加速度および減速度のスケーリングを有効または無効にするシステムスイッチです。

構文

...SCALE.ACCEL [robot_num]

使用上の留意事項

SWITCH 実数値関数でこのスイッチにアクセスし、robot_num が省略、または 0 の場合、ロボット1に対するスイッチの設定を返します。

このシステムスイッチはプログラム速度が SCALE.ACCEL の上限以下である限り、加速度のスケーリングに影響します。この設定に関する詳細については *『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』* を参照してください。

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数または式です。整数として解釈されます。このパラメータが省略されているか 0 の場合、すべてのロボットに対する設定が変わります。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

eV+ システムが初期化されるとこのスイッチが有効になります。

このスイッチが有効で、プログラム速度がプリセットされたしきい値以下の場合、そのロボットに対する有効な加速度と減速度は、次のようになります。ここで acceleration_setting および deceleration_setting は ACCEL キーワードによって設定される値です。

```
有効な加速度 = プログラム速度 * acceleration_setting
有効な減速度 = プログラム速度 * deceleration_setting
```

例えば、プログラム速度が 50% に指定され、しきい値が 150 の場合、有効な加速度と減速度は、現在の設定値の 50% になります。しきい値が 150 に設定され、プログラム速度が 150% より高い場合、現在の加速度と減速度は、修正なしに使用されます。

> **注:** すべてのロボットでは、その SCALE.ACCEL 速度しきい値はデフォルトで非常に大きい値（1e16）に設定されており、このスイッチが有効になるとすべての速度で加速度および減速度のスケーリングを事実上強制します。



警告: 例えば、プログラム速度が 100% 以上で、SCALE.ACCEL の制限値にデフォルトの設定が使用され、SCALE.ACCEL が有効な場合、ロボットは、V+ 11.0 と比較してかなり高い割合の加速度と減速度で駆動されます。

ロボットの SCALE.ACCEL スイッチが無効な場合、加速度と減速度は、プログラム速度に基づき増減されません。この場合、加速度と減速度は、通常の安全運転速度より高くなります。これは超低速において特に顕著であり、その結果としてロボットの動作はラフで突発的に見える可能性があります。

例

ロボット 2 の加速度スケーリングをオフにします。

```
DISABLE SCALE.ACCEL[2]
```

関連キーワード

ACCEL（プログラムコマンド）

ACCEL（実数値関数）

SPEED（モニタコマンド）

SPEED（実数値関数）

SCALE.ACCEL.ROT

SCALE.ACCEL.ROT

直線動作中、SCALE.ACCEL スイッチが直交座標回転速度を考慮するかどうかを指定するシステムスイッチです。

構文

...SCALE.ACCEL.ROT [robot_num]

パラメータ

robot_num	影響を受けるロボットの番号を示すオプションの実数値、変数または式です。整数として解釈されます。ENABLE、DISABLE キーワードでこの値が省略、または 0 の場合、すべてのロボットの設定が変更されます。それ以外の場合、指定されたロボットに対する設定だけが影響を受けます。

詳細

選択したロボットに対して SCALE.ACCEL.ROT を有効にすると、直交座標系の直線動作速度および回転速度の小さいほうの値を使用して、直線動作中の加速度および減速度をスケーリングします。選択したロボットに対して SCALE.ACCEL.ROT を無効にすると、SCALE.ACCEL が有効な時に、直交座標系の直線動作速度だけを考慮します。eV+ システムを初期化すると、デフォルトですべてのロボットに対して SCALE.ACCEL.ROT スイッチが有効になります。

例

次の例では、SCALE.ACCEL はロボット 2 に対して直交座標回転速度を使用しなくなります。

```
DISABLE SCALE.ACCEL.ROT[2]
```

関連キーワード

ACCEL

ACCEL（実数値関数）

SPEED（モニタコマンド）

SPEED（実数値関数）

UPPER

文字列を比較するとき、各文字の大文字と小文字の違いを制御するシステムスイッチです。

構文

...UPPER

使用上の留意事項

このシステムスイッチの値は、すべてのプログラムタスクでグローバルに共有されます。1 つのタスクで値を変更した場合、他のすべてのタスクにも影響します。通常のプログラム実行中は、このスイッチを変更しないでください。

詳細

このスイッチが有効で、演算子 <、<=、==、<>、>=、> を使用して 2 つの文字列を比較した場合、すべての小文字は大文字として扱われます。UPPER が有効な場合、次の比較はどちらも TRUE 値を返します。

```
"a" == "A"      and     "A" == "A"
```

UPPER が無効な場合、文字列の比較中、大文字、小文字は考慮されます。例えば、上記の左側のステートメントは、FALSE 値を返しますが、右側のステートメントは TRUE 値を返します。

大文字と小文字を区別せずに文字列を比較できるように、UPPER はデフォルトで有効になっています。

STRDIF 関数は常に大文字と小文字を区別して文字列を比較します。重要な状況では UPPER を有効にしたまま STRDIFを使用できます。

例

次の例では、モニタウィンドウに「0」（TRUE）が表示されます。

```
$s1="Case"
$s2="case"


DISABLE UPPER
a= $s1==$s2
TYPE a
```

次の例では、モニタウィンドウに「-1」（FALSE）が表示されます。

```
$s1="Case"
$s2="case"


ENABLE UPPER
a= $s1==$s2
TYPE a
```

関連キーワード

DISABLE

DISABLE（プログラムコマンド）

ENABLE（モニタコマンド）

ENABLE (プログラムコマンド)

SWITCH(モニタコマンド)

SWITCH (プログラムコマンド)

SWITCH（実数値関数）

STRDIF

●本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
●本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

●製品に関するお問い合わせ先
お客様相談室
フリー通話 **0120-919-066**（クイック／オムロン）
携帯電話・PHS・IP電話などではご利用いただけませんので、下記の電話番号へおかけください。
電話 **055-982-5015**（通話料がかかります）
■営業時間：8:00～21:00　■営業日：365日
●FAXやWebページでもお問い合わせいただけます。
FAX 055-982-5051 / www.fa.omron.co.jp

●その他のお問い合わせ
納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。
オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧いただけます。
www.fa.omron.co.jp
緊急時のご購入にもご利用ください。

オムロン商品のご用命は

Man. No. SBCE-487C　2021年1月現在

© OMRON Corporation 2021　All Rights Reserved.
お断りなく仕様などを変更することがありますのでご了承ください