

EPSON

Epson RC+ 8.0 SPEL+ランゲージリファレンス (RC800シリーズ)

翻訳版

© Seiko Epson Corporation 2024-2025

Rev.4
JAM259S7669F

目次

1. はじめに	24
1.1 はじめに	25
1.2 商標	25
1.3 表記について	25
1.4 ご注意	25
1.5 製造元	25
1.6 お問い合わせ先	25
1.7 ご使用の前に	26
1.7.1 安全上の注意	26
1.7.2 記号の意味	26
1.7.3 Epson RC+ 8.0のインストールフォルダーについて	26
2. SPEL+ コマンド一覧	27
2.1 SPEL+ コマンド一覧	28
3. SPEL+ ランゲージリファレンス	56
3.1 SPEL+ ランゲージリファレンス	57
3.2 演算子	58
3.3 !	59
3.3.1 !...! 並列処理	59
3.4 #	63
3.4.1 #define	63
3.4.2 #ifdef...#else...#endif	65
3.4.3 #ifndef...#endif	66
3.4.4 #include	67
3.4.5 #undef	68
3.5 A	69
3.5.1 AbortMotion	69
3.5.2 Abs関数	72
3.5.3 Accel	73
3.5.4 Accel関数	76
3.5.5 AccelMax関数	77
3.5.6 AccelR	78
3.5.7 AccelR関数	79

3.5.8 AccelS	80
3.5.9 AccelS関数	83
3.5.10 Acos関数	84
3.5.11 AgI関数	85
3.5.12 AgIToPls関数	86
3.5.13 AIO_In関数	87
3.5.14 AIO_InW関数	88
3.5.15 AIO_Out	89
3.5.16 AIO_Out関数	90
3.5.17 AIO_OutW	91
3.5.18 AIO_OutW関数	92
3.5.19 AIO_Set	93
3.5.20 AIO_Set関数	96
3.5.21 AIO_TrackingSet	97
3.5.22 AIO_TrackingStart	101
3.5.23 AIO_TrackingEnd	104
3.5.24 AIO_TrackingOn関数	105
3.5.25 Align関数	106
3.5.26 AlignECP関数	107
3.5.27 And 演算子	108
3.5.28 AOpen	110
3.5.29 Arc, Arc3	112
3.5.30 Arch	118
3.5.31 ArchClr	121
3.5.32 Arch関数	122
3.5.33 AreaCorrection関数	123
3.5.34 AreaCorrectionClr	125
3.5.35 AreaCorrectionDef関数	126
3.5.36 AreaCorrectionInv関数	127
3.5.37 AreaCorrectionOffset関数	128
3.5.38 AreaCorrectionSet	129
3.5.39 Arm	132
3.5.40 Arm関数	134
3.5.41 ArmCalib	135
3.5.42 ArmCalib関数	136

3.5.43 ArmCalibClr	137
3.5.44 ArmCalibDef関数	138
3.5.45 ArmCalibSet	139
3.5.46 ArmCalibSet関数	140
3.5.47 ArmClr	141
3.5.48 ArmDef関数	142
3.5.49 ArmSet	143
3.5.50 ArmSet関数	147
3.5.51 Asc関数	150
3.5.52 Asin関数	151
3.5.53 AtHome関数	152
3.5.54 Atan関数	153
3.5.55 Atan2関数	154
3.5.56 ATCLR	155
3.5.57 ATRQ	156
3.5.58 ATRQ関数	157
3.5.59 AutoLJM	158
3.5.60 AutoLJM関数	160
3.5.61 AutoOrientationFlag	161
3.5.62 AutoOrientationFlag関数	163
3.5.63 AutoSTM	164
3.5.64 AutoSTM関数	167
3.5.65 AvgSpeedClear	168
3.5.66 AvgSpeed	169
3.5.67 AvgSpeed関数	170
3.5.68 AvoidSingularity	171
3.5.69 AvoidSingularity関数	173
3.6 B	174
3.6.1 Base	174
3.6.2 BClr関数	175
3.6.3 BClr64関数	176
3.6.4 BGo	177
3.6.5 BMove	179
3.6.6 Boolean	181
3.6.7 BOpen	182

3.6.8 Box	184
3.6.9 Box関数	187
3.6.10 BoxClr	188
3.6.11 BoxDef関数	189
3.6.12 Brake	190
3.6.13 Brake関数	191
3.6.14 BSet関数	192
3.6.15 BSet64関数	193
3.6.16 BTst関数	194
3.6.17 BTst64関数	195
3.6.18 Byte	196
3.7 C	197
3.7.1 Calib	197
3.7.2 Call	199
3.7.3 CalPIs	201
3.7.4 CalPIs関数	203
3.7.5 ChDir	204
3.7.6 ChDisk	205
3.7.7 ChDrive	208
3.7.8 ChkCom関数	209
3.7.9 ChkNet関数	210
3.7.10 Chr\$関数	211
3.7.11 ClearHistory	212
3.7.12 ClearPoints	213
3.7.13 Close	214
3.7.14 CloseCom	215
3.7.15 CloseDB	216
3.7.16 CloseNet	217
3.7.17 Cls	218
3.7.18 Cnv_AbortTrack	219
3.7.19 Cnv_Accel	220
3.7.20 Cnv_Accel関数	221
3.7.21 Cnv_AccelLim	222
3.7.22 Cnv_AccelLim関数	223
3.7.23 Cnv_Adjust	224

3.7.24 Cnv_AdjustClear	225
3.7.25 Cnv_AdjustGet関数	226
3.7.26 Cnv_AdjustSet	227
3.7.27 Cnv_Downstream	228
3.7.28 Cnv_Downstream関数	229
3.7.29 Cnv_Fine	230
3.7.30 Cnv_Fine関数	231
3.7.31 Cnv_Flag関数	232
3.7.32 Cnv_LPulse関数	233
3.7.33 Cnv_Mode	234
3.7.34 Cnv_Mode関数	235
3.7.35 Cnv_Name\$関数	236
3.7.36 Cnv_Number関数	237
3.7.37 Cnv_OffsetAngle	238
3.7.38 Cnv_OffsetAngle関数	239
3.7.39 Cnv_Point関数	240
3.7.40 Cnv_PosErr関数	241
3.7.41 Cnv_PosErrOffset	242
3.7.42 Cnv_Pulse関数	243
3.7.43 Cnv_QueueAdd	244
3.7.44 Cnv_QueueGet関数	245
3.7.45 Cnv_QueueLen関数	247
3.7.46 Cnv_QueueList	248
3.7.47 Cnv_QueueMove	249
3.7.48 Cnv_QueueReject	250
3.7.49 Cnv_QueueReject関数	251
3.7.50 Cnv_QueueRemove	252
3.7.51 Cnv_QueueUserData	253
3.7.52 Cnv_QueueUserData関数	254
3.7.53 Cnv_RobotConveyor関数	255
3.7.54 Cnv_Speed関数	256
3.7.55 Cnv_Trigger	257
3.7.56 Cnv_Upstream	258
3.7.57 Cnv_Upstream関数	259
3.7.58 CollisionDetect	260

3.7.59 CollisionDetect関数	262
3.7.60 Cont	263
3.7.61 ContManualRecover	264
3.7.62 Copy	265
3.7.63 Cos関数	266
3.7.64 CP	267
3.7.65 CP関数	269
3.7.66 CP_Offset	270
3.7.67 CP_Offset関数	273
3.7.68 Ctr関数	274
3.7.69 CTRreset	275
3.7.70 CtrlDev関数	276
3.7.71 CtrlInfo関数	277
3.7.72 CtrlPref関数	280
3.7.73 CurDir\$関数	282
3.7.74 CurDisk\$関数	283
3.7.75 CurDrive\$関数	284
3.7.76 CurPos関数	285
3.7.77 Curve	286
3.7.78 CVMove	294
3.7.79 CX, CY, CZ, CU, CV, CW, CR, CS, CT	296
3.7.80 CX, CY, CZ, CU, CV, CW, CR, CS, CT関数	297
3.8 D	298
3.8.1 Date	298
3.8.2 Date\$関数	299
3.8.3 Declare	300
3.8.4 DegToRad関数	303
3.8.5 Del	304
3.8.6 DeleteDB	305
3.8.7 DiffPoint関数	306
3.8.8 DiffToolOrientation関数	307
3.8.9 DispDev	308
3.8.10 DispDev関数	309
3.8.11 Dist関数	310
3.8.12 Do...Loop	311

3.8.13 Double	313
3.9 E	314
3.9.1 ECP	314
3.9.2 ECP関数	315
3.9.3 ECPClr	316
3.9.4 ECPDef関数	317
3.9.5 ECPSet	318
3.9.6 ECPSet関数	319
3.9.7 ElapsedTime関数	320
3.9.8 Elbow	321
3.9.9 Elbow関数	323
3.9.10 EncTemper	324
3.9.11 EncTemper関数	325
3.9.12 Eof関数	326
3.9.13 Era関数	327
3.9.14 EResume	329
3.9.15 Erf\$関数	330
3.9.16 Erl関数	331
3.9.17 Err関数	332
3.9.18 Errb関数	334
3.9.19 ErrMsg\$関数	335
3.9.20 Error	336
3.9.21 ErrorOn関数	337
3.9.22 Ert関数	338
3.9.23 EStopOn関数	339
3.9.24 Eval関数	340
3.9.25 Exit	341
3.9.26 ExportPoints	342
3.10 F	343
3.10.1 FbusIO_GetBusStatus関数	343
3.10.2 FbusIO_GetDeviceStatus関数	344
3.10.3 FbusIO_SendMsg	345
3.10.4 FileDateTime\$関数	347
3.10.5 FileExists関数	348
3.10.6 FileLen関数	349

3.10.7 Find	350
3.10.8 FindPos関数	352
3.10.9 Fine	353
3.10.10 Fine関数	355
3.10.11 FineDist	356
3.10.12 FineStatus関数	358
3.10.13 Fix関数	359
3.10.14 Flush	360
3.10.15 FmtStr	361
3.10.16 FmtStr\$関数	365
3.10.17 FolderExists関数	369
3.10.18 For...Next	370
3.10.19 FreeFile関数	372
3.10.20 Function...Fend	373
3.11 G	376
3.11.1 GUIVer\$	376
3.11.2 GetIODef	377
3.11.3 GetProjectInfo	378
3.11.4 GetSetNet	379
3.11.5 GetRobotInsideBox関数	380
3.11.6 GetRobotInsidePlane関数	381
3.11.7 Global	382
3.11.8 Go	384
3.11.9 GoSub...Return	388
3.11.10 GoTo	390
3.12 H	391
3.12.1 Halt	391
3.12.2 Hand	393
3.12.3 Hand関数	395
3.12.4 HealthCalcPeriod	396
3.12.5 HealthCalcPeriod関数	397
3.12.6 HealthCtrlAlarmOn関数	398
3.12.7 HealthCtrlInfo	399
3.12.8 HealthCtrlInfo関数	400
3.12.9 HealthCtrlRateOffset	401

3.12.10 HealthCtrlReset	402
3.12.11 HealthCtrlWarningEnable	403
3.12.12 HealthCtrlWarningEnable関数	404
3.12.13 HealthRateCtrlInfo関数	405
3.12.14 HealthRateRBInfo関数	406
3.12.15 HealthRBAAlarmOn関数	408
3.12.16 HealthRBAAnalysis関数	409
3.12.17 HealthRBDistance	411
3.12.18 HealthRBDistance関数	412
3.12.19 HealthRBInfo	413
3.12.20 HealthRBInfo関数	415
3.12.21 HealthRBRateOffset	417
3.12.22 HealthRBReset	418
3.12.23 HealthRBSpeed	419
3.12.24 HealthRBSpeed関数	420
3.12.25 HealthRBStart	421
3.12.26 HealthRBAAnalysis	422
3.12.27 HealthRBStop	424
3.12.28 HealthRBTRQ	425
3.12.29 HealthRBTRQ関数	426
3.12.30 HealthRBWarningEnable	427
3.12.31 HealthRBWarningEnable関数	428
3.12.32 Here	429
3.12.33 Here関数	431
3.12.34 Hex\$関数	432
3.12.35 History	433
3.12.36 Hofs	434
3.12.37 Hofs関数	436
3.12.38 HofsJointAccuracy	437
3.12.39 Home	439
3.12.40 HomeClr	440
3.12.41 HomeDef関数	441
3.12.42 HomeSet	442
3.12.43 HomeSet関数	444
3.12.44 Hordr	445

3.12.45 Hordr関数	447
3.12.46 Hour	448
3.12.47 Hour関数	449
3.13 I	450
3.13.1 If...Then...Else...EndIf	450
3.13.2 ImportPoints	453
3.13.3 In関数	454
3.13.4 InBCD関数	456
3.13.5 Inertia	458
3.13.6 Inertia関数	460
3.13.7 InPos関数	461
3.13.8 Input	462
3.13.9 Input #	464
3.13.10 InputBox	466
3.13.11 InReal関数	468
3.13.12 InsideBox関数	469
3.13.13 InsidePlane関数	471
3.13.14 InStr関数	473
3.13.15 Int関数	474
3.13.16 Int32	475
3.13.17 Int64	476
3.13.18 Integer	477
3.13.19 InW関数	478
3.13.20 IODef関数	479
3.13.21 IOLabel\$関数	480
3.13.22 IONumber関数	481
3.14 J	482
3.14.1 J1Angle	482
3.14.2 J1Angle関数	483
3.14.3 J1Flag	484
3.14.4 J1Flag関数	485
3.14.5 J2Flag	486
3.14.6 J2Flag関数	487
3.14.7 J4Angle	488
3.14.8 J4Angle関数	489

3.14.9 J4Flag	490
3.14.10 J4Flag関数	491
3.14.11 J6Flag	492
3.14.12 J6Flag関数	493
3.14.13 JA関数	494
3.14.14 JogPanel	495
3.14.15 Joint	497
3.14.16 JointAccuracy	498
3.14.17 JointAccuracy関数	500
3.14.18 JRange	501
3.14.19 JRange関数	502
3.14.20 JS関数	503
3.14.21 JT関数	504
3.14.22 JTran	505
3.14.23 Jump	506
3.14.24 Jump3, Jump3CP	511
3.14.25 JumpTLZ	516
3.15 L	520
3.15.1 LatchEnable	520
3.15.2 LatchState関数	521
3.15.3 LatchPos関数	522
3.15.4 LCase\$関数	524
3.15.5 Left\$関数	525
3.15.6 Len関数	526
3.15.7 LimitTorque	527
3.15.8 LimitTorque関数	529
3.15.9 LimitTorqueLP	530
3.15.10 LimitTorqueLP関数	532
3.15.11 LimitTorqueStop	533
3.15.12 LimitTorqueStop関数	534
3.15.13 LimitTorqueStopLP	535
3.15.14 LimitTorqueStopLP関数	536
3.15.15 LimZ	537
3.15.16 LimZ関数	538
3.15.17 LimZMargin	539

3.15.18 LimZMargin関数	540
3.15.19 Line Input	541
3.15.20 Line Input #	542
3.15.21 LJM関数	543
3.15.22 LoadPoints	548
3.15.23 Local	550
3.15.24 Local関数	553
3.15.25 LocalClr	554
3.15.26 LocalDef関数	555
3.15.27 Lof関数	556
3.15.28 LogIn関数	557
3.15.29 Long	558
3.15.30 LSet\$関数	559
3.15.31 LShift関数	560
3.15.32 LShift64関数	561
3.15.33 LTrim\$関数	562
3.16 M	563
3.16.1 Mask 演算子	563
3.16.2 MCal	564
3.16.3 MCalComplete関数	565
3.16.4 MCordr	566
3.16.5 MCordr関数	569
3.16.6 MemIn関数	570
3.16.7 MemInW関数	572
3.16.8 MemOff	573
3.16.9 MemOn	575
3.16.10 MemOut	577
3.16.11 MemOutW	580
3.16.12 MemSw関数	581
3.16.13 MHour関数	582
3.16.14 Mid\$関数	583
3.16.15 MkDir	584
3.16.16 Mod 演算子	585
3.16.17 Motor	586
3.16.18 Motor関数	587

3.16.19 Move	588
3.16.20 MsgBox	591
3.16.21 MyTask関数	594
3.17 N	595
3.17.1 Next	595
3.17.2 Not 演算子	597
3.18 O	598
3.18.1 Off	598
3.18.2 OLAcel	600
3.18.3 OLAcel関数	602
3.18.4 OLRate	603
3.18.5 OLRate関数	604
3.18.6 On	605
3.18.7 OnErr	607
3.18.8 OpBCD	609
3.18.9 OpenDB	612
3.18.10 OpenCom	614
3.18.11 OpenCom関数	615
3.18.12 OpenNet	616
3.18.13 OpenNet関数	617
3.18.14 Oport関数	618
3.18.15 Or 演算子	620
3.18.16 Out	621
3.18.17 Out関数	624
3.18.18 OutReal	625
3.18.19 OutReal関数	626
3.18.20 OutW	627
3.18.21 OutW関数	628
3.19 P	629
3.19.1 P# (1. ポイント定義)	629
3.19.2 P# (2. ポイント指定)	631
3.19.3 PAgl関数	634
3.19.4 Pallet	635
3.19.5 Pallet関数	639
3.19.6 PalletClr	641

3.19.7 ParseStr / ParseStr関数	642
3.19.8 Pass	643
3.19.9 Pause	645
3.19.10 PauseOn関数	646
3.19.11 PDef関数	647
3.19.12 PDel	648
3.19.13 PDescription	649
3.19.14 PDescription\$関数	650
3.19.15 PeakSpeedClear	651
3.19.16 PeakSpeed	652
3.19.17 PeakSpeed関数	653
3.19.18 PerformMode	654
3.19.19 PerformMode関数	656
3.19.20 PG_FastStop	657
3.19.21 PG_LSpeed	658
3.19.22 PG_LSpeed関数	660
3.19.23 PG_Scan	661
3.19.24 PG_SlowStop	662
3.19.25 PLabel	663
3.19.26 PLabel\$関数	664
3.19.27 Plane	665
3.19.28 Plane関数	668
3.19.29 PlaneClr	669
3.19.30 PlaneDef関数	670
3.19.31 PList	671
3.19.32 PLocal	673
3.19.33 PLocal関数	674
3.19.34 PIs関数	675
3.19.35 PNumber関数	676
3.19.36 PosFound関数	677
3.19.37 Power	678
3.19.38 Power関数	680
3.19.39 PPIs関数	681
3.19.40 Print	682
3.19.41 Print #	684

3.19.42 ProjectName\$ 関数	686
3.19.43 PTCLR	687
3.19.44 PTPBoost	688
3.19.45 PTPBoost関数	689
3.19.46 PTPBoostOK関数	690
3.19.47 PTPTIME関数	691
3.19.48 PTran	692
3.19.49 PTRQ	693
3.19.50 PTRQ関数	694
3.19.51 Pulse	695
3.19.52 Pulse関数	697
3.20 Q	698
3.20.1 QP	698
3.20.2 QPDecelR	700
3.20.3 QPDecelR関数	702
3.20.4 QPDecelS	703
3.20.5 QPDecelS関数	705
3.20.6 Quit	706
3.21 R	708
3.21.1 RadToDeg関数	708
3.21.2 Randomize	709
3.21.3 Range	710
3.21.4 Read	712
3.21.5 ReadBin	713
3.21.6 Real	714
3.21.7 RealAccel関数	715
3.21.8 RealPls関数	717
3.21.9 RealPos関数	718
3.21.10 RealTorque関数	719
3.21.11 Recover	720
3.21.12 Recover関数	722
3.21.13 RecoverPos関数	724
3.21.14 Redim	725
3.21.15 Rename	727
3.21.16 RenDir	728

3.21.17 Reset	729
3.21.18 ResetElapsedTime	731
3.21.19 Restart	732
3.21.20 Resume	733
3.21.21 Return	734
3.21.22 Right\$関数	735
3.21.23 Rmdir	736
3.21.24 Rnd関数	737
3.21.25 Robot	738
3.21.26 Robot関数	739
3.21.27 RobotInfo関数	740
3.21.28 RobotInfo\$関数	743
3.21.29 RobotModel\$関数	744
3.21.30 RobotName\$関数	745
3.21.31 RobotSerial\$関数	746
3.21.32 RobotType関数	747
3.21.33 ROpen	748
3.21.34 ROTOK関数	749
3.21.35 RSet\$関数	750
3.21.36 RShift関数	751
3.21.37 RShift64関数	752
3.21.38 RTrim\$関数	753
3.21.39 RunDialog	754
3.22 S	755
3.22.1 SafetyOn関数	755
3.22.2 SavePoints	756
3.22.3 Seek	757
3.22.4 Select...Send	758
3.22.5 SelectDB関数	759
3.22.6 Sense	761
3.22.7 SetCom	763
3.22.8 SetLatch	765
3.22.9 SetIn	767
3.22.10 SetInReal	768
3.22.11 SetInW	769

3.22.12 SetIODef	770
3.22.13 SetNet	771
3.22.14 SetSw	772
3.22.15 SF_GetParam関数	773
3.22.16 SF_GetParam\$関数	781
3.22.17 SF_GetStatus関数	782
3.22.18 SF_LimitSpeedS	788
3.22.19 SF_LimitSpeedS関数	792
3.22.20 SF_LimitSpeedSEnable	793
3.22.21 SF_LimitSpeedSEnable関数	795
3.22.22 SF_PeakSpeedS	796
3.22.23 SF_PeakSpeedS関数	797
3.22.24 SF_PeakSpeedSClear	798
3.22.25 SF_RealSpeedS	799
3.22.26 SF_RealSpeedS関数	800
3.22.27 SF_SetIn	801
3.22.28 SF_SetSw	802
3.22.29 SFree	803
3.22.30 SFree関数	805
3.22.31 Sgn関数	806
3.22.32 Short	807
3.22.33 Signal	808
3.22.34 SimGet	809
3.22.35 SimSet	812
3.22.36 Sin関数	817
3.22.37 SingularityAngle	818
3.22.38 SingularityAngle関数	819
3.22.39 SingularityDist	820
3.22.40 SingularityDist関数	821
3.22.41 SingularitySpeed	822
3.22.42 SingularitySpeed関数	823
3.22.43 SLock	824
3.22.44 SoftCP	825
3.22.45 SoftCP関数	826
3.22.46 Space\$関数	827

3.22.47 Speed	828
3.22.48 Speed関数	830
3.22.49 SpeedFactor	831
3.22.50 SpeedFactor関数	832
3.22.51 SpeedR	833
3.22.52 SpeedR関数	834
3.22.53 SpeedRLimitation	835
3.22.54 SpeedRLimitation関数	837
3.22.55 SpeedS	838
3.22.56 SpeedS関数	840
3.22.57 Sqr関数	841
3.22.58 ST関数	842
3.22.59 STM関数	843
3.22.60 StartMain	846
3.22.61 Stat関数	847
3.22.62 Str\$関数	850
3.22.63 String	851
3.22.64 Sw関数	853
3.22.65 SyncLock	854
3.22.66 SyncUnlock	856
3.22.67 SyncRobots	857
3.22.68 SyncRobots関数	858
3.22.69 SysConfig	859
3.22.70 SysErr関数	861
3.23 T	863
3.23.1 Tab\$関数	863
3.23.2 Tan関数	864
3.23.3 TargetOK関数	865
3.23.4 TaskDone関数	866
3.23.5 TaskInfo関数	867
3.23.6 TaskInfo\$関数	869
3.23.7 TaskState関数	870
3.23.8 TaskWait	871
3.23.9 TC	872
3.23.10 TCLim	875

3.23.11 TCLim関数	877
3.23.12 TCPSpeed関数	878
3.23.13 TCSpeed	879
3.23.14 TCSpeed関数	881
3.23.15 TeachOn関数	882
3.23.16 TeachPoint	883
3.23.17 TGo	885
3.23.18 Till	887
3.23.19 TillOn関数	889
3.23.20 Time	890
3.23.21 Time関数	891
3.23.22 Time\$関数	892
3.23.23 TLClr	893
3.23.24 TLDef関数	894
3.23.25 TLSet	895
3.23.26 TLSet関数	899
3.23.27 TMOut	900
3.23.28 TMove	901
3.23.29 Tmr関数	903
3.23.30 TmReset	904
3.23.31 Toff	905
3.23.32 Ton	906
3.23.33 Tool	907
3.23.34 Tool関数	908
3.23.35 ToolWizard関数	909
3.23.36 Trap (ユーザー定義トリガー)	910
3.23.37 Trap (システム状態トリガー)	914
3.23.38 Trim\$関数	917
3.23.39 TW関数	918
3.24 U	919
3.24.1 UBound関数	919
3.24.2 UByte	920
3.24.3 UCase\$関数	921
3.24.4 UInt32	922
3.24.5 UInt64	923

3.24.6 UOpen	924
3.24.7 UpdateDB	925
3.24.8 UserErrorDef関数	926
3.24.9 UserErrorLabel\$関数	927
3.24.10 UserErrorNumber関数	928
3.24.11 UShort	929
3.25 V	930
3.25.1 Val関数	930
3.25.2 VSD	931
3.25.3 VSD関数	932
3.25.4 VxCalib	933
3.25.5 VxCalDelete	938
3.25.6 VxCalLoad	939
3.25.7 VxCalInfo関数	940
3.25.8 VxCalSave	941
3.25.9 VxTrans関数	942
3.26 W	943
3.26.1 Wait	943
3.26.2 WaitNet	946
3.26.3 WaitPos	947
3.26.4 WaitSig	948
3.26.5 Weight	949
3.26.6 Weight関数	951
3.26.7 Where	952
3.26.8 WindowsStatus関数	953
3.26.9 WOpen	954
3.26.10 WorkQue_Add	956
3.26.11 WorkQue_AutoRemove	957
3.26.12 WorkQue_AutoRemove関数	958
3.26.13 WorkQue_Get関数	959
3.26.14 WorkQue_Len関数	960
3.26.15 WorkQue_List	961
3.26.16 WorkQue_Reject	962
3.26.17 WorkQue_Reject関数	963
3.26.18 WorkQue_Remove	964

3.26.19 WorkQue_Sort	965
3.26.20 WorkQue_Sort関数	966
3.26.21 WorkQue_UserData	967
3.26.22 WorkQue_UserData関数	968
3.26.23 Wrist	969
3.26.24 Wrist関数	970
3.26.25 Write	971
3.26.26 WriteBin	972
3.27 X	973
3.27.1 Xor 演算子	973
3.27.2 Xqt	974
3.27.3 XY関数	982
3.27.4 XYLim	983
3.27.5 XYLim関数	985
3.27.6 XYLimClr	986
3.27.7 XYLimDef関数	987
3.27.8 XYLimMode	988
3.27.9 XYLimMode関数	990
4. Appendix A: SPEL+ コマンド 使用条件一覧	991
4.1 Appendix A: SPEL+ コマンド 使用条件一覧	992
5. Appendix B: 互換性に関する注意	1013
5.1 B-1: EPSON RC+ 6.0 からの互換性に関する注意	1014
5.1.1 概要	1014
5.1.2 全般にわたる違い	1014
5.1.3 命令の互換性一覧	1016
5.2 B-2: EPSON RC+ 5.0 からの互換性に関する注意	1030
5.2.1 概要	1030
5.2.2 全般にわたる違い	1030
5.2.3 命令の互換性一覧	1032
5.2.4 EPSON RC+ Ver.4.*からの移行コマンド一覧 (EPSON RC+5.0未サポート)	1045
6. Appendix C: Epson RC+8.0のコマンド	1047
6.1 C-1: EPSON RC+ 4.0以降で追加されたコマンド一覧	1048
6.2 C-2: Epson RC+ 8.0のバージョンごとに追加されたコマンド一覧	1053
6.3 C-3: EPSON RC+ 7.0のバージョンごとに追加されたコマンド一覧	1054

6.4 C-4: EPSON RC+ 7.0のバージョンごとに削除されたコマンド一覧	1060
7. Appendix D: 定数	1061
7.1 Appendix D: 定数	1062

1. はじめに

1.1 はじめに

このたびは当社のロボットシステムをお求めいただきましてありがとうございます。

本マニュアルは、Epson RC+ ソフトウェアを正しくお使いいただくために必要な事項を記載したものです。

システムをご使用になる前に、本マニュアルおよび関連マニュアルをお読みいただき、正しくお使いください。

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

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

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

1.2 商標

Microsoft, Windows, Windowsロゴ, Visual Basic, Visual C++は、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。

Pentiumは米国インテル社の商標です。

その他の社名, ブランド名, 製品名は、各社の登録商標または商標です。

1.3 表記について

- Microsoft® WindowsR 10 operating system 日本語版
- Microsoft® WindowsR 11 operating system 日本語版

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

1.4 ご注意

本取扱説明書の一部、または全部を無断で複製や転載をすることはできません。

本書に記載の内容は、将来予告なく変更することがあります。

本書の内容について、誤りや、お気づきの点がありましたら、ご連絡くださいますようお願いいたします。

1.5 製造元

セイコーエプソン株式会社

1.6 お問い合わせ先

お問い合わせ先の詳細は、以下のマニュアルの"販売元"に記載しています。

ご利用の地域によって、お問い合わせ先が異なりますのでご注意ください。

"安全マニュアル - お問い合わせ先"

安全マニュアルは、以下のサイトからも閲覧できます。

URL: <https://download.epson.biz/robots/>



1.7 ご使用の前に

マニュアルのご使用の前に、知っておいていただきたいことを記載しています。

1.7.1 安全上の注意

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

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

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

1.7.2 記号の意味

警告

この表示を無視して誤った取り扱いをすると、人が死亡、または重傷を負う可能性が想定される内容を示しています。

注意

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

1.7.3 Epson RC+ 8.0のインストールフォルダーについて

Epson RC+ 8.0は、インストールフォルダーパスを任意の場所に変更が可能です。本マニュアルでは、`C:\EpsonRC80` にEpson RC+ 8.0がインストールされた場合を想定して説明しています。

2. SPEL+ コマンド一覧

2.1 SPEL+ コマンド一覧

SPEL+ コマンド一覧です。

システム管理に関する命令

コマンド	説明
Reset	コントローラーを初期状態にリセット
SysConfig	システム設定パラメーターを表示
CtrlPref	Epson RC+ のシステム設定のコントローラ環境設定情報を取得
SysErr	最新のエラーステータス、またはワーニングステータスを返す
GUIVer\$	接続中のEpson RC+のバージョンを取得
GetProjectInfo	Epson RC+で開いているプロジェクト名とプロジェクトフォルダパスを取得
Date	日付の表示
Time	時刻の表示
Date\$	日付を文字列で返す関数
Time\$	時刻を文字列で返す関数
Time関数	コントローラーの積算通電時間の表示
Hour	コントローラーの積算通電時間の表示
Stat	コントローラーの状態ビットを返す
CtrlInfo	コントローラーの情報を返す
RobotInfo	ロボットの情報を返す
RobotInfo\$	ロボットのテキスト情報を返す
TaskInfo	タスク情報を返す
TaskInfo\$	タスクのテキスト情報を返す
DispDev	現在のディスプレイデバイスを設定
EStopOn	非常停止状態を返す
CtrlDev	現在の制御デバイスの番号を返す
Cls	Runウィンドウ, オペレーターウィンドウ, コマンドウィンドウのテキスト領域をクリア、TPのプリントパネルをクリア

コマンド	説明
Toff	LCDに実行行の表示をオフ
Ton	LCDに実行行の表示をオン
SafetyOn	安全扉の状態を返す
Eval	コマンドウィンドウのステートメントを実行し、エラー状態を返す
TeachOn	ティーチモードの状態を返す
WindowsStatus	Windowsの起動状態を返す
History	コマンドの実行履歴の表示
ClearHistory	コマンドの実行履歴の全削除

ロボット制御に関する命令

コマンド	説明
AIO_TrackingSet	距離追従機能の設定
AIO_TrackingStart	距離追従機能の開始
AIO_TrackingEnd	距離追従機能の終了
AIO_TrackingOn関数	距離追従機能の状態を返す関数
AtHome	現在のロボット姿勢がHome姿勢か否かを返す関数
Calib	現在のアーム姿勢のパルス値を、Calplsにより設定されているパルス値に置き換える
CalPls	キャリブレーションに使用する位置姿勢パルス値の設定と表示、または設定値を返す関数
Hofs	エンコーダー原点からソフトウェア原点への補正パルスの設定と表示
JointAccuracy	関節精度補正の補正值の設定と表示
HofsJointAccuracy	関節精度補正を変更せずに、エンコーダー原点からソフトウェア原点への補正パルスを設定
MCal	インクリメンタルエンコーダーロボットの原点復帰（メカ原点の検出）
MCalComplete	MCalの状態を返す
MCordr	MCal命令による原点復帰時の関節の動作順序の指定と表示、または設定値を返す関数
EncTemper	エンコーダー温度を表示します

コマンド	説明
EncTemper関数	エンコーダー温度を返す関数
Power	パワーモードをHighまたはLowにし、現在のモードを表示
MHour関数	ロボットのモーターの積算励磁時間を返す関数
Motor	ロボットの全軸のモーターオンまたはオフ
SFree	指定した関節をフリージョイント状態にします
SLock	指定された関節のフリージョイント状態を解除します
SyncRobots	動作予約中のロボットの動作を開始
Jump	目標座標にゲートモーションで移動する動作 (PTP動作)
Jump3	目標座標に3次元ゲート動作で移動する動作 (PTP動作+CP動作)
Jump3CP	目標座標に3次元ゲート動作で移動する動作 (CP動作)
JumpTLZ	目標座標に3次元ゲート動作で移動する動作 (PTP動作+CP動作)
Arch	Jump命令でのアーチパラメーターの設定と表示、および設定値を返す関数
ArchClr	アーチパラメーターの設定を初期値に戻します
LimZ	Jumpコマンドのときの第3関節の高さ (Z座標値)の初期値を設定、および設定値を返す関数
LimZMargin	LimZ設定値より高い位置で動作開始したときのエラー検出のマージンを設定、および設定値を返す関数
Sense	Jump命令でSense指定したときに、目標座標上方で停止させるための条件の設定と表示
JS	Jump Sense実行後、Sense入力が成立したのかどうかを返す関数
JT	直前のJump動作の結果を返す関数
Go	現在位置から指定位置までPTP動作で動かす
Pass	指定点の近傍を停止しないで通過するPTP動作
Pulse	各関節のパルス値で指定されたポイントまで、アームをPTP制御で動かす
BGo	選択したローカル座標系でのオフセットPTP動作
BMove	ローカル座標系でのオフセット直線補間動作
TGo	ツール座標系上でのオフセットPTP動作

コマンド	説明
TMove	ツール座標系上でのオフセット直線補間動作
Till	動作命令でTill指定したときに、動作途中で停止し、処理を終了させる条件の設定と表示
TillOn	Tillの状態を返す
!...!	動作中にI/Oなどの入出力処理を並列して行う機能
Speed	PTP動作速度の設定と表示、および設定値を返す関数
Accel	PTP動作の加減速度の設定と表示、および設定値を返す関数
SpeedFactor	PTP動作速度の設定と表示、および設定値を返す関数
Inertia	ロボットの負荷イナーシャと偏心量を設定
Weight	PTP動作時の、速度/加減速度を補正するためのパラメーター設定と表示
Arc	XY平面における円弧補間動作 (CP動作)
Arc3	3次元における円弧補間動作 (CP動作)
Move	直線補間動作 (CP動作)
Curve	CVMove命令による自由曲線CP動作で使用する動作軌跡のファイルを作成
CVMove	Curve命令で定義した自由曲線CP動作を実行
SpeedS	CP動作速度の設定と表示、および設定値を返す関数
AccelS	CP動作の加減速度の設定と表示、および設定を返す関数
SpeedR	ツール姿勢変化の速度の設定と表示、および設定値を返す関数
AccelR	ツール姿勢変化の加減速度の設定と表示、および設定値を返す関数
SpeedRLimitation	ツール姿勢変化速度による動作速度制限の設定と表示、および設定値を返す関数
AccelMax	Accelで設定可能な加減速度の最大値を返す関数
Brake	現在のロボットの指定した関節のブレーキのオンオフ
Home	ホーム位置 (待機姿勢) への移動 (PTP動作)
HomeClr	ホーム位置の定義をクリア
HomeDef	ホーム位置の定義を返す

コマンド	説明
HomeSet	ホーム位置 (待機位置) のパルスの設定と表示
Hordr	Home実行時の各関節の動作順序の指定と表示、および設定値を返す関数
InPos	ロボットの位置決め状態を返す関数
CurPos	ロボットの現在の動作目標位置を返す
TCPSpeed	算出された現在のツール先端速度を返す
Pallet	パレットの定義と定義パレットの表示
PalletClr	パレットの定義をクリア
Fine	目標位置の位置決め終了判断範囲の設定と表示 (単位: pulse)
FineDist	目標位置の位置決め終了判断範囲の設定と表示 (単位: mm)
FineStatus関数	Fineを使用しているかFineDistを使用しているかを整数値で返す
QP	クイックポーズ機能の設定/解除と、現在の設定の表示、および設定値を返す関数
QPDcelR	CP動作時のツール姿勢変化について、クイックポーズ減速度を設定
QPDcelS	CP動作時のクイックポーズ減速度を設定
CP	パスモーションを設定
Box	進入検出エリアの設定と表示
BoxClr	設定されたBoxをクリア
BoxDef	進入検出エリアが設定されているかどうかを返す
Plane	進入検出平面の設定と表示
PlaneClr	進入検出平面をクリア (未定義)
PlaneDef	進入検出平面の設定状態を返す
InsideBox	進入検出エリアの検出状態を返す
InsidePlane	進入検出平面の検出状態を返す
GetRobotInsideBox	進入検出エリアに進入しているロボットを返す
GetRobotInsidePlane	進入検出平面に進入しているロボットを返す
Find	動作命令中に座標を保存するための条件の設定と表示

コマンド	説明
FindPos	動作命令実行中にFindで保存した座標を返す
PosFound	Find命令時実行の状態を返す
WaitPos	パスモーションが有効な状態でも、次のステートメントを実行する前にロボットが減速停止するのを待つ
Robot	ロボットの選択 (Robot関数でロボット番号を返す)
RobotModel\$	ロボットの型名を返す
RobotName\$	ロボット名を返す
RobotSerial\$	ロボットのシリアル番号を返す
RobotType	ロボットタイプを返す
TargetOK	現在位置から目標座標へのPTP (point to point) 動作が可能かどうかを返す
ROTOk関数	目標座標への動作命令時に、ROT 修飾パラメーターが付加可能かどうかを返す
JRange	パルス値による指定関節の許容動作エリアの設定、および設定値を返す関数
Range	パルス値による許容動作エリアの設定と表示
XYLim	XY平面許容動作エリアの設定と表示、および設定値を返す関数
XYLimClr	設定されたXYLimをクリア
XYLimDef	XYLimを設定したかどうかを返す
XYLimMode	XYLimの監視方法の設定と表示、および設定値を返す関数
XY	指定された座標値をポイントデータとして返す
Dist	2つのロボット座標間の距離を返す
DiffToolOrientation関数	ツール座標系の各座標軸のなす角度を返す
DiffPoint関数	指定された2つのポイントの差分を返す
PTPBoost	PTP動作微小移動時の設定と表示、および設定値を返す関数
PTPBoostOK	現在位置から目標座標へのPTP (point to point) 動作が微小移動であるかどうかを返す
PTPTime	PTP動作命令の推定所要時間を返す、PTP動作は行なわない
CX	指定したポイントデータのX座標値の設定と設定値を返す関数

コマンド	説明
CY	指定したポイントデータのY座標値の設定と設定値を返す関数
CZ	指定したポイントデータのZ座標値の設定と設定値を返す関数
CU	指定したポイントデータのU座標値の設定と設定値を返す関数
CV	指定したポイントデータのV座標値の設定と設定値を返す関数
CW	指定したポイントデータのW座標値の設定と設定値を返す関数
CR	指定したポイントデータのR座標値の設定と設定値を返す関数
CS	指定したポイントデータのS座標値の設定と設定値を返す関数
CT	指定したポイントデータのT座標値の設定と設定値を返す関数
Pls	各関節のパルス値を返す関数
Agl	指定関節の角度または位置を返す関数
PAgl	指定した座標値から関節位置 (回転関節の角度, 直動関節の位置)を計算して返す関数
JA	関節の角度から計算したロボット座標を返す
AglToPls	ロボットの各関節の角度をパルスに変換
DegToRad	角度をラジアンに変換
RadToDeg	ラジアンを角度に変換
Joint	現在のロボットの位置を、関節の座標系で表示
JTran	原点復帰を必要としない1関節のみのPTP動作を行う
PTran	現在位置から指定したパルス量を、1関節のみPTP動作
RealPls	指定した関節のパルス値を返す
RealPos	指定したロボットの現在の位置を返す
RealAccel関数	OLAccelで自動調整したAccelの値を返す関数
PPls	指定した座標値から関節パルスを計算して返す関数
LJM関数	指定ポイントに対して参照ポイントから見て最も関節移動量が少なくなるように姿勢フラグを変換したポイントデータを返す
AutoLJM	自動LJMを設定
AutoLJM関数	自動LJMの状態を返す関数

コマンド	説明
AutoOrientationFlag	N6-A1000**の姿勢フラグを変更
AutoOrientationFlag関数	AutoOrientationFlagの状態を返す関数
AutoSTM	自動STM機能を設定
AutoSTM関数	自動STM機能の状態を返す関数
STM関数	参照ポイントを開始点としたときに動作時間が最も短くなるように指定ポイントの姿勢フラグを変換したポイントデータを返す関数
AvoidSingularity	特異姿勢回避機能を設定
AvoidSingularity関数	特異姿勢回避機能の状態を返す関数
SingularityAngle	特異姿勢回避機能の特異姿勢近傍角度を設定
SingularityAngle関数	特異姿勢回避機能の特異姿勢近傍角度を返す関数
SingularitySpeed	特異姿勢回避機能の特異姿勢近傍速度を設定
SingularitySpeed関数	特異姿勢回避機能の特異姿勢近傍速度を返す関数
SingularityDist	特異姿勢通過機能に必要な特異点近傍距離を設定
SingularityDist関数	特異姿勢通過機能に必要な特異点近傍距離を返す関数
AbortMotion	動作命令を中断し動作を実行していたタスクをエラーにする
Align関数	ロボットの姿勢をローカル座標系の最も近い座標軸に整列するよう変換したポイントデータを返す関数
AlignECP関数	ロボットの姿勢をECP座標系の最も近い座標軸に整列するよう変換したポイントデータを返す関数
SoftCP	SoftCP動作モードの設定と表示
SoftCP関数	SoftCP動作モードの状態を返す関数
Here	現在位置をロボット座標として教示
Where	ロボットの現在位置データを表示
PerformMode	ロボットの動作モードを設定
PerformMode関数	ロボットの動作モード番号を返す
VSD	スカラロボットの可変速度CP動作機能を設定する

コマンド	説明
VSD関数	スカラロボットの可変速度CP動作機能の設定を返す
CP_Offset	CP On時の後続する動作命令の開始オフセット時間を設定
CP_Offset関数	CP On時の後続する動作命令の開始オフセット時間を返す
AvgSpeedClear	関節の平均速度をクリアし、初期化
AvgSpeed	関節の平均速度を表示
AvgSpeed関数	関節の平均速度を返す関数
PeakSpeedClear	関節のピーク速度をクリアし、初期化
PeakSpeed	関節のピーク速度を表示
PeakSpeed関数	関節のピーク速度を返す関数

トルクに関する命令

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

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

入出力に関する命令

コマンド	説明
On	出力ビットをオンします。
Off	出力ビットをオフします。
Oport	出力ビットの状態を読み込み
Sw	I/O入力またはメモリーI/Oの状態を読み込み、I/Oで使用します。
In	1バイト (8ビット)の入力データを読み込み、I/Oで使用します。
InW	1ワード (16ビット)の入力データを読み込み、I/Oで使用します。
InBCD	入力ポートの入力状態をBCD (バイナリーコード化された10進数)フォーマットで読み込み
Out	1バイト (8ビット)の出力データを出力、I/Oで使用します。
OutW	1ワード (16ビット)の出力データを出力、I/Oで使用します。
OpBCD	出力ポートにBCDフォーマットで1バイトのデータを出力
MemOn	ビット番号で指定されたメモリーI/Oをオンする命令
MemOff	ビット番号で指定されたメモリーI/Oをオフする命令
MemSw	指定されたメモリーI/Oビットの状態を返す関数
MemIn	メモリーI/Oの状態をバイト単位で返す関数
MemOut	メモリーI/Oの状態をバイト単位で変更する命令
MemInW	メモリーI/Oポートの状態をワード単位で返す、ワードポートは16個のメモリーI/Oビットで構成
MemOutW	メモリーI/Oポートの状態を16ビット同時にワード単位で設定
GetIODef	入力, 出力, メモリーI/Oのビット, バイト, ワードのI/Oラベルとコメントを取得

コマンド	説明
SetIODef	入力, 出力, メモリーI/Oのビット, バイト, ワードのI/Oラベルとコメントを設定
Wait	イベント条件待機
TMOut	Wait命令実行時のタイムアウト時間の設定
TW	Wait命令の条件式が成立したかどうかを返す関数
Input	ディスプレイデバイスから入力を受け取り、変数に保存
InReal	2ワード (32ビット)の入力データを32ビット浮動小数点データ(IEEE754準拠)として読み込みます。I/Oで使します。
Print	出力画面, コマンド画面, オペレーターウィンドウへのデータ表示
Line Input	1行入力データを読み、そのデータを文字列変数に代入
Input #	ファイル, 通信ポート, データベース、またはデバイスから、文字列か数値をデータ入力し、変数に保存
Print #	指定されたファイル, 通信ポート, データベース, デバイスにデータを出力
Line Input #	ファイル, 通信ポート, データベース, デバイスから、1行データを読み込み
Lof	指定したRS-232Cポート、またはTCP/IPportのバッファの受信データの行数を返す
SetIn	仮想I/Oの入力ポート (8ビット)を設定
SetInW	仮想I/Oの入力ポート (16ビット)を設定
SetSw	仮想I/Oの入力を設定
IOLabel\$	指定した入出力のビット, バイト, ワードのI/Oラベルを返す
IONumber	指定したI/OラベルのI/Oポート番号を返す
IODef	指定したI/Oラベルが定義されているかを返す
OpenCom	RS-232Cポートをオープン
OpenCom関数	OpenComを実施しているタスク番号を取得する関数
CloseCom	OpenComでオープンしたRS-232Cポートをクローズ
SetCom	RS-232Cポートのパラメーターを設定または表示
ChkCom	通信ポートの受信バッファ内の文字数を返す

コマンド	説明
OpenNet	TCP/IPネットワークポートをオープン
OpenNet関数	OpenNetを実施しているタスク番号を取得する関数
OutReal	実数値の出力データを32ビット浮動小数点データ (IEEE754準拠) として出力ポート2ワード (32ビット) に出力します。I/Oで使用します。
CloseNet	OpenNetでオープンしたTCP/IPポートをクローズ
SetNet	TCP/IPポートのパラメーターを設定
GetSetNet	全TCP/IPポートのパラメーターを取得
ChkNet	ネットワークポートの受信バッファ内の文字数を返す
WaitNet	TCP/IPポートの接続が確立するまで待つ
Read	ファイル、または通信ポートから指定した文字数を読み込む
ReadBin	ファイル、または通信ポートからバイナリーデータを読む
Write	ファイル、または通信ポートに文字列を書き込む、行末ターミネーターは付加しない
WriteBin	ファイル、または通信ポートへバイナリーデータを書き出す
InputDialog	入力ダイアログボックスを表示し、入力した内容を返す
MsgBox	ダイアログボックスのメッセージの表示
RunDialog	SPEL+のプログラムからEpson RC+画面を起動
JogPanel	SPEL+のプログラムからEpson RC+のジョグ&ティーチ画面を起動
TeachPoint	SPEL+のプログラムからEpson RC+のティーチポイント画面を起動
ToolWizard	SPEL+のプログラムからEpson RC+のツールウィザード画面を起動
LatchEnable	R-I/O入力によるロボット位置のラッチ機能を有効/無効にする
LatchState関数	R-I/Oによるロボット位置のラッチ状態を返す
LatchPos関数	R-I/O入力信号でラッチしたロボットの位置を返す
SetLatch	R-I/O入力によるロボット位置のラッチ機能を設定
AIO_In関数	アナログI/O入力チャンネルからアナログ値を読み込む
AIO_InW関数	アナログI/O入力チャンネルから1ワードの入力データを読み込む

コマンド	説明
AIO_Out	アナログI/O出力チャンネルにアナログ値を出力
AIO_Out関数	アナログI/O出力チャンネルの出力状態を返す
AIO_OutW	アナログI/O出力チャンネルに1ワードのデータを出力
AIO_OutW関数	アナログI/O出力チャンネルの出力状態を1ワードで返す
AIO_Set	アナログI/O出力チャンネルに速度情報を出力する
AIO_Set関数	オプションのアナログI/O出力チャンネルに設定されているロボット速度出力の設定情報を返す

ポイント制御に関する命令

コマンド	説明
ClearPoints	メモリー上の位置データを消去
LoadPoints	ポイントファイルを、ロボットのポイントメモリーエリアに読込
SavePoints	メインメモリーにあるポイントデータを、カレントロボットのディスクファイルに保存
ImportPoints	現在選択中のロボットのプロジェクトにポイントファイルをインポート
ExportPoints	ポイントファイルをPCの指定パスにエクスポート
P#	ポイントデータを定義
PDef	指定したポイントデータが定義されているかを返す
PDel	指定したポイントデータを削除
PLabel	指定したポイントデータのラベルを設定
PLabel\$	指定したポイント番号に定義されたポイントラベルを返す
PNumber	ポイントラベルに対応するポイント番号を返す
PList	現在のロボットのメモリー上のポイントデータを表示
PLocal	指定したポイントデータのローカル属性を設定
PDDescription	指定したポイントデータのコメントを設定
PDDescription\$	指定したポイント番号に定義されたポイントのコメントを返す
WorkQue_Add	指定したワークキューにワークキューデータ (ポイントデータとユーザーデータ) を追加

コマンド	説明
WorkQue_AutoRemove	指定したワークキューに自動削除機能を設定
WorkQue_AutoRemove関数	ワークキューに設定されている自動削除機能の状態を返す
WorkQue_Get関数	指定したワークキューから、ポイントデータを返す
WorkQue_Len関数	指定したワークキューに登録された有効なワークキューデータの数进行返す
WorkQue_List	指定したワークキューのワークキューデータ一覧 (ポイントデータとユーザーデータ)を表示
WorkQue_Reject	指定したワークキューにポイントデータの2重登録防止のための最小距離を設定、表示
WorkQue_Reject関数	指定したワークキューに設定されている2重登録防止距離を返す
WorkQue_Remove	指定したワークキューからワークキューデータ (ポイントデータとユーザーデータ)を削除
WorkQue_Sort	指定したワークキューにSort方法を設定、表示
WorkQue_Sort関数	指定したワークキューに設定されているSort方法を返す
WorkQue_UserData	指定したワークキューに登録されているユーザーデータ (実数)の再設定、表示
WorkQue_UserData関数	指定したワークキューに登録されているユーザーデータ (実数)を返す

座標変換に関する命令

コマンド	説明
AreaCorrection関数	補正エリアを用いて補正をしたポイントを返す
AreaCorrectionClr	補正エリアをクリア
AreaCorrectionDef	補正エリアの設定を返す
AreaCorrectionInv関数	補正済みのポイントをもとに戻す
AreaCorrectionOffset関数	補正済みのポイントから相対移動したポイントを返す
AreaCorrectionSet	補正エリアの設定と表示
Arm	アームの選択と、現在選択されているアーム番号の表示または設定値を返す関数
ArmSet	増設アームの設定と表示
ArmDef	アームの設定を返す
ArmClr	アームの設定をクリア

コマンド	説明
ArmCalib	アーム長補正の有効、または無効の設定と表示
ArmCalibClr	アーム長補正の設定をクリア
ArmCalibDef関数	アーム長補正の設定を返す
ArmCalibSet	アーム長補正の設定と表示
Tool	ツールの選択、または選択されているツール番号の表示、または設定値を返す関数
TLSet	ツール座標系の設定と表示
TLDef	ツール座標系の設定を返す
TLClr	ツール座標系の設定をクリア
ECP	ECPの選択、または選択されているECP番号の表示、または設定値を返す関数
ECPSet	外部制御点 (ECP) の定義と表示
ECPDef	ECP設定を返す
ECPClr	ECP設定をクリア
Base	ベース座標系の定義と表示
Local	ローカル座標系データの定義と表示
LocalDef	ローカル座標系の設定を返す
LocalClr	ローカル座標系をクリア (未定義)
Elbow	ポイントの肘姿勢の設定と設定を返す関数
Hand	ポイントのハンド(腕系)姿勢の設定と設定を返す関数
Wrist	ポイントの手首姿勢の設定と設定を返す関数
J4Flag	ポイントのJ4Flag値の設定と設定値を返す関数
J6Flag	ポイントのJ6Flag値の設定と設定値を返す関数
J1Flag	ポイントのJ1Flag値の設定と設定値を返す関数
J2Flag	ポイントのJ2Flag値の設定と設定値を返す関数
J1Angle	ポイントのJ1Angle値の設定と設定値を返す関数
J4Angle	ポイントのJ4Angle値の設定と設定値を返す関数
VxCalib	Vision Guide以外のビジョンシステムのキャリブレーションデータを作成

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

プログラム制御に関する命令

コマンド	説明
Function	ファンクションの定義
For...Next	For...Next間にある一連のステートメントを繰り返し実行
GoSub	サブルーチンの実行
Return	サブルーチンからプログラムを戻す
GoTo	プログラム制御を指定のラベルに移す
Call	ファンクションのサブルーチンとしての呼び出し
If... Then...Else...EndIf	プログラムの条件分岐
Else	If...Then...Else形式で用い、条件式が満たされなければ、Elseに続くステートメントが実行、Else は省略可能
Select ... Send	比較値にしたがって、実行するステートメントを選択
Do...Loop	ループの先頭行もしくは最終行で条件判断を行い、条件が一致または不一致の場合 Do...Loop間を繰り返す
Declare	DLL (ダイナミックリンクライブラリー)で定義されている外部ファンクションを呼び出す
Trap	割り込み、および割り込み発生時の処理の定義
OnErr	エラー処理ルーチンの位置を定義
Era	エラーが発生した関節の番号を返す関数
Erf\$	エラーが発生した関数の名前を返す
Erl	エラーが発生した行番号を返す関数

コマンド	説明
Err	エラーコードを返す関数
Ert	エラーが発生したタスクの番号を返す関数
Errb	エラーが発生したロボットの番号を返す関数
ErrMsg\$	エラーコードに対応したエラーメッセージを返す関数
UserErrorDef	指定したユーザーエラー番号またはラベルが定義されているかを返す
UserErrorLabel\$	指定したユーザーエラー番号のユーザーエラーラベルを返す
UserErrorNumber	指定したユーザーエラーラベルのユーザーエラー番号を返す
Signal	WaitSig命令を実行中のタスクに信号を送る
SyncLock	相互排他ロックを用いて複数のタスクを同期
SyncUnlock	あらかじめSyncLockでロックされた、シグナル番号のロックを解除
WaitSig	他のタスクからのSignal命令による同期シグナルを待つ
ErrorOn	コントローラーのエラー状態を返す
Error	ユーザー定義エラーを発生
EResume	エラー処理ルーチンの終了後に、プログラムの実行を再開
PauseOn	一時停止状態 (Pause状態)を返す
Exit	ループまたはファンクションを強制終了します。

プログラムの実行に関する命令

コマンド	説明
Xqt	ファンクション名で指定したタスクを実行
Pause	一時停止可能なすべてのタスクを一時停止
Cont	一時停止中のすべてのタスクを再実行
Halt	実行中のタスクを一時停止
Quit	タスクを終了
Resume	一時停止中のタスクを継続実行
MyTask	現在のプログラムのタスク番号を返す
TaskDone	タスクの終了を確認する関数
TaskState	タスクの現在の状態を取得する関数

コマンド	説明
TaskWait	指定のタスクの終了を待つ
Restart	実行中のタスクをすべて中断し、実行していた最後のプログラムグループを再実行
Recover	安全扉が開いた時点の位置へ復帰動作を実行し、状態を返す
RecoverPos	安全扉が開いた時点の位置を返す
StartMain	バックグラウンドタスクからメインファンクションを実行

擬似命令

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

ファイル管理に関する命令

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

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

フィールドバスに関する命令

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

数値に関する命令

コマンド	説明
Ctr	カウンターのカウント値を返す関数
CTReset	カウンターの定義とリセット
ElapsedTime	タクトタイム計測関数
ResetElapsedTime	タクトタイム計測用タイマーのリセット, スタート
Tmr	タイマー関数
TmReset	タイマーのリセット, スタート
Sin	指定角度のサインを返す関数
Cos	指定角度のコサインを返す関数
Tan	指定角度のタンジェントを返す関数
Acos	指定数値のアーコサインを返す関数
Asin	指定数値のアークサインを返す関数
Atan	指定数値のアークタンジェントを返す関数
Atan2	座標原点と指定点を結ぶ直角の角度を返す関数
Sqr	平方根を返す関数
Abs	絶対値を返す関数
Sgn	数値の符号を返す関数
Int	数値の小数点以下を切り捨てた値を返す関数
BClr	指定した数値の1ビットをクリアし、その値を返す
BSet	指定した数値の1ビットをセットし、その結果を返す
BTst	数値の1ビット状態を返す
BClr64	指定した数値の1ビットをクリアし、その値を返す
BSet64	指定した数値の1ビットをセットし、その結果を返す
BTst64	数値の1ビット状態を返す
Fix	実数値から整数部分を取り出す
Hex\$関数	16進数表現の数値を文字化して返す
Randomize	乱数系列の初期化

コマンド	説明
Redim	ランタイム時に配列の最大要素番号の変更
Rnd	乱数を返す
UBound	指定した配列に設定できる要素番号の最大値を返す

文字列に関する命令

コマンド	説明
Asc	文字列の先頭文字のキャラクターコードを10進数で返す
Chr\$	キャラクターコードに関する文字を返す
Left\$	文字列の左から、指定数の文字列を取り出す
Mid\$	指定した文字列から、開始位置より、指定した文字数の文字を取り出す
Right\$	文字列の末尾から、指定数の文字列を取り出す
Len	文字列の長さ (文字数) を返す
LSet\$	指定された文字列の最後にスペースを加え、指定の長さにした文字列を返す
RSet\$	文字列の先頭にスペースを追加し、指定の長さにした文字列を返す
Space\$	指定した数のスペース文字列を返す
Str\$	数値を文字列に変換
Val	数字からなる文字列を数値に変換
LCase\$	指定した文字列を小文字で返す
UCase\$	指定した文字列を大文字で返す
LTrim\$	文字列の先頭からのスペースを削除して返す
RTrim\$	文字列の最後からのスペースを削除して返す
Trim\$	文字列の先頭と最後からスペースを削除して返す
ParseStr	文字列をトークンの配列に構文解析
FmtStr	数字、あるいは、日付, 時間の表記をフォーマット
FmtStr\$	数字、あるいは、日付, 時間の表記をフォーマット
InStr	文字列からの文字列検索をし、見つかった場所を返す

コマンド	説明
Tab\$	指定した数だけのタブ文字列を返す

論理演算に関する命令

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

変数に関する命令

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

コマンド	説明
UInt32	符号なし4バイト整数型変数を宣言
UShort	符号なし2バイト整数型変数を宣言
UInt64	符号なし8バイト整数型変数を宣言

セキュリティーに関する命令

コマンド	説明
Login	実行時、他のユーザーとしてEpson RC+ にログイン

コンベヤトラッキングに関する命令

コマンド	説明
Cnv_AbortTrack	コンベヤキューポイントへの動作コマンドを中断
Cnv_Accel関数	コンベヤ追従までの加速度、減速度設定値を返す
Cnv_Accel	コンベヤ追従までの加速度、減速度設定値を設定
Cnv_AccelLim	コンベヤ追従後の加速度、減速度の制限値を設定
Cnv_AccelLim関数	コンベヤ追従後の加速度、減速度の制限値を返す
Cnv_Adjust	コンベヤに対する追従遅れ補正値を取得する動作を実施するかを設定
Cnv_AdjustClear	コンベヤに対する追従遅れ補正値を消去
Cnv_AdjustGet関数	コンベヤに対する追従遅れ補正値を返す
Cnv_AdjustSet	コンベヤに対する追従遅れ補正値を設定
Cnv_Downstream関数	コンベヤの下流限設定値を返す
Cnv_Downstream	コンベヤの下流限設定値を設定
Cnv_Fine関数	指定コンベヤの追従完了判断範囲の設定を返す
Cnv_Fine	指定コンベヤに対し、コンベヤの追従完了判断範囲の設定、表示
Cnv_Flag関数	追従中止線に対する追従状態を返す
Cnv_Mode関数	コンベヤの設定モード設定値を返す
Cnv_Mode	コンベヤの設定モード設定値を設定
Cnv_Name\$関数	指定したコンベヤの名称を返す
Cnv_Number関数	指定したコンベヤ名称のコンベヤ番号を返す
Cnv_OffsetAngle	コンベヤキューデータのオフセット値を設定

コマンド	説明
Cnv_OffsetAngle関数	コンベヤーキューデータのオフセット値を返す
Cnv_Point関数	センサー座標値をコンベヤー座標値に変換して返す
Cnv_PosErr関数	現在のトラッキング位置とターゲットとの位置偏差を返す
Cnv_PosErrOffset	現在のトラッキング位置とターゲットとの位置偏差を補正する値を設定
Cnv_Pulse関数	コンベヤーの現在位置のパルスを返す
Cnv_QueueAdd	コンベヤーキューデータにポイントデータを追加
Cnv_QueueGet関数	指定したコンベヤーキューデータからポイントデータを返す
Cnv_QueueLen関数	指定したコンベヤーキューのデータ数を返す
Cnv_QueueList	指定したコンベヤーキューデータの一覧を表示
Cnv_QueueMove	上流コンベヤーのキューデータを下流コンベヤーのキューに移動
Cnv_QueueReject	コンベヤーの2重登録防止のための最小距離を設定、表示
Cnv_QueueReject関数	コンベヤーのキューの2重登録防止距離を返す
Cnv_QueueRemove	コンベヤーキューデータからキューデータを削除
Cnv_QueueUserData	キューエントリーに関連したユーザーデータの表示、設定
Cnv_QueueUserData関数	キューエントリーに関連したユーザーデータを返す
Cnv_RobotConveyor関数	追従中のコンベヤー番号を返す
Cnv_Speed関数	コンベヤーの動作速度を返す
Cnv_Trigger	次のCnv_QueueAddステートメントのために、コンベヤーの現在位置をラッチ
Cnv_Upstream関数	コンベヤーの上流限設定値を返す
Cnv_Upstream	コンベヤーの上流限設定値を設定

DBに関する命令

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

PGに関する命令

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

衝突検出に関する命令

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

部品消耗管理に関する命令

コマンド	説明
HealthCalcPeriod	部品消耗管理の演算期間を設定する
HealthCalcPeriod関数	部品消耗管理の演算期間を返す関数
HealthCtrlAlarmOn関数	コントローラーの部品の部品消耗アラーム状態を返す関数
HealthCtrlInfo	コントローラーの部品の交換推奨時期までの残り月数を表示する
HealthCtrlInfo関数	コントローラーの部品の交換推奨時期までの残り月数を返す関数
HealthCtrlRateOffset	コントローラーの部品の部品消耗率のオフセット値を設定する
HealthCtrlReset	コントローラーの部品の部品消耗率を初期化する
HealthCtrlWarningEnable	コントローラーの部品の部品消耗アラーム通知の有効無効を設定する
HealthCtrlWarningEnable関数	コントローラーの部品の部品消耗アラーム通知の有効無効状態を返す関数
HealthRateCtrlInfo関数	コントローラーの部品の部品消耗率を返す関数
HealthRateRBInfo関数	ロボットの部品の部品消耗率を返す関数
HealthRBAAlarmOn関数	ロボットの部品の部品消耗アラーム状態を返す関数
HealthRBAAnalysis	ロボットの部品の部品消耗に関する分析結果 (交換推奨時期までの残り月数) を表示する
HealthRBAAnalysis関数	ロボットの部品の部品消耗に関する分析結果 (交換推奨時期までの残り月数) を返す関数
HealthRBDistance	指定した関節の駆動量を表示する

コマンド	説明
HealthRBDistance関数	指定した関節の駆動量を返す関数
HealthRBInfo	ロボットの部品の交換推奨時期までの残り月数を表示する
HealthRBInfo関数	ロボットの部品の交換推奨時期までの残り月数を返す関数
HealthRBRateOffset	ロボットの部品の部品消耗率のオフセット値を設定する
HealthRBReset	ロボットの部品の部品消耗率を初期化する
HealthRBSpeed	指定した関節の平均速度を表示する
HealthRBSpeed関数	指定した関節の平均速度を返す関数
HealthRBStart	ロボットの部品の部品消耗に関する分析を開始する
HealthRBStop	ロボットの部品の部品消耗に関する分析を終了する
HealthRBTRQ	指定した関節のトルク値を表示する
HealthRBTRQ関数	指定した関節のトルク値を返す関数
HealthRBWarningEnable	ロボットの部品の部品消耗アラーム通知の有効無効を設定する
HealthRBWarningEnable関数	ロボットの部品の部品消耗アラーム通知の有効無効状態を返す関数

シミュレーターに関する命令

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

ハンドに関する命令

詳細は、以下のマニュアルを参照してください。

"Hand機能"

コマンド	説明
Hand_On	グripper: ハンドの把持動作を実行 電動ドライバー: ハンドのねじ締め動作を実行
Hand_On関数	グripper: ハンドが把持状態の場合、"True"を返す関数 電動ドライバー: ハンドがねじ締め完了状態の場合、"True"を返す関数
Hand_Off	グripper: ハンドの解放動作を実行 電動ドライバー: ハンドのねじ緩め動作を実行
Hand_Off関数	グripper: ハンドが解放状態の場合、"True"を返す関数 電動ドライバー: ハンドがねじ緩め完了状態の場合、"True"を返す関数

コマンド	説明
Hand_TW関数	直前のHand_Onコマンド, Hand_Offコマンドがタイムアウトした場合、"True"を返す関数
Hand_Def関数	ハンドが定義されている場合、"True"を返す関数
Hand_Type関数	ハンドのタイプ番号を返す関数
Hand_Label\$関数	ハンドのラベルを返す関数
Hand_Number関数	ハンド番号を返す関数

安全機能に関する命令

詳細は、以下のマニュアルを参照してください。

"ロボットコントローラー 安全機能マニュアル"

コマンド	説明
SF_GetParam関数	安全機能パラメーターの情報を返す
SF_GetParam\$関数	安全機能パラメーターのテキスト情報を返す
SF_GetStatus関数	安全機能の状態ビットを返す
SF_LimitSpeedS	SLS 有効時に速度を調整する機能について速度調整値を設定、表示する
SF_LimitSpeedS関数	SLS有効時に速度を調整する機能の速度調整値を返す
SF_LimitSpeedSEnable	SLS有効時に速度を調整する機能についてOn/Offの設定、設定状態を表示する
SF_LimitSpeedSEnable関数	SLS 有効時に速度を調整する機能の状態を返す
SF_PeakSpeedS	速度監視点のピーク速度値を表示する
SF_PeakSpeedS関数	速度監視点のピーク速度値を返す
SF_PeakSpeedSClear	速度監視点のピーク速度値をクリアし、初期化する
SF_RealSpeedS	速度監視点の現在速度を表示する
SF_RealSpeedS関数	速度監視点の現在速度を返す
SF_SetIn	仮想安全I/Oの入力ポート(8ビット)を設定する
SF_SetSw	仮想安全I/Oの入力を設定する。

VRTに関する命令

詳細は、以下のマニュアルを参照してください。

"Vibration Reduction Technology"

コマンド	説明
VRT	VRT番号を選択、または選択されているVRT番号を表示

コマンド	説明
VRT関数	現在設定されているVRT番号を返す関数
VRT_Clr	VRT機能の設定をクリア
VRT_CPMotion	CP動作時にVRT機能を有効または無効にするかを指定
VRT_CPMotion関数	CP動作時にVRT機能が有効か無効かを返す関数
VRT_Def関数	選択したVRT番号の設定状態を返す関数
VRT_Description	選択したVRT番号にコメントを設定
VRT_Description\$関数	選択したVRT番号のコメントを返す関数
VRT_Label	選択したVRT番号にラベルを設定
VRT_Label\$関数	選択したVRT番号のラベルを返す関数
VRT_Number関数	VRTラベルに対応するVRT番号を返す関数
VRT_Set	VRT機能のVRTParam1, VRTParam2をVRT番号ごとに設定
VRT_Set関数	VRT番号ごとに設定されたVRTParam1, VRTParam2の設定値を返す関数
VRT_Trigger	VRソフトウェアへ測定トリガーを出力する

3. SPEL+ ランゲージリファレンス

3.1 SPEL+ ランゲージリファレンス

この章では、下記の項目からSPEL+のコマンドを説明しています。

書式

書式は、そのコマンドを使うときのフォーマットを示します。コマンドによっては、複数の書式が"解説"欄との関連を示す参照番号とともに示されています。

書式の中で使われている記号の説明は以下の通りです。

記号：[]

省略可能なパラメーターです。

この括弧が使われているパラメーターには以下のパターンがあります。

- スペースで区切られているパラメーター
スペースで区切られている場合は、パラメーターの順番を変えたり、省略したりできます。
- カンマで区切られているパラメーター
カンマで区切られている場合は、順番は固定され、変えることはできません。
また、カンマ以降のパラメーターすべてを省略することはできますが、最後にカンマで区切られているパラメーター以前のパラメーターを個別に省略することはできません。

例：

Inertia [負荷イナーシャ [, 偏心量]]

この例の場合は、以下の通りです。

OK: Inertia [負荷イナーシャ]

OK: Inertia

NG: Inertia [, 偏心量]

記号：{ }

選択可能なパラメーターです。

この括弧の中で | で区切られているパラメーターを1つ選択できます。

例：

Exit { Do | For | Function }

この場合、Do, For, Functionの中から1つ選択をしてください。

パラメーター

そのコマンドのパラメーターについての説明です。

結果

そのコマンドが返す値についての説明です。

戻り値

その関数が返す値についての説明です。

解説

コマンドの機能について、詳細を説明します。

注意

そのコマンドについて、重要事項が説明されています。

参照

そのコマンドに関連するその他のコマンドのリストです。関連するコマンドを参照したいときは、目次からページを探すことができます。

使用例

そのコマンドを使った例が示されます。

3.2 演算子

SPEL+ 言語で使用する演算子を示します。

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

演算子の優先順位

演算子は、プログラム中で以下の順位で処理されます。

優先順位	演算子	書式例	説明
1	()	(A+B)	かっこ
2	**	A**B	べき乗
3	*	A*B	乗算
	/	A/B	除算
4	Mod	A Mod B	整数の剰余

優先順位	演算子	書式例	説明
5	+	A+B	加算
	-	A-B	減算
6	=	A=B	AとBは等しい
	<>	A<>B	AとBが等しくない
	<	A<B	AがBより小さい
	>	A>B	AがBより大きい
	<=	A<=B	AがBより小さいか等しい
	>=	A>=B	AがBより大きい等しい
7	Not	Not A	否定
8	And	A And B	論理積
9	Or	A Or B	論理和
10	Xor	A Xor B	排他的論理和

3.3 !

3.3.1 !...! 並列処理

動作中にI/Oなどの入出力処理を並列して行います。

書式

動作命令 !並列処理ステートメント!

パラメーター

動作命令

次のいずれかの並列処理可能な命令を記述します。

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

並列処理ステートメント

動作中に並列処理可能なI/O ステートメントを記述します。(下表参照)

解説

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

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

Dn	次の並列処理ステートメントが実行される前に移動量の%を指定し、動作命令と同期をとるために使用します。"n"は、0から100までの整数で指定される%値で、その動作のどこから並列処理ステートメントが開始するか、開始位置を指定します。つまり、動作が全移動量のn%に達したときDn以降のステートメントが実行されます。 Jumpコマンドと組み合わせて使う場合、移動量の%値には、第3関節（上下軸）の垂直移動動作が含まれていません。Jump動作のときは第3関節（上下軸）を除いた平行移動動作に対してのみ同期がとられます。第3関節の垂直移動動作が完了したところでステートメントを実行するためには、ステートメントの最初にD0（ゼロ）を入れてください。 Jump3コマンドと組み合わせて使う場合、移動量の%値には、退避、接近動作が含まれていません。Jump3動作のときは、スパン動作に対してのみ同期がとられます。退避動作が完了したところでステートメントを実行するためには、ステートメントの最初にD0（ゼロ）を入れてください。 "Dn"は、1回の並列処理ステートメント中、最高16回まで使うことができます。
On / Off n	出力ビット番号"n"をオン、またはオフします。
MemOn / MemOff n	メモリーI/Oビット番号"n"をオン、またはオフします。
Out p,d OpBCD p,q OutW p,d	出力ポート"p"にデータ"d"を出力します。
MemOut p, d MemOutW p,d	メモリーI/Oポート"p"にデータ"d"を出力します。
Signal s	シンクロ信号を発します。
WaitSig s	信号"s"を待ってから、次のステートメントを処理します。
Wait t	"t"秒間待ち、次の並列処理ステートメントを実行します。
Wait Sw(n) = j	入力ビット"n"が"j"で定義される条件 (OnまたはOff) に一致するまで、次の並列処理ステートメントの実行を待ちます。
Wait MemSw(n) = j	メモリーI/Oビット"n"が"j"で定義される条件 (OnまたはOff) に一致するまで、次の並列処理ステートメントの実行を待ちます。
Wait その 他の条件	上記2パターン以外のWaitも可能です。詳細はWaitを参照してください。
Print	ディスプレイデバイスにデータを出力します。
Print #	指定されたコミュニケーションポートにデータを出力します。
外部ファン クション	Declareステートメントで宣言した外部ファンクションを実行します。

Hand_On n Hand_Off n	ハンド番号"n"のHand_On/Hand_Off動作を実行します。

注意

- I/Oコマンドがすべて終了する前に動作が終わった場合

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

- アームを停止するTillステートメントで、途中で動作が終了した場合

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

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

動作が終了した以降のDステートメントは実行されません。

- 100%に近い"n"値を設定した場合のパスモーションの減速

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

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

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

- Jumpステートメントと並列処理

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

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

- Hereステートメントと並列処理

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

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

上記のような使い方はできません。

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

上記のようなプログラムに変更してください。

参照

[Arc](#), [Arc3](#), [Go](#), [Jump](#), [Jump3](#), [Jump3CP](#), [Move](#), [BGo](#), [BMove](#), [TGo](#), [TMove](#)

!...! 並列処理使用例

下記は、並列処理機能を動作命令とともに使う例です。

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

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

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

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

3.4

3.4.1 #define

識別子を指定文字列に置換するために定義します。

書式

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

パラメーター

識別子

文字列パラメーターの省略のキーワードです。識別子の規則は次のとおりです。

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

パラメーター

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

置換文字列

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

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

解説

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

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

パラメーターが指定されれば、識別子はマクロとして使うことができます。

注意

- #defineを変数宣言やラベル置換に使うと、エラーになります

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

参照

[#ifdef...#else...#endif, #ifndef...#endif](#)

#define使用例

・デバッグモードの場合は次の行を非コメント化

```
' #define DEBUG
```

```
Input #1, A$
```

```
  #ifdef DEBUG
```

```
    Print "A$ = ", A$
```

```
  #endif
```

```
Print "The End"
```

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

```
Integer a
```

```
a = 25
```

```
SHOWVAL(a)
```

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

条件付コンパイルを行います。

書式

#ifdef 識別子 ...条件付コンパイルのためのソースコード

[#else]...偽の条件のためのソースコード

#endif

パラメーター

識別子

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

解説

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

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

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

参照

[#define, #ifndef...#endif](#)

#ifdef使用例

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

「デバッグモードの場合は次の行を非コメント化

」 #define DEBUG

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

3.4.3 #ifndef...#endif

条件付コンパイルを行います。

書式

#ifndef 識別子 .. 条件付コンパイルのためのソースコード

[#else]... 真の条件のためのソースコード

#endif

パラメーター

識別子

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

解説

この命令は、「もし定義されていなければ」命令とも呼ばれ、#ifdefとは反対の条件をチェックします。

#ifndef...#else...#endifは、選択したソースコードの条件付コンパイルを行います。#elseステートメントは省略可能です。

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

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

注意

■ #ifdefと#ifndefの違い

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

参照

[#define, #ifdef...#else...#endif](#)

#ifndef使用例

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

・遅延する場合は次の行をコメント化

```
#define NODELAY 1

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

3.4.4 #include

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

書式

#include "インクルードファイル名.INC "

パラメーター

インクルードファイル名

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

解説

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

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

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

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

参照

[#define, #ifdef...#else...#endif, #ifndef...#endif](#)

#include使用例

```
Include File (Defs.inc)

#define DEBUG 1
#define MAX_PART_COUNT 20

Program File (main.prg)

#include "defs.inc"

Function main
    Integer i

    Integer Parts(MAX_PART_COUNT)

Fend
```

3.4.5 #undef

#defineステートメントで定義された識別子を未定義にします。

書式

#undef 識別子名

パラメーター

識別子名

#defineステートメントで使ったキーワードを指定します。

参照

[#define](#), [#ifdef...#else...#endif](#), [#ifndef...#endif](#)

3.5 A

3.5.1 AbortMotion

動作命令を中断し動作を実行していたタスクをエラーにします。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

AbortMotion { ロボット番号 | All }

パラメーター

ロボット番号

動作を中断するロボット番号を指定します。

All

すべてのロボットの動作を中断します。

解説

AbortMotionを実行したときのロボットの状態により、次のようになります。

いずれの場合も処理を継続するには、OnErrorでエラーをフックして、エラー処理を記述する必要があります。

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

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

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

■ ロボットが動作命令を実行中の場合

ロボットはアーム動作を直ちに一時停止 (クイックポーズ) し、残りの動作を破棄します。

ロボットに対して動作命令を実行していたタスクでエラー2999 (ERROR_DOINGMOTION)が発生します。

次の動作命令は、一時停止した位置から直接次の目標位置へ動作します。

■ ロボットが一時停止 (クイックポーズ) 中の場合

AbortMotionを実行した時点では何も発生しませんが、内部的には残りの動作が破棄されます。

継続実行 (Cont)を指示した時点で、ロボットに対して動作命令を実行していたタスクで、エラー2999 (ERROR_DOINGMOTION)が発生します。

次の動作命令は、一時停止していた位置から直接次の目標位置へ動作します。

■ ロボットがWaitRecover状態 (安全扉開状態) の場合

AbortMotionを実行した時点では何も発生しませんが、内部的には残りの動作が破棄されます。

その先の動作はRecoverコマンドのフラグにより選択可能です。

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

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

■ ロボットが動作命令以外の命令を実行中の場合

直前にロボットに対して動作命令を実行していたタスクでエラー2998 (ERROR_NOMOTION)が発生します。タスクがWait やInput コマンドによる待機中の場合、即座に中断しエラー2998になります。

CP Onの設定で動作命令を実行し、動作命令からプログラムが抜けている場合は、ロボットが動作中でもエラー2998が発生します。

■ 指定したロボットが、プログラム (タスク) から動作していない場合

エラーとなります。

注意

■ 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

参照

[OnErr](#), [Recover](#), [Till](#)

AbortMotion使用例

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

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

  Go P0
  Wait Sw(1)
  Go P1

  Quit sub
  Exit Function

errstart:
  Home
  Quit sub
  Exit Function

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

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

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

3.5.2 Abs関数

絶対値を返す関数です。

書式

Abs(数値)

パラメーター

数値

式または直接数値で指定します。

戻り値

指定した数値の絶対値を返します。

解説

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

参照

[Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Mod 演算子](#), [Not 演算子](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#)

Abs関数使用例

次の例はコマンドウィンドウからPrint命令を使って実行します。

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

3.5.3 Accel

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

書式

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

パラメーター

加速設定値

最大加速度に対する割合を1以上の整数で指定します。(単位: %)

減速設定値

最大減速度に対する割合を1以上の整数で指定します。(単位: %)

退避加速設定値

Jump時における退避加速度を1以上の整数で指定します。省略可能です。Jumpコマンド時のみ設定可能です。

退避減速設定値

Jump時における退避減速度を1以上の整数で指定します。省略可能です。Jumpコマンド時のみ設定可能です。

接近加速設定値

Jump時における接近加速度を1以上の整数で指定します。省略可能です。Jumpコマンド時のみ設定可能です。

接近減速設定値

Jump時における接近減速度を1以上の整数で指定します。省略可能です。Jumpコマンド時のみ設定可能です。

結果

パラメーターが省略されたときは、現在のAccelパラメーターが返されます。

解説

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

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

AccelMax関数はAccel設定できる最大値を返します。

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

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

下記のいずれかの場合、Accel値は、初期化されます。

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

注意

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

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

■ Accel と AccelSの違い

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

■ 100を越えるAccel設定

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

参照

[AccelR](#), [AccelS](#), [Go](#), [Jump](#), [Jump3](#), [Jump3CP](#), [Power](#), [Pulse](#), [Speed](#), [TGo](#)

Accel使用例

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

[例1]

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

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

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

[例2]

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

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

>Accel
  100      100
```

100	100
100	35
>	

3.5.4 Accel関数

現在の加減速度の設定値を返す関数です。

書式

Accel(設定値番号)

パラメーター

設定値番号

整数値で下記のそれぞれの値を指定します。

- 1: 加速設定値
- 2: 減速設定値
- 3: Jump動作時の退避加速設定値
- 4: Jump動作時の退避減速設定値
- 5: Jump動作時の接近加速設定値
- 6: Jump動作時の接近減速設定値

戻り値

1以上の整数 (%)を返します。

参照

[Accel](#)

Accel関数使用例

プログラム中にAccel関数を使う例です。

```
Integer currAccel, currDecel

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


3.5.5 AccelMax関数

Accelで設定可能な加減速度の最大値を返す関数です。

書式

AccelMax(最大値番号)

パラメーター

最大値番号

整数値で下記のそれぞれの値を指定します。

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

戻り値

1以上の整数 (%)を返します。

参照

[Accel](#)

AccelMax関数使用例

プログラム中にAccelMax関数を使う例です。

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

3.5.6 AccelR

CP動作のツール姿勢変化について、加減速度の設定と表示をします。

書式

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

パラメーター

加速設定値

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

減速設定値

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

パラメーターの有効設定値

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

(単位: deg/sec²)

結果

パラメーターを省略すると、現在のAccelR設定値を表示します。

解説

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

下記のいずれかの場合、AccelR値は、初期化されます。

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

参照

[Arc](#), [Arc3](#), [BMove](#), [Jump3](#), [Jump3CP](#), [Power](#), [SpeedR](#), [TMove](#)

AccelR使用例

AccelR 360, 200

3.5.7 AccelR関数

指定したツール姿勢変化の加減速度の設定値を返します。

書式

AccelR(設定値番号)

パラメーター

設定値番号

下記のいずれかの値を、式または数値で指定します。

- 1: 加速度設定値
- 2: 減速度設定値

戻り値

加速、または減速設定値を実数値 (単位: deg/sec^2) で返します。

参照

[AccelR](#)

AccelR関数使用例

```
Real currAccelR, currDecelR
```

```
・現在の加減速度を取得
```

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

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

3.5.8 AccelS

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

書式

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

(2) AccelS

パラメーター

加速設定値

直線動作またはCP動作時の加速度を実数値で指定します (単位: mm/s^2)。減速設定値を省略すると、加速設定値が、加速時と減速時に適用されます。

減速設定値

直線動作またはCP動作時の減速度を実数値で指定します (単位: mm/s^2)。省略可能です。

退避加速設定値

Jump3時とJump3CP時における退避動作の退避加速度を実数値で指定します(単位: mm/s^2)。省略可能です。

退避減速設定値

Jump3時とJump3CP時における退避動作の退避減速度を実数値で指定します(単位: mm/s^2)。省略可能です。

接近加速設定値

Jump3時とJump3CP時における接近動作の接近加速度を実数値で指定します(単位: mm/s^2)。省略可能です。

接近減速設定値

Jump3時とJump3CP時における接近動作の接近減速度を実数値で指定します(単位: mm/s^2)。省略可能です。

パラメーターの有効設定値 (単位: mm/s^2)

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

結果

パラメーターを省略すると、加速値と減速値を表示します。

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

解説

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

下記のいずれかの場合、AccelS値は、初期化され加減速度は遅くなります。

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

注意

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

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

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

- AccelとAccelSの違い

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

- 上限値

スカラロボット (RSシリーズ含む)のAccelSの上限値は、Weight設定とスプラインユニットの位置で変化します。詳細は、以下のマニュアルを参照してください。

"マニピュレーターマニュアル - CP動作時のACCELSの設定"

垂直6軸型ロボットのAccelSの上限値は、Weightの設定で変化します。詳細は、以下のマニュアルを参照してください。

"マニピュレーターマニュアル - 仕様表"

参照

[Accel](#), [Arc](#), [Arc3](#), [Jump3](#), [Jump3CP](#), [Power](#), [Move](#), [TMove](#), [SpeedS](#)

AccelS使用例

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

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

  slow = 20          '低速度の設定
  fast = 100         '高速度の設定
  accslow = 200      '低加速度の設定
  accfast = 5000     '高加速度の設定
  AccelS accslow
  SpeedS slow
```

```
Move P1
On 1
AccelS accfast
SpeedS fast
Jump P2
.
.
.
Fend
```

3.5.9 AccelS関数

CP動作の加減速度の設定値を返す関数です。

書式

AccelS(設定値番号)

パラメーター

設定値番号

下記のいずれかの値を、整数値または式で指定します。

- 1: 加速設定値
- 2: 減速設定値
- 3: Jump3時とJump3CP時の退避加速設定値
- 4: Jump3時とJump3CP時の退避減速設定値
- 5: Jump3時とJump3CP時の接近加速設定値
- 6: Jump3時とJump3CP時の接近減速設定値
- 7: ハンド質量によって補正された加速値
- 8: ハンド質量によって補正された減速値
- 9: ハンド質量によって補正されたJump3時とJump3CP時の退避加速値
- 10: ハンド質量によって補正されたJump3時とJump3CP時の退避減速値
- 11: ハンド質量によって補正されたJump3時とJump3CP時の接近加速値
- 12: ハンド質量によって補正されたJump3時とJump3CP時の接近減速値

戻り値

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

参照

[AccelS](#), [Arc](#), [Arc3](#), [SpeedS](#), [Jump3](#), [Jump3CP](#)

AccelS関数使用例

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

3.5.10 Acos関数

指定数値のアーコサインを返します。

書式

Acos(数値)

パラメーター

数値

角度のコサインを実数値で指定します。

戻り値

指定した数値のアーコサイン値を実数値 (単位ラジアン) で返します。

解説

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

ラジアン値を、度に変換するためには、RadToDeg関数を使います。

参照

[Abs関数](#), [Asin関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [DegToRad関数](#), [RadToDeg関数](#), [Sgn関数](#), [Sin関数](#), [Tan関数](#), [Val関数](#)

Acos関数使用例

```
Function acostest
  Double x

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


3.5.11 Agl関数

指定された回転関節の角度、または直動関節の位置を返す関数です。

書式

Agl(関節番号)

パラメーター

関節番号

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

戻り値

指定された回転関節の角度、または直動関節の位置が返されます。

解説

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

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

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

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

参照

[PAgl関数](#), [PIs関数](#), [PPIs関数](#)

Agl関数使用例

下記は、コマンドウィンドウからPrint命令を使う例です。

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

3.5.12 AglToPls関数

ロボットの各関節の角度をパルスに変換します。

書式

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

パラメーター

関節位置1 - 6

関節の角度を実数値で指定します。

関節位置7

第7関節の角度を実数値で指定します。ジョイント型7軸ロボットのみを使用します。

関節位置8

付加軸S関節の角度を実数値で指定します。

関節位置9

付加軸T関節の角度を実数値で指定します。

戻り値

各関節位置から計算した関節パルスを返します。

解説

関節角度から関節パルスに変換するにはAglToPls関数を使います。

注意

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

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

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

```
Move AglToPls(0, 0, 0, 90, 0, 0)
```

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

参照

[Agl関数](#), [JA関数](#), [Pls関数](#)

AglToPls関数使用例

```
Go AglToPls(0, 0, 0, 90, 0, 0)
```

3.5.13 AIO_In関数

オプションのアナログI/O入力チャンネルからアナログ値を読み込みます。

書式

AIO_In(チャンネル番号)

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

戻り値

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

解説

In関数

参照

[AIO_InW関数](#), [AIO_Out](#), [AIO_OutW](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_In関数使用例

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

3.5.14 AIO_InW関数

オプションのアナログI/O入力チャンネルからアナログ値を読み込みます。

書式
AIO_InW(チャンネル番号)

パラメーター

チャンネル番号
アナログI/Oのチャンネル番号を指定します。

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

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

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

参照
[AIO_In関数](#), [AIO_Out](#), [AIO_OutW](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_In関数使用例

```
Long word0  
  
word0 = AIO_InW(1)
```

3.5.15 AIO_Out

オプションのアナログI/O出力チャンネルからアナログ値を出力します。

書式

AIO_Out チャンネル番号, 出力データ [, Forced]

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

出力データ

出力したい電圧[V]、または電流値[mA]を表すReal型実数を、式または数値で指定します。

Forced

省略可能です。通常は省略します。

解説

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

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

注意

■ Forcedフラグ

NoPauseタスク, NoEmgAbortタスク (Xqt実行時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)から、非常停止中や安全扉開時にアナログI/O出力を行う場合にこのフラグを指定します。

非常停止中や安全扉開時にアナログI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[AIO_In関数](#), [AIO_OutW](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#)

AIO_Out使用例

アナログI/Oチャンネル#1から7.0[V]を出力します。

```
AIO_Out 1, 7.0
```

3.5.16 AIO_Out関数

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

書式

AIO_Out(チャンネル番号)

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

戻り値

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

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

参照

[AIO_In関数](#), [AIO_Out](#), [AIO_OutW](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_Out関数使用例

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

3.5.17 AIO_OutW

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

書式

AIO_OutW チャンネル番号, 出力データ [, Forced]

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

出力データ

出力データ (0から65535の整数)を、式または数値で指定します。

Forced

省略可能です。通常は省略します。

解説

チャンネル番号で指定したアナログI/Oチャンネルに出力します。出力データは0から65535の整数を、式または数値で指定します。

出力電圧(電流)は、ボード上のスイッチで設定される出力レンジ設定により下記の通りとなります。

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

注意

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

参照

[AIO_In関数](#), [AIO_Out](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_OutW使用例

```
AIO_OutW 1, &H8000
```

3.5.18 AIO_OutW関数

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

書式

AIO_OutW(チャンネル番号)

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

戻り値

指定したアナログI/Oチャンネルの出力状態を0～65535のLong型整数で返します。

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

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

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

参照

[AIO_In関数](#), [AIO_Out](#), [AIO_OutW](#), [AIO_Out関数](#), [AIO_Set](#), [Wait](#)

AIO_OutW関数使用例

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


3.5.19 AIO_Set

オプションのアナログI/O出力チャンネルにロボットの速度情報を出力する。

書式

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

(2) AIO_Set チャンネル番号, Off

(3) AIO_Set [チャンネル番号]

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

On

速度情報をアナログ出力します。

Off

速度情報のアナログ出力を終了し、出力0に初期化します。

RefTCPSpeed

現在選択されているTCPの指令速度を出力します。

RealTCPSpeed

現在選択されているTCPの実速度を出力します。

RefECPSpeed

現在選択されているECPの指令速度を出力します。

RealECPSpeed

現在選択されているECPの実速度を出力します。

最大出力時速度

出力レンジの最大値を出力するときの速度を表すReal型実数値(単位[mm/s])を、式または数値で指定します。

最小出力時速度

出力レンジの最小値を出力するときの速度を表すReal型実数値(単位[mm/s])を、式または数値で指定します。省略可能です。省略時は"0"[0mm/s]です。

Cnv

コンベヤーとの相対TCP速度を出力します。コンベヤー番号と一緒に指定します。

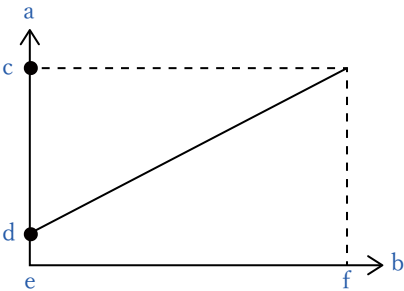
コンベヤー番号

相対TCP速度の計算に使用するコンベヤー番号を指定します。

解説

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

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



記号	説明
a	先端速度 [mm/s]
b	出力 [V or mA]
c	最大出力時速度
d	最小出力時速度
e	最小出力
f	最大出力

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

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

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

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

ECPとCnvは同時に指定できません。

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

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

参照

[AIO_In関数](#), [AIO_Out](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_Set使用例

アナログ出力チャンネル1にロボット1、ツール1のTCPの実速度出力を設定します。

ロボット動作中の速度をアナログ出力した後、速度出力設定を解除します。

```
Robot 1
Tool 1
Motor On
Power High
SpeedS 2000
AccelS 5000
```

```
AIO_Set 1, On, RealTCPSpeed, 2000.0, 0.0  
Move P1  
AIO_Set 1, Off
```

3.5.20 AIO_Set関数

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

書式

AIO_Set(チャンネル番号、インデックス)

パラメーター

チャンネル番号

アナログI/Oのチャンネル番号を指定します。

インデックス

取得する設定情報のインデックスを整数値で指定します。

戻り値

AIO_Set関数で取得可能な情報は以下の通りです。

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

参照

[AIO_In関数](#), [AIO_Out](#), [AIO_OutW](#), [AIO_Out関数](#), [AIO_OutW関数](#), [AIO_Set](#), [Wait](#)

AIO_Set関数使用例

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

3.5.21 AIO_TrackingSet

距離追従機能の設定を行います。

書式

(1) AIO_TrackingSet

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

(2) AIO_TrackingSet

チャンネル番号

パラメーター

チャンネル番号

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

計測値と距離の変換係数

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

距離0mmの測定値

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

(単位: V, mA)

アナログI/O基板の入力レンジ設定の範囲内の値を設定します。

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

追従可能範囲の下限值

追従可能範囲の下限值は、距離追従機能実行時に許容する変位量の下限值です。下限値は-300～300の実数で指定します。(単位: mm)

距離センサーの測定可能範囲の下限值より大きな値を指定してください。追従可能範囲の下限值は、追従可能範囲の上限値より小さい値を指定します。

追従可能範囲の上限値

追従可能範囲の上限値は、距離追従機能実行時に許容する変位量の上限値です。上限値は-300～300の実数で指定します。(単位: mm)

距離センサーの測定可能範囲の上限値より小さな値を指定してください。追従可能範囲の上限値は、追従可能範囲の下限值より大きい値を指定します。

追従可能範囲外での動作

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

定数	値	内容
AIOTRACK_ERRSTOP	0	追従可能範囲外でロボットがエラー停止します。

定数	値	内容
AIOTRACK_CONTINUE	1	追従可能範囲外でロボットの動作を継続します。

距離追従を行う軸

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

省略可能です。省略された場合は、"2"が設定されます。定数は以下の通りです。

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

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

結果

書式2の場合は、現在の設定値をコンソールに表示します。

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

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

表示例は以下の通りです。

例1: チャンネル番号1が設定されている場合

Ch1: ScaleFactor 1.000[V/mm or mA/mm] RefVoltage 0.000 [V or mA] ThresholdMin -10.000[mm]
ThresholdMax 10.000[mm] OutOfRangeMode AIOTRACK_ERRSTOP TrackingAxis AIOTRACK_TOOL_Z

例2: チャンネル番号1が設定されていない場合

Ch1: Undefined

解説

AIO_TrackingSetは、距離追従機能のパラメーターを設定します。設定するパラメーターは、使用する距離センサ

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

以下は、パラメーターの詳細な説明です。

計測値と距離の変換係数

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

距離0mmの測定値

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

追従可能範囲の上,下限値

アプリケーションで許容されるばらつきに応じて、上下限值を設定します。設定する値は、必ず距離センサーの測定可能範囲内の値に設定してください。距離センサーの測定可能範囲は、各センサーやユーザーの設定により異なります。距離追従機能実施前に、必ず確認してください。本パラメーターが、距離センサーの測定可能範囲外に設定された場合は、距離追従機能が正しく動作できず、ロボットが意図しない動作をする場合があります。

追従可能範囲外での動作

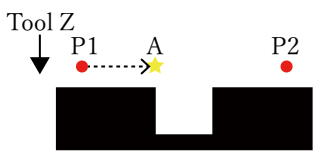
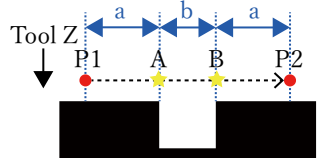
以下の図は、ToolのZ方向に対して距離追従機能を実施したときに、"追従可能範囲外での動作"パラメーターの設定を"0"にした場合と、"1"にした場合でのロボットの移動軌跡を示しています。

- P1: 距離追従開始位置
- P2: 目標位置

A点で計測範囲外になり、B点で計測範囲内に戻る対象物を図示しています。

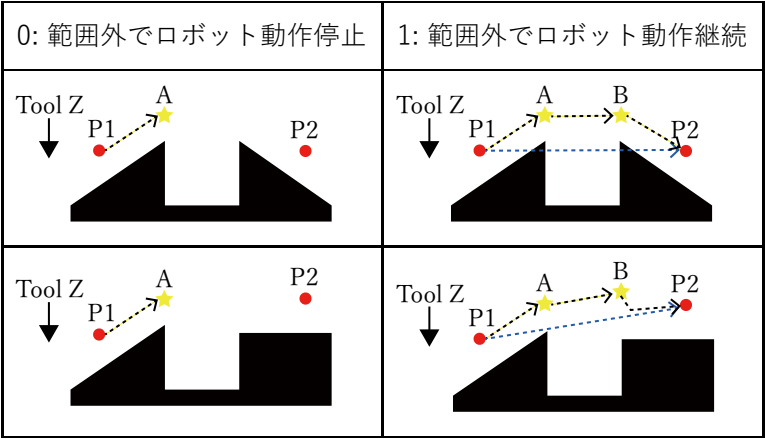
距離追従機能は、機能を開始する位置P1でのTool Z方向の測定値(変位量)を基準値として、常に測定値が基準値になるようにロボットを制御します。そのため、P1からP2へ移動したとき、P1から点Aまでは測定値が一定になるようにロボットが移動します。

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

0: 範囲外でロボット動作停止	1: 範囲外でロボット動作継続
	
	■ a: 距離追従機能 有効 ■ b: 範囲外、距離追従機能 無効

"1"を設定した場合の追従可能範囲外でのロボット動作は、開始位置(P1)から、目標位置(P2)に向かってCP動作する軌道で移動します。以下の図では、追従可能範囲外である点A-B間の軌道が、P1-P2間の軌道と平

行になります。点Bに到達後は、追従可能範囲内となるため、計測値を基準値にするロボットの制御が働き、急にロボットが移動する場合があります。



 **注意**

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

使用する機器や実施環境に応じて適切に設定してください。

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

参照

[AIO_TrackingStart](#), [AIO_TrackingEnd](#), [AIO_TrackingOn関数](#)

AIO_TrackingSet使用例

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

 **注意**

使用例で設定しているパラメーターは参考値です。

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

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

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

  Go P1      ' 開始位置P1に移動
  AIO_TrackingSet 1,1,0,-5,5,0,2    ' 距離追従機能の設定
  AIO_TrackingStart 1,5,5,5        ' 距離追従機能の開始
  Move P2    ' P2まで距離追従機能実行しながら移動
  AIO_TrackingEnd    ' 距離追従機能の終了
  Motor Off
Fend
```


3.5.22 AIO_TrackingStart

距離追従機能を開始します。

書式

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

パラメーター

チャンネル番号

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

ProportionalGain

距離追従機能の比例ゲインを、0をのぞく50以下の正の実数で指定します。ロボット移動速度やワークの形状などにより最適値が変わるため、使用環境に応じて設定が必要です。

IntegralGain

距離追従機能の積分ゲインを、100以下の正の実数で指定します。省略可能です。省略された場合は、"0"が設定されます。距離追従精度を上げる場合は、積分ゲインを調整してください。

DifferentialGain

距離追従機能の微分ゲインを、100以下の正の実数で指定します。省略可能です。省略された場合は、"0"が設定されます。距離追従精度を上げる場合は、微分ゲインを調整してください。

解説

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

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

距離追従機能は、AIO_TrackingStart実行により開始され、AIO_TrackingEnd実行で終了します。

AIO_TrackingEndを実行するまでは、距離追従機能が動作し続けます。距離追従機能を使用しない場合は、直ちにAIO_TrackingEndを実行し、距離追従機能を終了してください。

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

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

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

距離追従機能実行中に特異点近傍を通過した場合は、エラーになります。

以下のコマンドは距離追従機能実行中に実行できません。

MOTOR OFFになるコマンド	Motor off, SFree
PTP動作コマンド	BGo, Go, JTran, Jump, Jump3, Jump3CP, JumpTLZ, Pass, Ptran, Pulse, TGo
力覚制御実行コマンド	FCKeep, FC付き動作命令, FS#.Reset, FS.Reboot
トルク制御実行コマンド	TC
コンベヤトラッキング実行コマンド	動作命令 + Cnv_QueGet

VRT実行コマンド	VRT, VRT_CPMotion
設定変更コマンド	AIO_TrackingSet, Arm, ArmSet, Base, Calib, CalPls, ECP, ECPSet, Hofs, Inertia, MCal, Power, TLSet, Tool, Weight (AIO_TrackingSet, ArmSet, ECPSet, TLSetは、使用中の番号に関する変更のみエラーになります。)
その他	Brake, Here, Home, VCal, WaitPos

ProportionalGain, IntegralGain, DifferentialGainの設定

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

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

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

各ゲインの設定方法は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - 距離追従機能"

 注意

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

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

異常な動作をした場合には、すぐに非常停止ボタンを押してください。

参照

[AIO_TrackingSet](#), [AIO_TrackingEnd](#), [AIO_TrackingOn](#)関数

AIO_TrackingStart使用例

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

 注意

使用例で設定しているパラメーターは参考値です。

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

また、異常な動作をした場合には、すぐに非常停止ボタンを押してください。

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

  Go P1      ' 開始位置P1に移動
  AIO_TrackingSet 1,1,0,-5,5,0,2      ' 距離追従機能の設定
  AIO_TrackingStart 1,1,0,0      ' 距離追従機能の開始
```

```
Move P2      ' P2まで距離追従機能実行しながら移動
Move P3      ' P3まで距離追従機能実行しながら移動
AIO_TrackingEnd    ' 距離追従機能の終了
Motor Off
Fend
```

3.5.23 AIO_TrackingEnd

距離追従機能を終了します。

書式

AIO_TrackingEnd

解説

AIO_TrackingStartで開始した距離追従機能を終了します。

参照

[AIO_TrackingSet](#), [AIO_TrackingStart](#), [AIO_TrackingOn](#)関数

AIO_TrackingEnd使用例

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

注意

使用例で設定しているパラメーターは参考値です。

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

また、異常な動作をした場合には、すぐに非常停止ボタンを押してください。

```
Function Main
  Integer ChNo
  Motor On
  Power High
  SpeedS 30
  AccelS 300,300
  ChNo=1

  Go P1      ' 開始位置P1に移動
  AIO_TrackingSet ChNo,10,0,-3,3,0,2      ' 距離追従機能の設定
  AIO_TrackingStart ChNo,1,0,0      ' 距離追従機能の開始
  Move P2      ' P2まで距離追従機能実行しながら移動
  Move P3      ' P3まで距離追従機能実行しながら移動
  AIO_TrackingEnd      ' 距離追従機能の終了
  Motor Off
Fend
```

3.5.24 AIO_TrackingOn関数

指定したロボットが距離追従機能を実行しているか否かを返します。

書式

AIO_TrackingOn(ロボット番号)

パラメーター

ロボット番号

状態を取得したいロボットの番号を式、または数値で指定します。

戻り値

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

参照

[AIO_TrackingSet](#), [AIO_TrackingStart](#), [AIO_TrackingEnd](#)

AIO_TrackingOn使用例

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

コマンドウィンドウでの使用例

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

3.5.25 Align関数

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

書式

Align (ポイント指定[, ローカル座標系番号[, 座標軸指定]])

パラメーター

ポイント指定

対象とするポイントデータを指定します。

ローカル座標系番号

姿勢を整列するための基準にしたいローカル座標系番号を指定します。省略するとベース座標系が指定されます。

座標軸指定

整列したい座標軸を指定します。省略すると最も近い座標軸に整列します。

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

解説

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

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

参照

[AlignECP関数](#), [LJM関数](#)

Align関数使用例

```
Move Align(P0) ROT

P1 = Align(P0, 1)
Move P1 ROT

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

3.5.26 AlignECP関数

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

書式

AlignECP (ポイント指定, ECP座標系番号)

パラメーター

ポイント指定

対象とするポイントデータを指定します。

ECP座標系番号

姿勢を 整列するための基準にしたいECP座標系番号を指定します。

解説

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

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

参照

[Align関数](#), [LJM関数](#)

AlignECP関数使用例

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

3.5.27 And 演算子

2つの値のAnd演算 (論理またはビット)を行います。

書式

result =値1 And 値2

パラメーター

値1, 値2

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

result

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

解説

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

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

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

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

参照

[LShift関数](#), [Mask 演算子](#), [Not 演算子](#), [Or 演算子](#), [RShift関数](#), [Xor 演算子](#)

And演算子使用例

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

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

Function BitWiseAnd()

    If (Stat(0) And &H800000) = &H800000 Then
        Print "The enable switch is ON"
    EndIf
```


Fend

```
>print 15 and 7  
7  
>
```

3.5.28 AOpen

ファイルをアペンドモード (追加書き込み)で開きます。

書式

AOpen ファイル名As #ファイル番号

.
.

Close #ファイル番号

パラメーター

ファイル名

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

ファイル番号

30～63の整数値、または式で指定します。

解説

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

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

Closeステートメントでファイルを閉じ、ファイル番号を解放します。

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

注意

- ネットワークパスが使用できます。
- ファイルへの書き込みはバッファリングされます。
- バッファリングされたデータはFlushステートメントで書き込むことができます。Closeステートメントでファイルを閉じた場合も書き込まれます。

参照

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

AOpen使用例

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

```
    Print #FileNum, i  
Next i  
Close #FileNum
```

3.5.29 Arc, Arc3

Arcはベース座標系のXY平面で、アームを現在位置から指定位置まで円弧補間動作で動かします。

Arc3は3次元で、アームを現在位置から指定位置まで円弧補間動作で動かします。

この2つのコマンドは、ロボットが、水平多関節型 (RSシリーズを含む) でも垂直6軸型 (Nシリーズを含む) でも使うことができます。

書式

Arc

- (1) Arc 経由座標, 目標座標 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]
- (2) Arc 目標座標, 半径, 長短, 回転方向 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]
- (3) Arc 目標座標, 回転角 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]
- (4) Arc 目標座標, 中心座標, 長短 [ROT] [CP] [Till | Find] [!並列処理!] [SYNC]

Arc3

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

パラメーター

経由座標

ポイントデータ、もしくはXY関数で指定します。アームが現在位置から目標座標に移動するまでの軌道で通過しなくてはならないポイントです。

目標座標

ポイントデータ、もしくはXY関数で指定します。アームが円弧動作で移動する到達地点、目標位置です。

中心座標

ポイントデータ、もしくはXY関数で指定します。アームが円弧動作する際の中心座標です。Arcコマンドの場合、Z座標と姿勢 (U,V,W)は利用されません。Arc3コマンドの場合、現在地, 目標座標, 中心座標のX, Y, Z座標を通る平面上を円弧動作します。その際、中心座標の姿勢 (U, V, W)は利用されません。

半径

円弧動作の半径を実数または式 (単位 :mm)で指定します。

長短

円弧が現在位置から目標座標に対して短い経路を通るか、長い経路を通るかを指定します。短い経路を通る場合はARC_SHORT、長い経路を通る場合はARC_LONGを指定します。

回転方向

円弧動作の回転方向を指定します。ベース座標系のXY平面に対しZ軸方向の反時計回りの場合はARC_PLUS、時計回りの場合はARC_MINUSを指定します。

回転角

円弧動作の回転する角度を実数または式 (単位 :度)で指定します。
360度より小さく、-360度より大きい値を指定します。ベース座標系のXY平面に対しZ軸方向の反時計回りの場合はプラス、時計回りの場合はマイナスで指定します。

ROT

ツール姿勢変化を優先させて、動作の速度、加減速度を決めます。省略可能です。

ECP

外部制御点動作を指定します。省略可能です。(ECPオプション使用時のみ有効)

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

！並列処理！

Arcステートメントには並列処理ステートメントが使えます。省略可能です。(詳細については"並列処理"を参照してください。)

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。省略可能です

解説

ArcとArc3は、アームを現在位置から目標座標まで、経由座標を通過して円弧補間動作で動かすときに使われます。与えられた3点(現在地, 経由座標, 目標座標)または、与えられた2点(現在地, 目標座標)と半径, 回転角, 中心座標などのパラメーターから自動的に円弧補間軌道を演算し、目標座標に至るまで、その軌道に沿ってアームを動かします。

また、水平多関節型ロボットの場合は、現在地から目標座標の点へU座標が動きますが、垂直6軸型ロボットの場合は、現在地から目標座標に向かう最短の回転姿勢でU, V, W座標が動きます。経由座標で指定した姿勢(U, V, W)は通過しません。

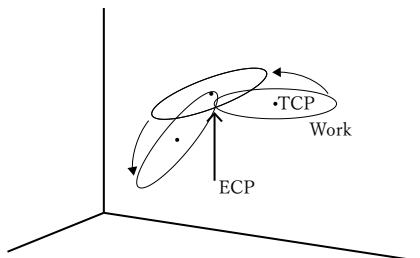
使用する場合は、実際の動きを事前に確認してください。

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

ロボットのツール先端位置を、特定の座標に固定したままツール姿勢のみを変化させようとした場合や、ツール先端の移動距離に対してツール姿勢変化が大きい場合、ツール姿勢変化速度が著しく速くなる恐れがあります。これを防ぐため、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働きます。

CP動作時のツール姿勢変化速度の上限値を手動で設定したい場合は、SpeedRLimitationをオンにしてください。SpeedRLimitationをオンにした場合、CP動作時のツール姿勢変化速度が設定したSpeedRを超える場合は、ツール姿勢変化速度がSpeedRになるように動作速度を制限します。ツール姿勢変化速度が設定したSpeedRを超えない場合は、設定したSpeedSで動作します。ツール姿勢変化速度の上限値はSpeedRであらかじめ設定してください。

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



Arc動作の速度と加速度の設定

ArcとArc3命令に対する速度と加減速度の設定は、それぞれSpeedSとAccelSにより行います。SpeedSで速度(単位: mm/s)、AccelSで加減速度(単位: mm/s²)を指定します。

注意

- Arc命令は水平面においてのみ有効です

Arc命令によって描かれる軌道はXY平面上での真円弧になります。経由座標で指定したZ座標や姿勢 (U, V, W) は通過せず、現在のポイントと目標座標値を補間します。

Arc3では3次元空間における円弧軌道を指定できます。経由座標で指定したZ座標は通過しますが、姿勢 (U, V, W) は通過せず、現在のポイントと目標座標値を補間します。

- Arc命令に対する範囲確認

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

- Arc動作の設定にあたって

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

- Arc, Arc3をCPとともに用いる

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

- コンベヤートラッキングでArc, Arc3コマンドを利用する場合

経由座標を指定しないArc, Arc3書式は、Cnv_QueueGetで得たポイントデータを入れて利用することはできません。経由座標指定のArc, Arc3書式を使用してください。

- 円弧の角度が±180度、±360度付近の場合

経由座標を指定しないArc, Arc3書式の場合、円弧の角度が±180度付近や±360度付近では想定通りの軌跡にならない場合があります。またArc3で円弧の角度が±180度となるように中心座標、目標座標を指定した場合、円弧が特定できないためエラーになります。円弧の角度が±180度、±360度付近では、経由座標指定のArc, Arc3書式を使用してください。

- 半径指定Arcコマンドの制限

半径は、現在地から目標座標まで距離の1/2以上の値を設定してください。指定した半径が短すぎると円弧が描けないためエラーになります。

- 回転角指定Arcコマンドの制限

回転角は、360度より小さく、-360度より大きい値を指定してください。0度は円弧にならないためエラーになります。

起こりやすいエラー

- ハンド(腕系)の属性を変更する

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

- 可動範囲外にアームを動かそうとするととき

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

- ほぼ直線で動かそうとするととき

もし指定された円弧動作が、ほぼ直線になる場合、エラーになります。Moveコマンドを利用してください。

参照

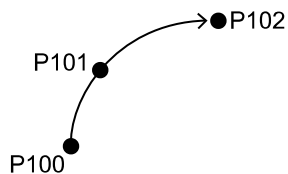
[!...! 並列処理](#), [AccelS](#), [Move](#), [SpeedS](#)

Arc, Arc3使用例（経由座標指定）

下記は、図のような軌道を描くプログラムの例です。

P100から動作を開始し、P101を経由して、P102に到着する円弧動作軌道をとります。

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



ヒント

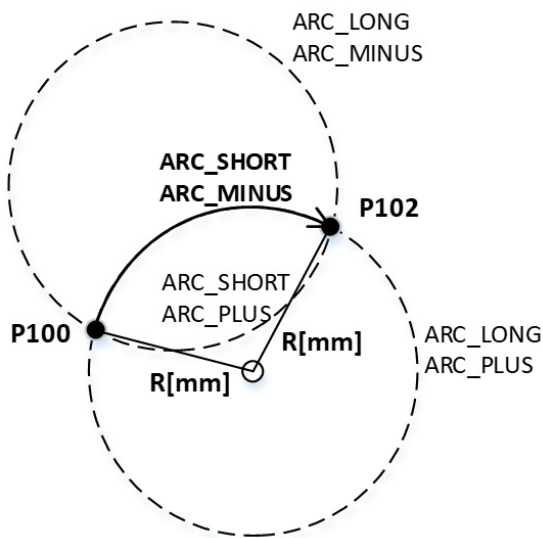
初めてArc命令を使うときには、可動範囲の中にあるロボットの側のポイントを使って、簡単な円弧で試してみることをお奨めします。そのとき、実際に描かれる円弧軌道を想定してみてください。そしてアームが通常の可動範囲外を移動してしまうポイントをティーチングしないようにしてください。

Arc使用例（半径指定）

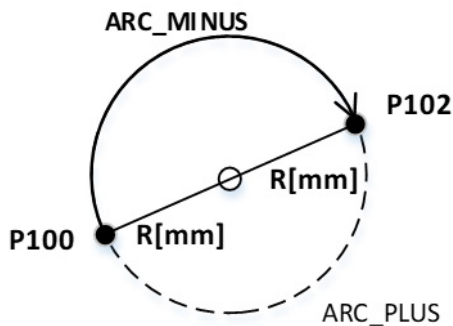
下記は、図のような軌道を描くプログラムの例です。

P100から動作を開始し、半径 R[mm]（下記の例では500.0mm）でP102に到着する円弧動作軌道をとります。

```
Function ArcTest
  Go P100
  Arc P102, 500.0, ARC_SHORT, ARC_MINUS
Fend
```



半径R[mm]が、現在地P100から目標座標P102まで距離の1/2に等しい場合、下記のように動作します。半径R[mm]がさらに短い場合、円弧が描けないためエラーになります。

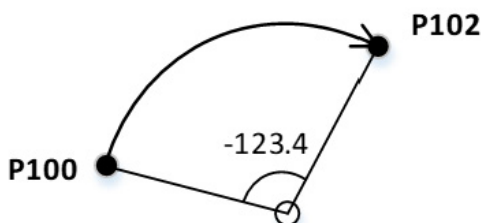


Arc使用例（回転角指定）

下記は、図のような軌道を描くプログラムの例です。

P100から動作を開始し、指定した回転角（単位：度）だけ回転してP102に到着する円弧動作軌道をとります。回転角が負の値の場合、下記の例のように、ベース座標系のXY平面に対しZ軸方向の時計回りに回転します。

```
Function ArcTest
  Go P100
  Arc P102, -123.4
Fend
```

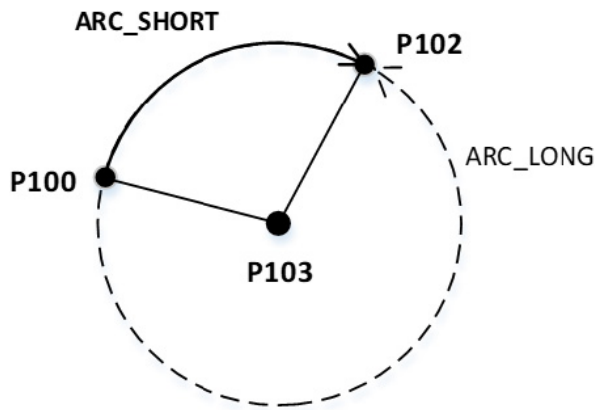


Arc, Arc3使用例（中心座標指定）

下記は、図のような軌道を描くプログラムの例です。

P103を中心座標として、P100から動作を開始し、P102に到着する円弧動作軌道をとります。

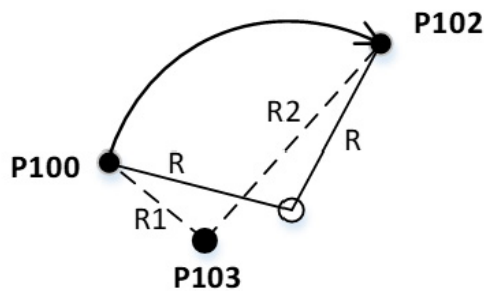
```
Function ArcTest
  Go P100
  Arc P102, P103, ARC_SHORT
Fend
```

ヒント

■ 中心座標がずれていた場合

中心座標P103と現在地P100との距離R1と、中心座標P103と目標座標P102との距離R2が異なる位置に中心座標P103を指定した場合、P103ではなく現在地と目標座標の距離が等しくなる位置を中心座標として円弧動作を行います。この場合、想定通りの軌跡にならない場合があるので、現在地と目標座標から等距離に中心座標を指定してください。



■ 中心座標指定のArcコマンドで±180度付近を指定した場合

Arcコマンドにおいて、円弧の角度が±180度となるように中心座標P103を指定した場合、ARC_LONGを指定するとベース座標系のXY平面に対しZ軸方向の時計回りに回転します。ARC_SHORTを指定するとベース座標系のXY平面に対しZ軸方向の反時計回りに回転します。

ただし、円弧の角度が±180度からわずかにずれると、ARC_LONGまたは、ARC_SHORTで指定した長さの経路で回転します。±180度付近で想定通りの円弧を描くためには経由座標指定を利用してください。

3.5.30 Arch

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

書式

(1) Arch アーチ番号, 退避距離, 接近距離

(2) Arch アーチ番号

(3) Arch

パラメーター

アーチ番号

アーチ番号を0～6の整数で指定します。有効値は0～6の整数で、次ページのArchテーブルのように有効な値は全部で7つあります。

退避距離

Jump命令で水平動作前の退避距離（出発点からの垂直距離）を指定します。（単位: mm）Jump3, Jump3CP命令で、スパン動作前の退避距離を指定します。（単位: mm）

接近距離

Jump命令で水平移動が完全に終了した段階での、接近距離（目的点からの垂直距離）を指定します。（単位: mm）

Jump3, Jump3CP命令で、スパン動作が完全に終了した段階での、接近距離を指定します。（単位: mm）

結果

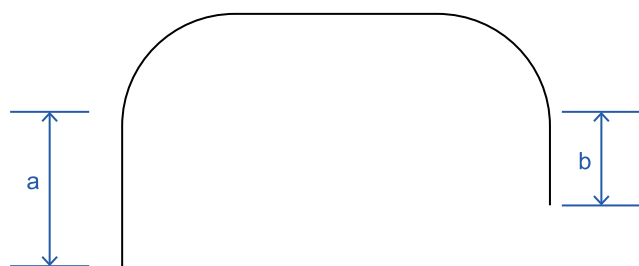
すべてのパラメーターを省略すると、Archテーブルの内容すべてが表示されます。

アーチ番号だけを指定すると、指定アーチ番号のArchテーブル内容が表示されます。

解説

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

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

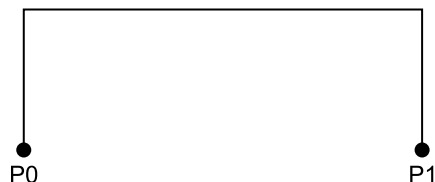


記号	説明
a	退避距離
b	接近距離

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

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

ゲートモーション (Arch 7 のJump動作)



Arch テーブル デフォルト値

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

注意

■ ゲートモーションになるその他の場合

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

■ Arch 値は保存されます

Archテーブルの値は、ユーザーが変更しない限り、保持されます。

アーチ使用時の重要事項

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

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

え、垂直下降量が減る傾向が強くなります。期待した垂直下降距離が出ない場合は、速度や減速度を下げるか、下降距離を長めに設定してください。

参照

[ArchClr](#), [Jump](#), [Jump3](#), [Jump3CP](#)

Arch使用例

下記は、コマンドウィンドウから行うArch値の設定例です。

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

3.5.31 ArchClr

アーチパラメーターの設定を初期値に戻します。

書式

```
ArchClr アーチ番号
```

パラメーター

アーチ番号

初期値に戻すアーチの番号を整数 または 式で指定します。

参照

[Arch](#)

ArchClr 使用例

```
ArchClr 1
```

3.5.32 Arch関数

アーチの設定を返します。

書式

Arch(アーチ番号, パラメーター番号)

パラメーター

アーチ番号

0～6の整数を指定します。

パラメーター番号

- 1: 退避距離
- 2: 接近距離

戻り値

パラメーター番号で指定した距離を返します。

参照

[Arch](#), [ArchClr](#)

Arch関数使用例

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

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

3.5.33 AreaCorrection関数

補正エリアを用いて補正をしたポイントを返す関数

書式

AreaCorrection(ポイント指定, エリア番号)

パラメーター

ポイント指定

補正の対象とするポイントデータを指定します。

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

解説

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

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

設定されていない補正エリアを指定した場合は、エラーになります。

注意

- ティーチング済みのポイント

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

- 補正エリアから遠い場合

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

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

- 補正エリアを設定した姿勢フラグと異なる場合

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

- 補正エリアを設定した姿勢(U, V, W)と異なる場合

スカルロボット(RSシリーズを含む)の場合、補正できます。

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

参照

[AreaCorrectionSet](#), [AreaCorrectionClr](#), [AreaCorrectionDef](#)関数, [AreaCorrectionInv](#)関数,
[AreaCorrectionOffset](#)関数, [DiffToolOrientation](#)関数

AreaCorrection使用例

Function sample

```
' P(1:4) 基準点
P1 = XY(-100, 200, -20, 0)
P2 = XY(100, 200, -20, 0)
P3 = XY(-100, 400, -20, 0)
P4 = XY(100, 400, -20, 0)
' P(11:14) 実際はP(1:4)をティーチングしたポイントを用いる
P11 = XY(-100, 200.5, -20, 0)
P12 = XY(100.3, 200.1, -20, 0)
P13 = XY(-100.4, 400.8, -20, 0)
P14 = XY(100.2, 400.4, -20, 0)
' 補正エリアを設定
AreaCorrectionSet 1, P(1:4), P(11:14), MODE_PLANE
P999 = AreaCorrection(P1, 1) ' P999は補正済みのポイント
Print Dist(P11, P999)
P999 = AreaCorrection(XY(0, 300, -20, 0), 1) ' エリア内の点を補正
Print P999
```

Fend

[出力結果]

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


3.5.34 AreaCorrectionClr

補正エリアをクリアします。

書式

AreaCorrectionClr エリア番号

パラメーター

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

結果

エリア番号に対応する補正エリアをクリアします。

ロボット動作中は本命令を実行できません。停止した状態で使用してください。

参照

[AreaCorrectionSet](#), [AreaCorrectionDef関数](#), [AreaCorrectionInv関数](#), [AreaCorrectionOffset関数](#)

AreaCorrectionClr使用例

```
AreaCorrectionClr 1
```

3.5.35 AreaCorrectionDef関数

指定した補正エリアが設定されているかを返します。

書式

AreaCorrectionDef(エリア番号)

パラメーター

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

戻り値

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

参照

[AreaCorrectionSet](#), [AreaCorrectionClr](#), [AreaCorrectionInv関数](#), [AreaCorrectionOffset関数](#)

AreaCorrectionDef関数使用例

```
Function DisplayAreaCorrectionDef(areaNum As Integer)

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

3.5.36 AreaCorrectionInv関数

補正済みのポイントをもとに戻す関数

書式

AreaCorrectionInv(ポイント指定, エリア番号)

パラメーター

ポイント指定

補正の対象とするポイントデータを指定します。

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

解説

AreaCorrection関数で補正済みのポイントに対して、補正前のポイントデータを返します。

実際にティーチングをして作成したポイント、または補正済みのポイントに対してAreaCorrectionInvを適用すると、補正前のポイントデータが得られます。

設定されていない補正エリアを指定した場合は、エラーになります。

参照

[AreaCorrectionSet](#), [AreaCorrectionClr](#), [AreaCorrectionDef関数](#), [AreaCorrectionOffset関数](#)

AreaCorrectionInv使用例

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

[出力結果]

```
0
0
```

3.5.37 AreaCorrectionOffset関数

補正済みのポイントから相対移動したポイントを返す関数

書式

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

パラメーター

ポイント 指定

相対移動の基準となる位置のポイントデータを指定します。

相対移動量 指定

相対移動量をポイントデータで指定します。

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

相対関係選択

どの座標系を基準として相対移動するかを表します。省略した場合は、ローカル座標系基準になります。

ローカル座標系基準で動かす場合は、ポイント 指定が定義されている座標系を基準として相対移動した座標になります。ツール座標系基準で動かす場合は、ポイント 指定位置を基準として相対移動した座標になります。

相対関係選択	定数	値
ローカル座標基準	AC_LOCAL	0
ツール座標基準	AC_TOOL	1

解説

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

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

設定されていない補正エリアを指定した場合はエラーになります。

注意

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

参照

[AreaCorrectionSet](#), [AreaCorrectionClr](#), [AreaCorrectionDef](#)関数, [AreaCorrectionInv](#)関数, [Here](#)

AreaCorrectionOffset使用例

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

3.5.38 AreaCorrectionSet

補正エリアの設定と表示を行います。

書式

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

(2) AreaCorrectionSet エリア番号

(3) AreaCorrectionSet

パラメーター

エリア番号

エリア番号(1～8の整数)を式または数値で指定します。

基準連続ポイント

基準点としたいポイントデータの連番を、始点と終点の2つのポイント番号をコロンで結合してP(1:4)のように指定します。

補正の効果を最大にするために、基準点は補正したいポイントを取り囲むように選択してください。

教示連続ポイント

基準連続ポイントに対応する教示したポイントデータを、始点と終点の2つのポイント番号をコロンで結合してP(1:4)のように指定します。ポイントデータの並び順は、基準連続ポイントに対応するように設定してください。

補正の種類

補正の種類を表す整数値です。

平面補正では、基準点として選んだポイントによって構成される平面上のポイントに対して補正を行うことができます。平面補正を選択する場合は、基準連続ポイントを平面上に配置してください。基準点数は、最低3点必要になります。

空間補正では、基準点として選んだポイントによって構成される立体空間上のポイントに対して補正を行うことができます。立体補正を選択する場合は、基準連続ポイントが補正したい領域を取り囲むように選択してください。基準点数は、最低4点必要になります。

補正の種類	定数	値
平面	MODE_PLANE	2
空間	MODE_SPACE	3

結果

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

解説

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

動作中は本命令を実行できません。停止した状態で使用してください。

基準位置の選定方法は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - エリア歪み補正機能"

注意

■ 補正エリアデータについて

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

■ ツールについて

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

■ 補正エリアから遠い場合

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

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

■ 補正エリアを設定した姿勢フラグと異なる場合

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

■ 補正エリアを設定した姿勢(U,V,W)と異なる場合

スカラロボット(RSシリーズを含む)の場合、補正できます。

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

参照

[AreaCorrectionClr](#), [AreaCorrectionDef](#)関数, [AreaCorrectionInv](#)関数, [AreaCorrectionOffset](#)関数, [DiffToolOrientation](#)関数

AreaCorrectionSet使用例

使用例を示します。P11～P14には、ティーチングしたポイントを用いてください。

以下のようにP1～P4を基準点の図面上での位置とした場合、補正エリアはP1, P2, P3, P4を頂点とする幅200mmの正方形になります。

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

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

3.5.39 Arm

アームの選択と、現在選択されているアーム番号の表示をします。

書式

- (1) Arm アーム番号
- (2) Arm

パラメーター

アーム番号

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

結果

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

解説

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

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

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

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

注意

- Arm 0

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

- アーム長補正オプション使用時

ArmCalibをOnにすると、Arm 0にアーム長補正值が適用され、自動的にArm 0に切り替わります。アーム長補正值を適用した動作を行う場合は、Arm 0で使用してください。ArmCalibをOnにしても、Arm 0以外では、アーム長補正值を適用した動作になりません。

ロボットアームの標準パラメーターを使用する場合は、ArmCalibをOffにして、Arm 0を使用してください。

- 設定されていないアーム番号

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

参照

[ArmClr](#), [ArmSet](#), [ECPSet](#), [TLSet](#), [ArmCalibSet](#)

Arm使用例

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

コマンドウィンドウでの操作例

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

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

3.5.40 Arm関数

現在選択されているロボットのアーム番号を返します。

書式

Arm

戻り値

現在のアーム番号を返します。

参照

[Arm](#)

Arm関数使用例

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

3.5.41 ArmCalib

現在選択されているロボットのアーム長補正を有効、または無効にします。

書式

ArmCalib On | Off

パラメーター

On | Off

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

解説

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

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

アーム長補正オプションを購入した場合は、出荷時にアーム長補正が有効に設定されています。

アーム長補正有効状態でArmを "0" 以外に設定した場合は、設定したArm番号の設定で動作します。

注意

- このコマンドは、アーム長補正オプションがインストールされている場合のみ使用できます。
- ArmCalib On状態のバックアップファイルをアーム長補正オプション無効のコントローラーへリストアしないでください。

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

参照

[ArmCalibClr](#), [ArmCalibSet](#), [ArmCalibDef](#)関数

ArmCalib使用例

次の例はコマンドウィンドウから実行します。

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

3.5.42 ArmCalib関数

現在選択されているロボットのアーム長補正の状態を返します。

書式

ArmCalib

戻り値

- 0 = アーム長補正無効
- 1 = アーム長補正有効

参照

[ArmCalib](#)

ArmCalib関数使用例

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

3.5.43 ArmCalibClr

アーム長補正の設定をクリアします。

書式

ArmCalibClr

注意

- ArmCalibClrは、必要な時以外、絶対に使用しないでください。

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

参照

[ArmCalib](#), [ArmCalibSet](#)

ArmCalibClr使用例

```
ArmCalibClr
```

3.5.44 ArmCalibDef関数

アーム長補正の設定状態を返します。

書式

ArmCalibDef

戻り値

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

参照

[ArmCalib](#), [ArmCalibClr](#), [ArmCalibSet](#)

ArmCalibDef関数使用例

```
Function DisplayArmCalibDef
  Integer i

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

3.5.45 ArmCalibSet

アーム長と関節オフセットの設定と表示をします。

書式

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

パラメーター

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

結果

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

解説

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

注意

- ArmCalibSetは、必要な時以外、絶対に変更しないでください

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

参照

[ArmCalib](#), [ArmCalibClr](#), [ArmCalibDef](#)関数

ArmCalibSet使用例

次の例はコマンドウィンドウから実行します。

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

3.5.46 ArmCalibSet関数

アーム長補正の設定値を1つ返します。

書式

ArmCalibSet(設定値番号)

パラメーター

設定値番号

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

水平多関節型ロボット

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

戻り値

上記表のいずれかを指定したパラメーターの設定値を、実数値で返します。

解説

アーム長補正が設定されていない場合、エラーが発生します。

参照

[ArmCalibClr](#), [ArmCalibSet](#)

ArmCalibSet関数使用例

```
Double L1, L2, Angle

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


3.5.47 ArmClr

アームの設定をクリアします。

書式

ArmClr アーム番号

パラメーター

アーム番号

15個のアームのうち、設定をクリアするアームの番号を整数と式で指定します。

(アーム0は、デフォルトアームなので、クリアすることはできません。)

参照

[Arm](#), [ArmSet](#), [ECPSet](#), [Tool](#), [Local](#), [LocalClr](#), [TLSet](#)

ArmClr使用例

```
ArmClr 1
```

3.5.48 ArmDef関数

アームの設定状態を返します。

書式

ArmDef (アーム番号)

パラメーター

アーム番号

状態を返すアームの番号を整数値で指定します。

戻り値

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

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLClr](#), [TLSet](#)

ArmDef使用例

```
Function DisplayArmDef (armNum As Integer)

    Integer i

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

3.5.49 ArmSet

増設アームの設定と表示をします。

書式

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

パラメーター

アーム番号

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

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

結果

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

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

解説

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

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

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

- 同一のデータポイントを複数のアームで使用
- Palletを使う
- CP動作を使う
- 相対位置指定を行う
- ローカル座標系を使用

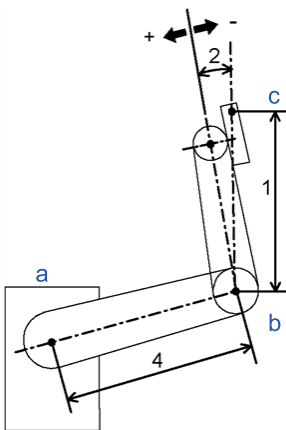
回転関節を持つ水平多関節型 (RSシリーズを含む) では、関節角度の演算はArmSetのパラメーターに基づいて行われるので、増設アームやハンドの設定が必要な場合には、この設定が重要です。

注意

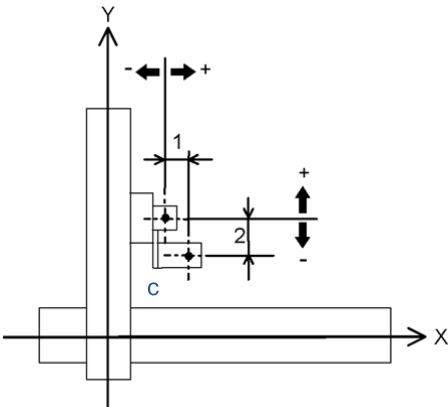
■ Arm 0

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

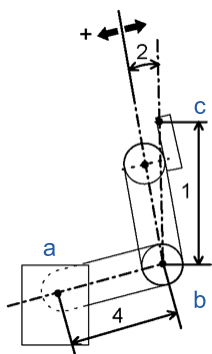
■ 水平多関節型ロボット



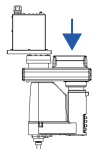
■ 直角座標型ロボット



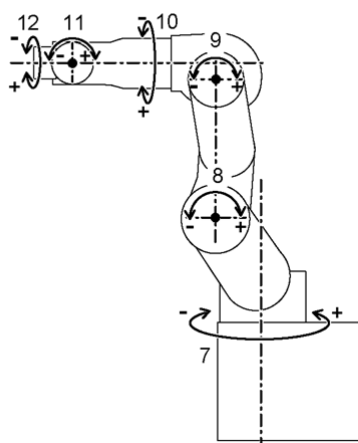
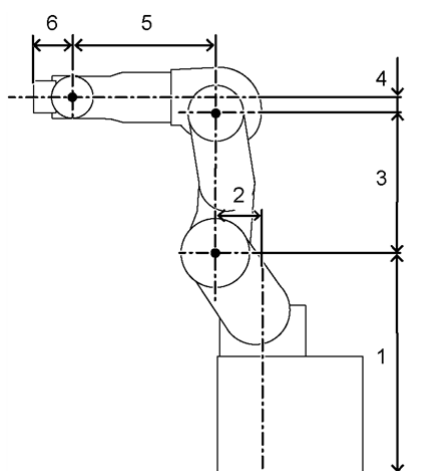
■ 水平多関節型ロボット (RSシリーズ)



RSシリーズは、この方向からのイラストです。



■ 垂直多関節型ロボット



記号	説明
a	第1関節
b	第2関節
c	増設アーム

参照

[Arm](#), [ArmClr](#)

ArmSet使用例

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

コマンドウィンドウでの操作例

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

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

3.5.50 ArmSet関数

増設アームの設定値を1つ返します。

書式

ArmSet(アーム番号, 設定値番号)

パラメーター

アーム番号

参照するアーム番号を、式または数値で指定します。

設定値番号

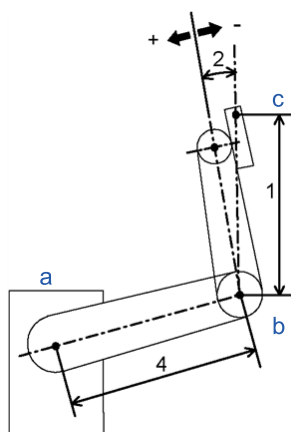
参照する設定値番号(0～5の整数)を、式または数値で指定します。(下記を参照してください。)

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

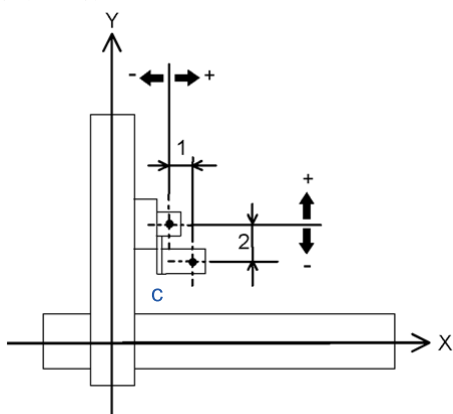
戻り値

上記表のいずれかを指定したパラメーターの設定値を、実数値で返します。

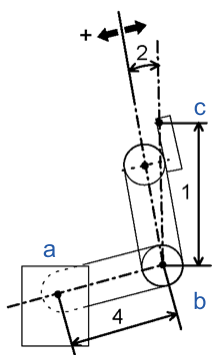
■ 水平多関節型ロボット



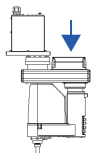
■ 直角座標型ロボット



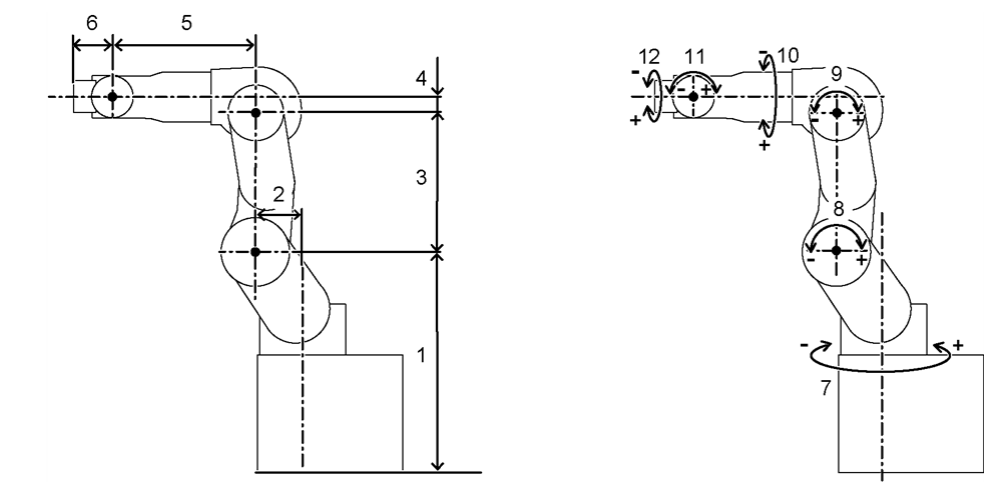
■ 水平多関節型ロボット (RSシリーズ)



RSシリーズは、この方向からのイラストです。



■ 垂直多関節型ロボット



記号	説明
a	第1関節
b	第2関節
c	増設アーム

解説
指定したアーム番号の増設アーム長が設定されていない場合、エラーが発生します。

参照
[ArmClr](#), [ArmSet](#)

ArmSet関数使用例

```
Real x
x = ArmSet(1, 1)
```

3.5.51 Asc関数

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

書式

Asc(置換文字列)

パラメーター

置換文字列

1文字以上の文字列を、文字列式または直接文字列で指定します。

戻り値

Asc関数に送られた文字列の、第1番目の文字に対する整数値を返します。

解説

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

注意

- ASCII値が返されるのは、第1文字についてだけです

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

参照

[Chr\\$関数](#), [InStr関数](#), [Left\\$関数](#), [Len関数](#), [Mid\\$関数](#), [Right\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Asc関数使用例

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

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

コマンドウィンドウでの操作例

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

3.5.52 Asin関数

指定数値のアークサインを返します。

書式

Asin(数値)

パラメーター

数値

角度のサインを実数値で指定します。

戻り値

指定した数値のアークサイン値を実数値 (単位: ラジアン) で返します。

解説

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

ラジアン値を、"度"に変換するためには、RadToDeg関数を使います。

参照

[Abs関数](#), [Acos関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [DegToRad関数](#), [RadToDeg関数](#), [Sgn関数](#), [Sin関数](#), [Tan関数](#), [Val関数](#)

Asin関数使用例

```
Function asintest
  Double x

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

3.5.53 AtHome関数

現在のロボット姿勢がHome位置か否かを返します。

書式

AtHome

戻り値

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

解説

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

参照

[Home](#), [HomeClr](#), [HomeDef関数](#), [HomeSet](#), [Hordr](#), [MCalComplete関数](#)

3.5.54 Atan関数

指定数値のアーктanジェントを返します。

書式

Atan(数値)

パラメーター

数値

角度のタンジェントを実数値で指定します。

戻り値

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

解説

Atanは、指定された数値のアークトanジェントを返します。指定する数値は、どんな数値でもかまいません。戻り値の範囲は、 $-\pi/2 \sim \pi/2$ ラジアンです。

ラジアン値を、"度"に変換するためには、RadToDeg関数を使います。

参照

[Abs関数](#), [Acos関数](#), [Asin関数](#), [Atan2関数](#), [Cos関数](#), [DegToRad関数](#), [RadToDeg関数](#), [Sgn関数](#), [Sin関数](#), [Tan関数](#), [Val関数](#)

Atan関数使用例

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

3.5.55 Atan2関数

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

書式

Atan2(X座標値, Y座標値)

パラメーター

X

X座標値を実数値で指定します。

Y

Y座標値を実数値で指定します。

戻り値

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

解説

Atan2 (X, Y) は、座標原点 (0, 0) と ([x座標値], [y座標値]) とを結ぶ直線の角度を返す関数です。

この三角関数はすべての4つの四分円でアーктanジェント値 (角度) を返します。

参照

[Abs関数](#), [Acos関数](#), [Asin関数](#), [Atan関数](#), [Cos関数](#), [DegToRad関数](#), [RadToDeg関数](#), [Sgn関数](#), [Sin関数](#), [Tan関数](#), [Val関数](#)

Atan2関数使用例

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

3.5.56 ATCLR

関節の実効トルクをクリアし、初期化します。

書式

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

パラメーター

関節指定1 -関節指定9

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

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

解説

ATCLRは指定した関節の実効トルク値をクリアします。

ATRQ実行前に、必ずATCLRを実行してください。

参照

[ATRQ](#), [PTRQ](#)

ATCLR使用例

[例1] 全関節の実効トルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

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

[例2] 垂直多関節ロボットにおいて、J1, J4, J5の実効トルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

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

3.5.57 ATRQ

指定した関節の実効トルク値を表示します。

書式

ATRQ [関節番号]

パラメーター

関節番号

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

結果

全関節について現在の実効トルク値を表示します。

解説

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

ATRQ実行前に、必ずATCLRを実行してください。

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

参照

[ATCLR](#), [ATRQ関数](#), [PTRQ](#)

ATRQ使用例

```
> atclr
> go p1
> atrq 1
    0.028
> atrq
    0.028    0.008
    0.029    0.009
    0.000    0.000
>
```


3.5.58 ATRQ関数

指定した関節の実効トルクを返します。

書式

ATRQ (関節番号)

パラメーター

関節番号

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

戻り値

0～1までの実数値で返します。

解説

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

ATRQ関数実行前に、必ずATCLRを実行してください。

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

参照

[ATRQ](#), [PTCLR](#), [PTRQ](#)

ATRQ関数使用例

プログラム中にATRQ関数を使用した例です。

```
Function CheckAvgTorque
  Integer i

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

3.5.59 AutoLJM

自動LJMを設定します。

書式

AutoLJM { On | Off }

パラメーター

On | Off

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

解説

AutoLJMは下記の命令で有効です。

Arc, Arc3, Go, Jump3, Jump3CP, Move

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

例えば Go LJM(P1)に対して、同じ効果を得るため

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

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

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

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

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

注意

- AutoLJMとLJM関数の2重適用について

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

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

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

- AutoLJM使用上の注意

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

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

参照

[AutoLJM関数](#), [LJM関数](#)

AutoLJM使用例

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

3.5.60 AutoLJM関数

AutoLJMの状態を返します。

書式

AutoLJM

戻り値

- 0 = 自動LJMオフ
- 1 = 自動LJMオン

参照

[AutoLJM](#)

AutoLJM関数使用例

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

3.5.61 AutoOrientationFlag

N6-A1000**の姿勢フラグを変更します。

書式

AutoOrientationFlag { On | Off }

パラメーター

On | Off

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

解説

AutoOrientationFlagは、下記の命令で有効です。

Go, BGo, TGo, Jump3, JumpTLZ

下記の姿勢フラグを変更します。

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

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

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

LJM関数との併用

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

AutoOrientationFlag使用例

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

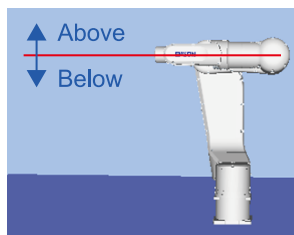
```
Go P1
Go P2
```

- AutoOrientationFlagを"ON"にした場合

P点と赤線との位置の関係により、次のようにフラグが変更されます。

- P点が赤線より上の位置: Above

- P点が赤線より下の位置: Below



3.5.62 AutoOrientationFlag関数

AutoOrientationFlagの状態を返します。

書式

AutoOrientationFlag

戻り値

- 0 = AutoOrientationFlagオフ
- 1 = AutoOrientationFlagオン

参照

[AutoOrientationFlag](#)

AutoOrientationFlag関数使用例

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

3.5.63 AutoSTM

自動STM機能を設定します。

書式

- (1) AutoSTM モード指定
- (2) AutoSTM

モード指定

■ RSシリーズ

STM_H_MODE_FASTEST: 自動STM機能が有効になります。PTP動作命令を実行した場合に、動作時間が最小となるように姿勢フラグを変更します。J1Flag、J2Flagおよび腕姿勢 (Handフラグ) が変更されます。X,Y,Z およびU軸の値は変更されません。動作時間が最小となる姿勢フラグが複数存在する場合は、姿勢フラグの変化が少ないものが選択されます。

STM_OFF: 自動STM機能は無効になります。

■ その他の機種

AutoSTMの対象外です。

解説

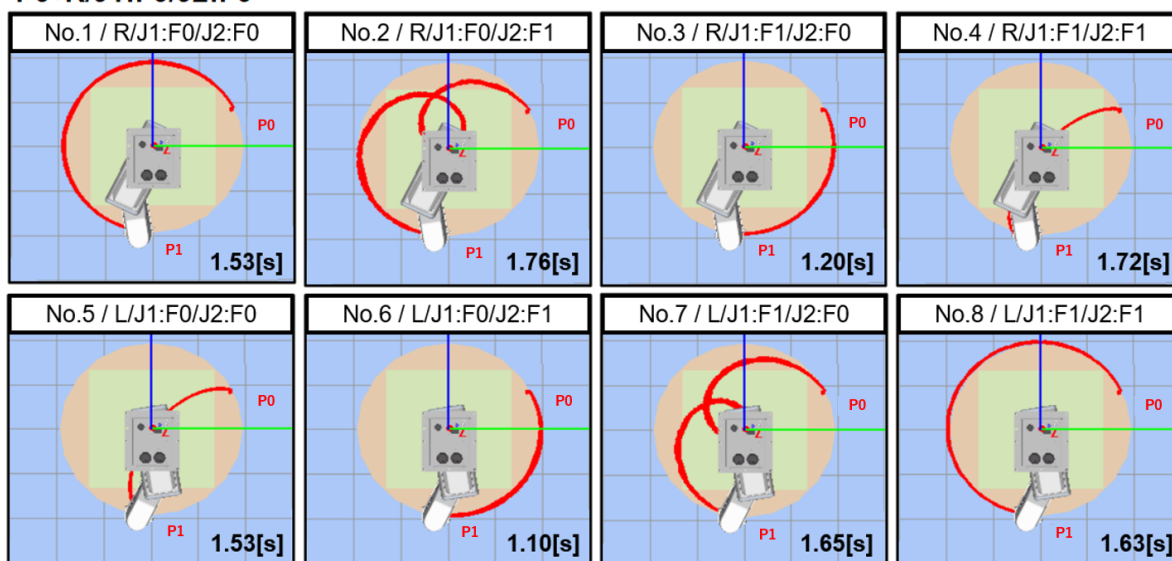
モード指定を省略した場合、AutoSTMを設定します。

モード指定を指定した場合、現在のAutoSTMの状態を表示します。

RSシリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、姿勢フラグを正しく指定しないと、第1アーム、第2アームが意図しない方向へ回転することがあります。また、どの姿勢フラグを選択すれば動作時間が最小になるのか、実行しなければわかりません。AutoSTMはアームの意図しない回転を防止するとともに、動作時間が最小となるように動作します。これによりタクトタイムの短縮が期待できます。

下の図のポイントP0からポイントP1にPTP動作することを考えます。RSシリーズでは、J1Flag、J2Flagおよび腕姿勢の組み合わせがあるため、姿勢フラグの選び方は合計で8パターンあります。図中の赤線はポイントP0からそれぞれの姿勢フラグに対応するポイントP1に動作したときの動作軌跡を表しています。また、図の左下の数値は動作時間を表します。AutoSTMを有効にすると、複数ある姿勢フラグのうち動作時間が最小となるものにむかって動作します。下の図の例ではNo.6の左腕系、J1F0、J2F1の姿勢フラグが選ばれます。

P0:R/J1:F0/J2:F0



AutoSTMは下記の命令で有効です。

Go, BGo, TGo, Pass, Jump

動作時間は、関数実行時に設定されているWeight、Inertia設定値およびSpeed,Accel設定をもとに計算されます。これらの設定値を変更した場合、AutoSTMによる姿勢フラグの変換結果が変わる可能性があります。AutoSTMのテストを行う場合は、実環境での速度加速度設定を用いてください。

パワーモードの設定値、SpeedFactorの設定値、およびSLS速度調整値に関わらず、STM関数による姿勢フラグの変換結果は変わりません。パワーモードがLowの場合、パワー出力が制限される前のSpeedおよびAccelをもとに動作時間の計算を行い姿勢フラグを変換します。テストモードかAUTOモードかによって、姿勢フラグの変換結果は変わりません。

パルス動作範囲外の姿勢フラグは選択されません。

AutoSTMを有効にすると、STM関数を適用したときと同様に動作時間が最小となるように動作を実行します。例えば、一連の動作に対してSTM関数を適用したい場合、STM関数のみを用いた場合、プログラムは以下のようになります。

```
Go STM(P1)
Go STM(P2)
Go STM(P3)
```

AutoSTMを用いると、同じ効果を得るため以下のようにプログラムを記述できます。

```
AutoSTM STM_H_MODE_FASTEST
Go P1
Go P2
Go P3
AutoSTM STM_OFF
```

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

下記の場合、AutoSTMは無効状態になります。

- コントローラー起動時
- Reset実行時
- 全タスク中断時
- Motor On実行時

操作モードがTEACHモードの場合は、AutoSTMの設定が無視されます。

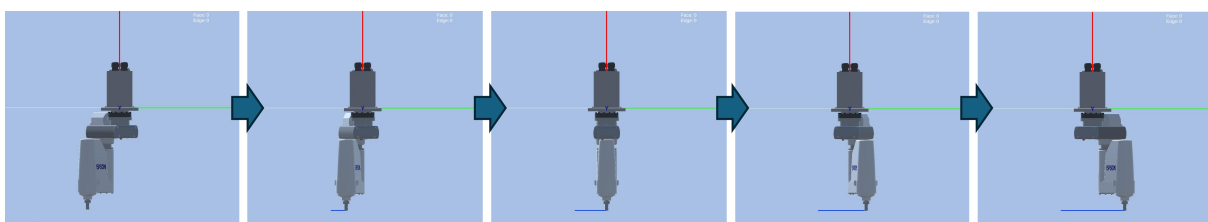
AutoSTMとAutoLJM機能は同時に有効にできません。AutoSTMが有効な状態で、AutoLJMを有効にした場合、AutoSTMは無効状態、AutoLJMは有効状態になります。AutoLJMが有効な状態で、AutoSTMを有効にした場合、AutoLJMは無効状態、AutoSTMは有効状態になります。

注意

- 配線、装置レイアウトについて

AutoSTM機能を使用した場合、動作前後で腕姿勢が切り替わることがあります。腕姿勢が切り替わる場合、アームを畳みこむような動作になるため、配線に注意してください。

また、右腕左腕系が切り替わっても周辺機器との干渉がないようにレイアウトしてください。下図は腕姿勢が切り替わった時の動作例です。



- 姿勢フラグの違いによる誤差について

本コマンドは粗位置決めでの使用を想定しています。同一ポイントを右腕系と左腕系で位置決めした場合、ツール設定の誤差やキャリブレーションの誤差により、位置ずれが生じる場合があります。精密な位置合わせが要求される作業では本コマンドを使用せずティーチしたポイントを使用してください。

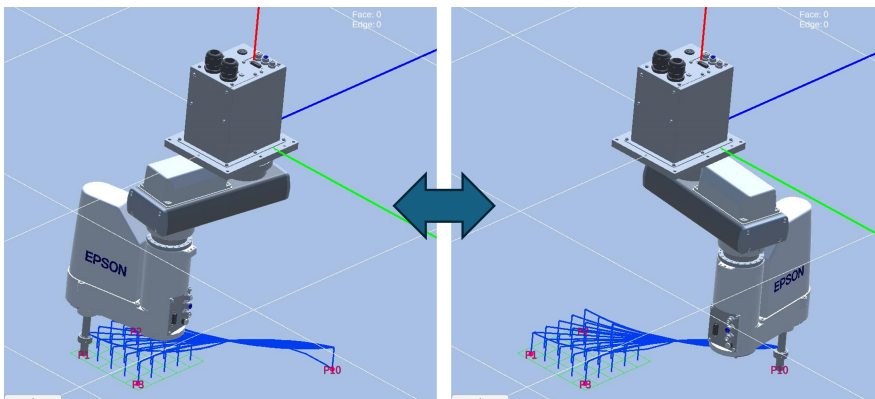
参照

[Pallet](#), [STM関数](#), [AutoSTM関数](#), [AutoLJM](#), [Power](#), [SpeedFactor](#), [Speed](#), [Accel](#), [Weight](#), [Inertia](#), [SF_LimitSpeedS](#)

AutoSTM使用例

以下のプログラムはRSシリーズでのAutoSTMの使用例です。パレット-指定点間をJump動作で往復します。動作時間が最短になるように姿勢フラグが選択されています。

```
Function AutoSTM_Example1
  Integer i;
  ' モーター、速度、加速度設定
  Motor On; Power High;
  Speed 20; Accel 20, 20;
  Weight 1.0; Inertia 0.005;
  P1 = XY(-200, -200, -50, 0)
  P2 = XY(-200, -50, -50, 0)
  P3 = XY(-50, -200, -50, 0)
  P10 = XY(150, 150, -50, 0)
  ' パレットの設定
  Pallet 1, P1, P2, P3, 5, 5
  Pulse 0, 0, 0, 0
  ' AutoSTM機能を有効にする
  AutoSTM STM_H_MODE_FASTEST
  For i = 1 To 25
    Jump Pallet(1, i)
    Jump P10
  Next
  ' AutoSTM機能を無効にする
  AutoSTM STM_OFF
Fend
```



3.5.64 AutoSTM関数

自動STM機能のモード指定を返します。

書式

AutoSTM

戻り値

- 0 = STM_OFF: STM機能は無効状態です
- 1 = STM_H_MODE_FASTEST: STM機能は有効状態です

参照

[STM関数](#), [AutoSTM](#)

AutoSTM関数使用例

以下のファンクションをRSシリーズの機種で実行すると「AutoSTM mode is STM_H_MODE_FASTEST.」が表示されます。

```
Function AutoSTMTest
  AutoSTM STM_H_MODE_FASTEST
  If AutoSTM = STM_H_MODE_FASTEST Then
    Print "AutoSTM mode is STM_H_MODE_FASTEST."
  EndIf
  AutoSTM STM_OFF
Fend
```

3.5.65 AvgSpeedClear

関節速度の絶対値の平均をクリアし、初期化します。

書式

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

パラメーター

関節指定1 -関節指定9

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

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

解説

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

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

本コマンドは、PG付加軸はサポートしません。

参照

[AvgSpeed](#), [PeakSpeed](#)

AvgSpeedClear使用例

[例1] 全関節の平均速度値をクリアしたのち、指定した関節番号の平均速度値を表示させるコマンド実行例です。

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

[例2] 垂直多関節ロボットにおいて、第1関節, 第4関節, 第5関節の平均速度値をクリアしたのち、指定した関節番号の平均速度値を表示させるコマンド実行例です。

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

3.5.66 AvgSpeed

指定した関節の速度の絶対値の平均値を表示します。

書式

AvgSpeed [関節番号]

パラメーター

関節番号

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

結果

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

解説

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

平均値が0.001以下の値になる場合は "0" になります。

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

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

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

本コマンドは、PG付加軸はサポートしません。

参照

[AvgSpeedClear](#), [AvgSpeed関数](#), [PeakSpeed](#)

AvgSpeed使用例

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

3.5.67 AvgSpeed関数

指定した関節の速度の絶対値の平均値を返します。

書式

AvgSpeed (関節番号)

パラメーター

関節番号

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

戻り値

0～1までの実数値で返します。

解説

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

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

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

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

本関数は、PG付加軸はサポートしません。

参照

[AvgSpeed](#), [AvgSpeedClear](#), [PeakSpeed](#)

AvgSpeed関数使用例

プログラム中にAvgSpeed関数を使用した例です。

```
Function CheckAvgSpeed
  Integer i

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

3.5.68 AvoidSingularity

特異姿勢通過機能を設定します。

書式

AvoidSingularity { mode }

パラメーター

mode

使用する特異点通過モードを表す整数式

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

解説

AvoidSingularityのSING_THRU,SING_THRUROT,SING_AVOID は下記の命令で有効です。

Move, Tmove, Bmove, Arc, Arc3, Jump3, Jump3CP, JumpTLZ

AvoidSingularityのSING_VSD, SING_AUTOは下記の命令で有効です。

Move, Tmove, Bmove, Arc, Arc3, Jump3, Jump3CP, JumpTLZ, Cvmove

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

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

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

特異姿勢通過機能を設定した場合でも加速度エラーや過速度エラーが発生する場合は、AccelS,SpeedSを小さくしてください。

コントローラー起動時は、AvoidSingularityはコントローラー設定で指定した設定状態 (工場出荷時: 1) になります。また、AvoidSingularityの設定値を変更した場合、SingularityAngle, SingularitySpeed, SingularityDistの各パラメーターはデフォルト値に戻ります。SING_AUTOは、SING_THRUとSING_VSDを合わせたモードです。動作または速度によってSING_THRUかSING_VSDが選択されます。

AvoidSingularityを使用しても加速度エラーや過速度エラーが発生する場合は、AccelS, DecelS, SpeedS を小さくしてください。

注意

■ 垂直6軸ロボットとNシリーズロボットの特異姿勢近傍の条件設定

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

■ RSシリーズロボットの特異姿勢近傍の条件設定

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

■ Nシリーズロボットの注意事項

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

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

Nシリーズは、手首特異点姿勢と腕特異点姿勢の他に、肘特異姿勢領域を持ちます。肘特異姿勢領域は、第3関節が0度のとき (第3関節と第2関節が重なったとき)の姿勢です。

肘特異姿勢領域の通過動作についての詳細は、以下のマニュアルを参照してください。
"Epson RC+ ユーザーズガイド"

■ SING_THRUとSING_AVOIDの違い

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

参照

[AvoidSingularity関数](#), [SingularityAngle](#), [SingularitySpeed](#), [SingularityDist](#)

AvoidSingularity使用例

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


3.5.69 AvoidSingularity関数

AvoidSingularityの状態を返します。

書式

AvoidSingularity

戻り値

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

参照

[AvoidSingularity](#)

AvoidSingularity関数使用例

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

3.6 B

3.6.1 Base

ベース座標系の定義と表示をします。

書式

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

パラメーター

座標系データ

ベース座標系の原点と向きをポイントデータで直接指定します。

原点

ベース座標系原点を定義するロボット座標系上の位置を、P#(整数)またはP(式)で指定します。

X軸指定

ベース座標系のX軸上の点を定義するロボット座標系上の位置を、P#(整数)またはP(式)で指定します。

Y軸指定

ベース座標系のY軸上の点を定義するロボット座標系上の位置を、P#(整数)またはP(式)で指定します。

X

X軸指定を優先的にX軸に一致させます。省略可能です。(デフォルト)

Y

Y軸指定を優先的にY軸に一致させます。省略可能です。

解説

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

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

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

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

注意

ベース座標系を変更すると、すべてのローカル定義に影響があります。

ベース座標系を変更したら、ローカル座標系をすべて定義し直してください。

参照

[Local](#)

Base使用例

ベース座標系の原点をX軸100mm、Y軸100mm地点に定義します。

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

3.6.2 BClr関数

指定した数値の1ビットをクリアし、その値を返します。

書式

BClr (数値, ビット番号)

パラメーター

数値

ビットをクリアする数値を、式または直接数値で指定します。

ビット番号

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

戻り値

ビットをクリアした値 (整数)を返します。

参照

[BClr64関数](#), [BSet関数](#), [BSet64関数](#), [BTst関数](#), [BTst64関数](#)

BClr関数使用例

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

3.6.3 BClr64関数

指定した数値の1ビットをクリアし、その値を返します。

書式

BClr64 (数値, ビット番号)

パラメーター

数値

ビットをクリアする数値を、式または直接数値で指定します。

ビット番号

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

戻り値

ビットをクリアした値 (整数)を返します。

参照

[BClr関数](#), [BSet関数](#), [BSet64関数](#), [BTst関数](#), [BTst64関数](#)

BClr64関数使用例

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

3.6.4 BGo

選択したローカル座標系での、オフセットPTP動作を実行します。

書式

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

パラメーター

目標座標

ポイントデータを用いて動作の目標位置を指定します。

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

!並列処理!

動作中に、I/Oその他のコマンドを実行するよう、並列処理ステートメントを付け加えることができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

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

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

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

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

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

!並列処理!を使って、動作と並列に他の処理を実行することができます。

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

参照

[Accel](#), [BMove](#), [Find](#), [!...! 並列処理](#), [P# \(2. ポイント指定\)](#), [Speed](#), [Till](#), [TGo](#), [TMove](#), [Tool](#)

BGo使用例

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

Function BGoTest
```

```
Speed 50
Accel 50, 50
Power High

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

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

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

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

Fend
```

[出力結果]

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

3.6.5 BMove

選択したローカル座標系で、オフセット直線補間動作を実行します。

書式

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

パラメーター

目標座標

ポイントデータを用いて動作の目標位置を指定します。

ROT

ツール姿勢変化を優先させて、速度と加減速度を指定します。省略可能です。

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find  
Till Sw(式) = {On | Off}  
Find Sw(式) = {On | Off}
```

! 並列処理 !

動作中にI/Oその他のコマンドを実行するために、並列処理ステートメントをつけ加えることができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

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

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

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

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

ロボットのツール先端位置を、特定の座標に固定したままツール姿勢のみを変化させようとした場合や、ツール先端の移動距離に対してツール姿勢変化が大きい場合、ツール姿勢変化速度が著しく速くなる恐れがあります。これを防ぐため、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働きます。

CP動作時のツール姿勢変化速度の上限値を手動で設定したい場合は、SpeedRLimitationをオンにしてください。SpeedRLimitationをオンにした場合、CP動作時のツール姿勢変化速度が設定したSpeedRを超える場合は、ツール姿勢変化速度がSpeedRになるように動作速度を制限します。ツール姿勢変化速度が設定したSpeedRを超えない場合は、設定したSpeedSで動作します。ツール姿勢変化速度の上限値はSpeedRであらかじめ設定してください。

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

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

!並列処理!を使って、動作と並列に他の処理を実行できます。

注意

■ BMoveをCPとともに用いる

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

参照

[AccelS](#), [BGo](#), [Find](#), [!...! 並列処理](#), [P# \(2. ポイント指定\)](#), [SpeedS](#), [TGo](#), [Till](#), [TMove](#), [Tool](#)

BMove使用例

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

```
Function BMoveTest
```

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

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

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

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

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

```
Fend
```

[出力結果]

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


3.6.6 Boolean

Boolean型変数を宣言します。(サイズ:2バイト)

書式

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

パラメーター

変数名

Boolean型と宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。

各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

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

解説

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

参照

[Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Boolean使用例

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

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

3.6.7 BOpen

ファイルをバイナリーモードで開きます。

書式

BOpen ファイル名 As #ファイル番号

・
・

Close #ファイル番号

パラメーター

ファイル名

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

ファイル番号

30～63の整数値、または式で指定します。

解説

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

注意

ネットワークパスが使用できます。

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

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

Closeステートメントでファイルを閉じ、ファイル番号を解放します。

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

参照

[Close](#), [AOpen](#), [FreeFile関数](#), [ReadBin](#), [ROpen](#), [UOpen](#), [WOpen](#), [WriteBin](#)

BOpen使用例

```
Integer fileNum, i

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

Flush #fileNum
Seek #fileNum, 10
ReadBin #fileNum, i
```

```
Print "data = ", i  
Close #fileNum
```

3.6.8 Box

進入検出エリアの設定と表示をします。

書式

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

パラメーター

エリア番号

設定するエリアの番号を1～15の整数値で指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。(1) の書式で省略した場合、現在選択されているロボットが対象となります。(2), (3) の書式では、省略できません。

X軸下限位置

設定するエリアの下限位置のX座標値 (実数) を、数値または式で指定します。

X軸上限位置

設定するエリアの上限位置のX座標値 (実数) を、数値または式で指定します。

Y軸下限位置

設定するエリアの下限位置のY座標値 (実数) を、数値または式で指定します。

Y軸上限位置

設定するエリアの上限位置のY座標値 (実数) を、数値または式で指定します。

Z軸下限位置

設定するエリアの下限位置のZ座標値 (実数) を、数値または式で指定します。

Z軸上限位置

設定するエリアの上限位置のZ座標値 (実数) を、数値または式で指定します。

リモート出力論理

On | Off

リモート出力の論理を設定します。Box進入時にI/O出力をOnしたい場合には、Onを設定、Box進入時にI/O出力をOffしたい場合には、Offを設定します。パラメーターを省略した場合は、Onが設定されます。

ローカル番号

ローカル座標系番号 (0から15) を指定します。

番号の前に、必ず"/LOCAL"をつけてください。パラメーターを省略した場合は、ローカル座標系番号0が設定されます。

結果

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

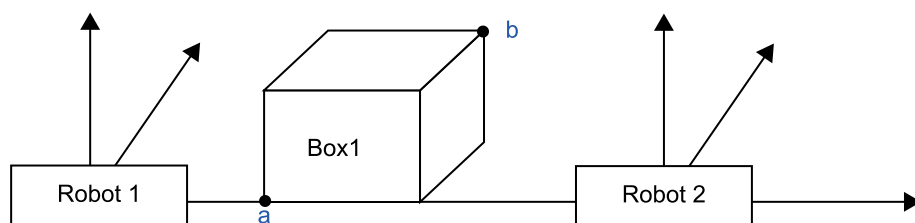
解説

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

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

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

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



記号	説明
a	X, Y, Z軸下限位置
b	X, Y, Z軸上限位置

Robot 1から見たBox1の設定

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

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

Robot 2から見たBox1の設定

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

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

注意

- 各座標軸に進入検出エリアの設定をオフにする

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

この場合、XYの2次元平面に進入しているかどうかを判別します。

- 進入検出エリアのデフォルト

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

- ツール選択

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

- 付加軸

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

ヒント

- Boxをロボットマネージャーで設定する

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

参照

[BoxClr](#), [BoxDef関数](#), [GetRobotInsideBox関数](#), [InsideBox関数](#), [Plane](#)

Box使用例

[例1] 下記はコマンドウィンドウからBox値を設定し、値を表示する簡単な操作例です。

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

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

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

```
Function SetBox
    Integer i

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

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

Fend
```

3.6.9 Box関数

設定された進入検出エリアを返します。

書式

Box(エリア番号[, ロボット番号], 参照データ)

パラメーター

エリア番号

確認するエリア番号を式または数値で指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

参照データ

戻り値として返すデータを整数値で指定します。- 1: 下限位置 - 2: 上限位置

戻り値

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

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

参照

[Box](#), [BoxClr](#), [BoxDef関数](#), [GetRobotInsideBox関数](#), [InsideBox関数](#)

Box関数使用例

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

3.6.10 BoxClr

設定されたBoxをクリアします。

書式

BoxClr エリア番号[, ロボット番号]

パラメーター

エリア番号

設定をクリアにする進入検出エリア番号 (1～15の整数)を、式または数値で指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

参照

[Box](#), [BoxDef関数](#), [GetRobotInsideBox関数](#), [InsideBox関数](#)

BoxClr関数使用例

下記はBoxClr関数を使用したプログラムです。

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


3.6.11 BoxDef関数

進入検出エリアが設定されているかどうかを返します。

書式

BoxDef(エリア番号[, ロボット番号])

パラメーター

エリア番号

状態を返す進入検出エリアの番号 (1～15の整数)を指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

戻り値

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

参照

[Box](#), [BoxClr](#), [GetRobotInsideBox関数](#), [InsideBox関数](#)

BoxDef関数使用例

下記はBoxDef関数を使用したプログラムです。

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

3.6.12 Brake

現在のロボットの指定した関節のブレーキを、オン、またはオフします。

書式

Brake 状態, 関節番号

パラメーター

状態

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

関節番号

1～6までの関節番号を指定します。

解説

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

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

Brake命令を実行すると、ロボット制御パラメーターを初期化します。詳細は、Motor Onを参照してください。

警告

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

注意

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

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

参照

[Motor](#), [Power](#), [Reset](#), [SFree](#), [SLock](#)

Brake使用例

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

3.6.13 Brake関数

指定した関節のブレーキ状態を返します。

書式

Brake (関節番号)

パラメーター

関節番号

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

戻り値

0 = ブレーキOff, 1 = ブレーキOn

参照

[Brake](#)

Brake関数使用例

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

3.6.14 BSet関数

指定した数値の1ビットをセットし、その結果を返します。

書式

BSet (数値, ビット番号)

パラメーター

数値

ビットをセットする数値を、式または直接数値で指定します。

ビット番号

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

戻り値

ビットをセットした値 (整数)を返します。

参照

[BClr関数](#), [BClr64関数](#), [BSet64関数](#), [BTst関数](#), [BTst64関数](#)

BSet関数使用例

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

3.6.15 BSet64関数

指定した数値の1ビットをセットし、その結果を返します。

書式

BSet64 (数値, ビット番号)

パラメーター

数値

ビットをセットする数値を、式または直接数値で指定します。

ビット番号

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

戻り値

ビットをセットした値 (整数)を返します。

参照

[BClr関数](#), [BClr64関数](#), [BSet関数](#), [BTst関数](#), [BTst64関数](#)

BSet関数使用例

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

3.6.16 BTst関数

数値の1ビット状態を返します。

書式

BTst (数値, ビット番号)

パラメーター

数値

ビットテストをする数値を、式または直接数値で指定します。

ビット番号

テストするビット番号 (0～31の整数値)を数値で指定します。

戻り値

ビットテストの値を1または0で返します。

参照

[BClr関数](#), [BClr64関数](#), [BSet関数](#), [BSet64関数](#), [BTst64関数](#)

BTst関数使用例

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

3.6.17 BTst64関数

数値の1ビット状態を返します。

書式

BTst64 (数値, ビット番号)

パラメーター

数値

ビットテストをする数値を、式または直接数値で指定します。

ビット番号

テストするビット番号 (0～63の整数値)を数値で指定します。

戻り値

ビットテストの値を1または0で返します。

参照

[BClr関数](#), [BClr64関数](#), [BSet関数](#), [BSet64関数](#), [BTst関数](#)

BTst64関数使用例

```
Int64 flags
flags = BSet64(flags, 1)
If BTst64(flags, 1) = 1 Then
    Print "Bit 1 is set"
EndIf
```

3.6.18 Byte

Byte型の変数を宣言します。(サイズ: 2バイト)

書式

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

パラメーター

変数名

Byte型と宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

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

参照

[Boolean](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Byte使用例

次はByteを使って、Bit7の値を表示するプログラム例です。

```
Function main
  Byte varByte

  varByte = 127
  Call BitCheckByte(varByte)

  varByte = -1
  Call BitCheckByte(varByte)
Fend

Function BitCheckByte(var As Byte)
  Print "Value:", var, " Bit7 =", BTst(var, 7)
Fend
```

[出力結果]

```
Value: 127 Bit7 = 0
Value: -1 Bit7 = 1
```


3.7 C

3.7.1 Calib

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

書式

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

パラメーター

関節番号

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

解説

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

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

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

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

注意

- Calibコマンドの使用には注意が必要です

Calibはメンテナンス作業だけの目的に設計されています。必要なとき以外は使わないでください。

Calibを実行するとHofs値が置き換えられます。Hofs値が知らない間に変更されていると、ロボットが思わぬ動きをすることになりますので、Calibは必要なとき以外は絶対に使わないでください。

関節精度補正の対応機種では、Calibを実行すると、指定した軸についてJointAccuracyで設定した補正值が"0"になります。

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

Safety基板を搭載したコントローラーでは、コントローラーのHofs値と安全機能を実現するSafety基板のHofs値が一致する必要があります。

本コマンドを実行すると、コントローラーのHofs値のみが変更され、Safety基板設定と差異が生じるため、ワーニングが発生します。

このため、本コマンド実行後は、Safety基板設定を更新するために、安全機能マネージャーを起動してください。

詳細は、以下のマニュアルを参照してください。
"ロボットコントローラー 安全機能マニュアル"

起こりやすいエラー

関節番号が指定されていないとエラーになります

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

参照

[CalPls](#), [JointAccuracy](#), [HofsJointAccuracy](#), [Hofs](#)

Calib使用例

コマンドウィンドウからの操作

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

3.7.2 Call

ファンクションをサブルーチンとして呼び出します。

書式

Call ファンクション名 [(引数のリスト)]

パラメーター

ファンクション名

呼び出すファンクション名を指定します。

引数のリスト

ファンクション宣言で指定された引数のリストを指定します。引数は、下記の書式を使用してください。省略可能です。[ByRef]変数名[()]、または数式

ByRef

呼び出すファンクションの変数を参照する場合は、ByRefを指定します。この場合、ファンクション内での引数の変更を呼び出し側の変数に反映させることができます。参照で渡された値は変更できます。ByValを指定して変数を参照する場合、ファンクション宣言内でも変数をByRef指定する必要があります。

解説

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

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

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

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

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

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

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

配列変数を引数として渡すときには、ByRefが必須となります。

参照

[Function...Fend, GoSub...Return](#)

Call使用例

```
[File1: MAIN.PRG]
Function main
    Call InitRobot
Fend

[File2: INIT.PRG]
Function InitRobot
```

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

3.7.3 CalPls

キャリブレーションに使用する位置姿勢パルス値の設定と表示をします。

書式

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

(2) CalPls

パラメーター

第1関節パルス値

第1関節のパルス値 (整数)を式または数値で指定します。

第2関節パルス値

第2関節のパルス値 (整数)を式または数値で指定します。

第3関節パルス値

第3関節のパルス値 (整数)を式または数値で指定します。

第4関節パルス値

第4関節のパルス値 (整数)を式または数値で指定します。

第5関節パルス値

第5関節のパルス値 (整数)を式または数値で指定します。省略可能です。

第6関節パルス値

第6関節のパルス値 (整数)を式または数値で指定します。省略可能です。

第7関節パルス値

第7関節のパルス値 (整数)を式または数値で指定します。省略可能です。

第8関節パルス値

第8関節のパルス値 (整数)を式または数値で指定します。省略可能です。

第9関節パルス値

第9関節のパルス値 (整数)を式または数値で指定します。省略可能です。

戻り値

パルス値を省略すると、現在設定されているパルス値を表示します。

解説

キャリブレーションを行うための位置の、正しいパルス値を入力、保存します。

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

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

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

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

注意

CalPls値は電源オフによって変更できません

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

参照

[Calib](#), [Hofs](#)

CalPls使用例

モニターウィンドウ操作

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

3.7.4 CalPls関数

CalPlsで設定されたキャリブレーションに使用するパルス値を返します。

書式

CalPls(関節番号)

パラメーター

関節番号

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

戻り値

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

参照

[CalPls](#)

CalPls関数使用例

CalPls関数を使用したプログラム例です。

```
Function DisplayCalPlsValues
  Integer i

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

3.7.5 ChDir

カレントディレクトリの変更と表示をします。

書式

- (1) ChDir パス名
- (2) ChDir

パラメーター

パス名

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

解説

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

ディスクがPCの場合に実行できます。

本命令をプログラムから実行するときは、パス名を["]で囲ってください。

接続されているコントローラーで最後にビルドしたプロジェクトがあったディレクトリがカレントディレクトリとなります。コントローラーにビルド情報がない場合は、ルートディレクトリがカレントディレクトリとなります。

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

参照

[ChDrive](#), [ChDisk](#), [CurDir\\$関数](#)

ChDir使用例

コマンドウィンドウ操作例

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

プログラム実行例

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


3.7.6 ChDisk

ファイル操作する対象ディスクを設定します。

書式

ChDisk PC|USB|RAM|FLASH

パラメーター

PC

PC内のフォルダー (ハードディスクなど)

USB

コントローラーに接続されたUSBメモリー

RAM

コントローラー内のメモリー

FLASH

コントローラー内のプロジェクトフォルダー

解説

ファイル操作の対象ディスクを指定します。デフォルトはPCです。

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

- PC: PC内のフォルダーを対象とします。

電源投入時の設定はPCで、通常のケースではPCから変更する必要はありません。

プロジェクトフォルダなどに置かれたファイルにアクセスできます。

- USB: コントローラーのメモリーポートに接続したUSBメモリーを対象とします。RC+との連携を行わないなど、PC部を使用しないケースで、ファイルとのやりとりを行いたい場合に使用します。

T/VTシリーズでは、パラメーターにUSBを指定することはできません。

- RAM: メモリー上の一時ファイルを対象とします。

コントローラーの電源を落とすとファイルは保存されません。

一時的なデータの保管に使用します。

- FLASH: コントローラー内のプロジェクトフォルダーを対象とします。

コントローラーの電源を落としても削除されません。

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

- ChDisk, ChDrive, ChDirの影響を受けない
 - Curve, CVMove, LoadPoints, SavePoints, ImportPoints のファイル名
 - 常にプロジェクトのフォルダーを対象とする。
 - ファイル名のみが指定可能でパスを指定するとエラー
- ChDiskの影響を受けない
 - OpenDBのAccess, Excelファイル名, ImportPointsのソースパス, VLoadModel, VSaveImage, VSaveModel
 - 常にWindowsのフォルダーを対象とする。

- ファイル名のみを指定した場合は、カレントドライブやカレントフォルダーの影響を受ける。フルパスの指定も可能
- ChDiskがPCの場合のみ実行可能
 - ChDir, FolderExists, Mkdir, Rmdir, Rmdir
 - ChDiskがPC以外で実行するとエラー。
 - ファイル名やディレクトリー名のみを指定した場合は、カレントドライブやカレントフォルダーの影響を受ける。フルパスの指定も可能
- USBとRAMにはディレクトリーの概念がありません。
- ChDiskがUSB、RAM、FLASHでも実行可能
 - Copy, Del, FileDateTime, FileExist, FileLen, AOpen, BOpen, ROpen, UOpen, WOpen, Rename
 - ChDiskがPCの場合: ファイル名やディレクトリー名のみを指定した場合は、カレントドライブやカレントフォルダーの影響を受ける。フルパスの指定も可能
 - ChDiskがUSBやRAMの場合: ファイル名のみが指定可能でパスを指定するとエラー
- 特殊
 - Declare
 - 詳細はDeclareを参照。指定されたファイル名をそのまま処理。カレントドライブやカレントフォルダーの影響を受けない

ChDiskがPCの場合のフルパスの決定方法は下記の通りです。

- ファイル名のみ "abc.txt"

カレントドライブ+カレントディレクトリー+指定ファイル名

```
"C:\EpsonRC80\Projects\ProjectName\abc.txt"
```

- ドライブなしのフルパス "\abc.txt"

カレントドライブ+指定フルパス

```
"C:\abc.txt"
```

- ドライブありのフルパス "d:\abc.txt"

指定フルパスそのまま

```
"d:\abc.txt"
```

- ドライブがネットワークフォルダー "k:\abc.txt"

指定フルパスそのまま

```
"k:\abc.txt"
```

- ネットワークパス "\\Epson\data\abc.txt"

指定フルパスそのまま

```
"\\Epson\data\abc.txt"
```

ChDiskの設定は、コントローラーで1つです。

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

参照

[ChDir](#), [ChDrive](#), [CurDisk\\$関数](#)

ChDisk使用例

コマンドウィンドウ操作例

```
> ChDisk PC
```

3.7.7 ChDrive

カレントドライブを変更します。

書式

ChDrive ドライブ名

パラメーター

ドライブ名

有効なドライブ (1文字)を指定します。

解説

ディスクがPCの場合に実行できます。

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

詳細は、ChDiskを参照してください。

参照

[ChDir](#), [ChDisk](#), [CurDrive\\$関数](#)

ChDrive使用例

コマンドウィンドウ操作例

```
> ChDrive d
```

3.7.8 ChkCom関数

通信ポートの受信バッファ内の文字数を返します。

書式

ChkCom (通信ポート番号)

パラメーター

通信ポート番号

RS-232Cのポート番号の整数値を指定します。

- SPEL+制御部: 1～4
- PC部: 1001～1008

戻り値

受信文字数(整数値)を返します。

受信データが存在しない場合、ポートの状態を下記の負の値で返します。

- -2: ポートが他のタスクで使用中
- -3: ポートがオープンされていない

参照

[CloseCom](#), [OpenCom](#), [Read](#), [Write](#)

ChkCom関数使用例

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

3.7.9 ChkNet関数

ネットワークポートの受信バッファ内の文字数を返します。

書式

ChkNet(通信ポート番号)

パラメーター

通信ポート番号

TCP/IPのポート番号 (201～216)を指定します。

戻り値

受信文字数 (整数値)を返します。

受信データが存在しない場合、ポートの状態を下記の負の値で返します。

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

参照

[CloseNet](#), [OpenNet](#), [Read](#), [Write](#)

ChkNet関数使用例

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

3.7.10 Chr\$関数

ASCIIコードに対する文字を返します。

書式

Chr\$(コード番号)

パラメーター

コード番号

1～255までの整数値を指定します。

戻り値

指定したコードに対応する文字を返します。

解説

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

参照

[Asc関数](#), [InStr関数](#), [Left\\$関数](#), [Len関数](#), [Mid\\$関数](#), [Right\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Chr\$関数使用例

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

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

3.7.11 ClearHistory

コマンドの実行履歴を全て削除します。

書式

ClearHistory

解説

記憶している全てのコマンドの実行履歴を削除します。

注意

このコマンドはコマンドウィンドウのみ実行可能です。

参照

[History](#)

ClearHistory使用例

```
> Motor On
> Go P1
> History
Motor On
Go P1

> ClearHistory
> History
```


3.7.12 ClearPoints

ロボットの、メモリー上の位置データを消去します。

書式

ClearPoints

解説

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

参照

[PList](#), [LoadPoints](#), [SavePoints](#)

ClearPoints使用例

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

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

3.7.13 Close

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

書式

Close #ファイル番号

パラメーター

ファイル番号

30～63の整数値、または式で指定します。

解説

ファイル番号で参照中のファイルをクローズし、ファイル番号を開放します。

参照

[AOpen](#), [BOpen](#), [Flush](#), [FreeFile関数](#), [Input #](#), [Print #](#), [ROpen](#), [UOpen](#), [WOpen](#)

Close使用例

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

```
Integer fileNumber, i, j

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

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

3.7.14 CloseCom

OpenComでオープンしたRS-232Cポートをクローズします。

書式

CloseCom #通信ポート番号 | All

パラメーター

通信ポート番号

クローズするRS-232Cのポート番号を指定します。

- SPEL+制御部: 1～4
- PC部: 1001～1008

Allを指定するとそのタスクがオープンしたRS-232Cポートをすべてクローズします。

参照

[ChkCom関数](#), [OpenCom](#)

CloseCom使用例

```
CloseCom #1
```

3.7.15 CloseDB

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

書式

CloseDB #データベース番号

パラメーター

データベース番号

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

解説

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

注意

RC+ がインストールされているPCの接続が必要です

参照

[OpenDB](#), [SelectDB関数](#), [UpdateDB](#), [DeleteDB](#), [Input #](#), [Print #](#)

CloseDB使用例

OpenDB使用例を参照してください。

3.7.16 CloseNet

OpenNetでオープンしたTCP/IPポートをクローズします。

書式

CloseNet #通信ポート番号| All

パラメーター

通信ポート番号

クローズするTCP/IPのポート番号 (201～216)を指定します。

Allを指定するとそのタスクがオープンしたTCP/IPポートをすべてクローズします。

参照

[ChkNet関数](#), [OpenNet](#)

CloseNet使用例

```
CloseNet #201
```

3.7.17 Cls

Epson RC+ のRunウィンドウ, オペレーターウィンドウ, コマンドウィンドウのテキスト領域をクリアします。

TPのプリントパネルをクリアします。

書式

- (1) Cls #デバイスID
- (2) Cls

パラメーター

デバイスID

- 21 RC+
- 20 TP4

省略された場合はディスプレイデバイスが対象になります。

解説

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

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

デバイスIDが省略された場合は、ディスプレイデバイスの画面をクリアします。

Cls使用例

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

```
Function main
  Integer i

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

3.7.18 Cnv_AbortTrack

コンベヤーキューポイントへの追従動作を中断します。

書式

Cnv_AbortTrack [停止Z高さ]

パラメーター

停止Z高さ

停止時のロボットのZ位置を実数値で指定します。省略可能です。

解説

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

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

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

注意

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

参照

[Cnv_RobotConveyor関数](#)

Cnv_AbortTrack使用例

・ 追従するワークが下流コンベヤーに流れたロボットを監視するタスク

Function WatchDownstream

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

3.7.19 Cnv_Accel

コンベヤトラッキングの追従までの加速度、減速度を設定します。

書式

Cnv_Accel コンベヤ番号, 加速度・減速度

パラメーター

コンベヤ番号

コンベヤの番号を整数値 (1~16) で指定します。

加速度・減速度

追従動作の加速度, 減速度を設定します。

解説

コンベヤトラッキングの追従動作の加速度、および減速度を設定します。

加速度と減速度を別々に設定することはできません。

加速度指令エラーが発生する場合または、ワークをピックアップする時間を短縮する場合に変更してください。初期値は2000[mm/s²]です。

注意

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

参照

[Cnv_Accel関数](#)

Cnv_Accel使用例

```
Cnv_Accel 1,2000
```


3.7.20 Cnv_Accel関数

コンベヤトラッキングの追従までの加速度、減速度を返します。

書式

Cnv_Accel (コンベヤ番号)

パラメーター

コンベヤ番号

コンベヤの番号を式または数値 (1~16) で指定します。

戻り値

実数値 (単位: mm/s²) で返します。

注意

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

参照

[Cnv_Accel](#)

Cnv_Accel関数使用例

```
Print Cnv_Accel (1)
```

3.7.21 Cnv_AccelLim

コンベヤートラッキング追従後の加速度、減速度の制限値を設定します。

書式

Cnv_AccelLim コンベヤー番号, モード番号, 加速度・減速度

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16) で指定します。

モード番号

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

加速度・減速度

コンベヤートラッキング追従後の加速度, 減速度制限値を実数値 (単位: mm/s²) で設定します。設定範囲は AccelS コマンドと同じです。

初期値

- 個数優先モード: 500 [mm/s²]
- 精度優先モード: 2000 [mm/s²]
- 可変速コンベヤー対応モード: 6000 [mm/s²]

解説

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

加速度と減速度を別々に設定することはできません。

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

注意

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

参照

[Cnv_AccelLim関数](#)

Cnv_AccelLim使用例

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

3.7.22 Cnv_AccelLim関数

コンベヤートラッキング追従後の加速度、減速度の制限値を返します。

書式

Cnv_AccelLim (コンベヤー番号, モード番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

モード番号

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

戻り値

実数値 (単位: mm/s²)で返します。

注意

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

参照

[Cnv_AccelLim](#)

Cnv_AccelLim関数使用例

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

3.7.23 Cnv_Adjust

コンベヤーに対する追従遅れ補正値を取得する動作を実施するかを設定します。

書式

Cnv_Adjust コンベヤー番号, On | Off

パラメーター

コンベヤー番号

コンベヤーの番号を式または数値 (1～16) で指定します。

On | Off

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

解説

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

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

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

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

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

注意

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

参照

[Cnv_AdjustGet関数](#), [Cnv_AdjustSet](#), [Cnv_AdjustClear](#), [Cnv_QueueGet関数](#)

Cnv_Adjust使用例

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

3.7.24 Cnv_AdjustClear

コンベヤーに対する追従遅れ補正値を消去します。

書式

Cnv_AdjustClear コンベヤー番号

パラメーター

コンベヤー番号

コンベヤーの番号を、式または数値 (1～16) で指定します。

解説

コンベヤートラッキングの追従遅れ補正取得動作の結果、補正量を消去し、"0"に設定します。

注意

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

参照

[Cnv_Adjust](#), [Cnv_AdjustSet](#), [Cnv_AdjustGet](#)関数, [Cnv_QueueGet](#)関数

Cnv_AdjustClear使用例

```
Cnv_AdjustClear 1
```

3.7.25 Cnv_AdjustGet関数

コンベヤーに対する追従遅れ補正値を返します。

書式

Cnv_AdjustGet(コンベヤー番号, モード番号)

パラメーター

コンベヤー番号

コンベヤーの番号を式または数値 (1～16) で指定します。

モード番号

- 0: 補正取得動作の結果
- 1: 補正量

戻り値

- モード番号0: 実数値0～2を返します。
 - 0: 補正値の取得動作は未実施
 - 1: 補正値の取得済み
 - 2: 補正値の取得失敗
- モード番号1: 実数値 (単位: mm) を返します。

解説

Cnv_AdjustおよびCnv_QueueGet関数を使用してコンベヤートラッキングの追従遅れ補正値を取得していない場合、モード番号0～2のいずれも戻り値は "0" を返します。

注意

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

参照

[Cnv_Adjust](#), [Cnv_AdjustSet](#), [Cnv_AdjustClear](#), [Cnv_QueueGet](#)関数

Cnv_AdjustGet関数使用例

```
Print Cnv_AdjustGet(1, 1)
```

3.7.26 Cnv_AdjustSet

コンベヤーに対する追従遅れ補正値を設定します。

書式

Cnv_AdjustSet コンベヤー番号, 補正量

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

補正量

補正量を実数値(単位: mm)で指定します。

解説

Cnv_AdjustSetを実施しない場合、補正量は前回設定した値が適用されます。

コントローラー電源オン後に1度も補正値を取得する動作を実施していない場合、補正量は初期値 "0"が設定されています。

Cnv_Modeコマンドのモード番号設定が1: 精度優先モードの場合のみ、Cnv_AdjustSetコマンドの設定は有効になります。

Cnv_AdjustSetは円形コンベヤーでは使用できません。

注意

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

参照

[Cnv_Adjust](#), [Cnv_AdjustGet関数](#), [Cnv_AdjustClear](#), [Cnv_QueueGet関数](#)

Cnv_AdjustSet使用例

```
Cnv_AdjustSet 1, 4.5
```

3.7.27 Cnv_Downstream

指定コンベヤーの下流限を設定します。

書式

Cnv_Downstream コンベヤー番号, 下流限

パラメーター

コンベヤー番号

コンベヤーの番号を式または数値(1~16)で指定します。

下流限

追従開始領域の下流側の境界線を設定します

解説

キャリブレーションウィザードで設定した下流限を変更できます。ただし、斜め下流限を使用している場合はCnv_Downstreamから下流限の値を変更することはできません。

注意

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

参照

[Cnv_Upstream](#)

Cnv_Downstream使用例

```
Cnv_Downstream 1,500
```


3.7.28 Cnv_Downstream関数

指定コンベヤーの下流限設定値を返します。

書式

Cnv_Downstream(コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を式または数値 (1～16) で指定します。

戻り値

- 直線コンベヤー: 実数値 (単位: mm) で返します。
- 円形コンベヤー: 実数値 (単位: deg) で返します。

注意

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

参照

[Cnv_Upstream](#)

Cnv_Downstream関数使用例

```
Print "Downstream limit: ", Cnv_Downstream(1)
```

3.7.29 Cnv_Fine

指定コンベヤーに対し、コンベヤーの追従完了判断範囲の設定、表示をします。

書式

Cnv_Fine コンベヤー番号 [, Fine設定値]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

Fine設定値

トラッキング完了を判断する距離を示す実数値 (単位: mm)で指定します。

値を0にすると、Cnv_Fineは使用されません。Fine設定値を省略すると現在のCnv_Fine 設定を表示します。

解説

コンベヤーの追従完了を判断し、次のコマンドを受付可能にするためのワークからの距離を指定します。"0"に設定するとCnv_Fineは使用されず、動作命令の完了で次のコマンドを受付可能となります。デフォルト値は"0"で、次のタイミングのとき、デフォルト値が自動的に設定されます。

- コンベヤー定義時
- コントローラー起動時

注意

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

参照

[Cnv_Fine関数](#)

Cnv_Fine使用例

```
Cnv_Fine 1, 5
```

3.7.30 Cnv_Fine関数

指定コンベヤーの追従完了判断範囲の設定を返します。

書式

Cnv_Fine(コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

Cnv_Fine設定を、実数値 (単位: mm)で返します。

注意

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

参照

[Cnv_Fine](#)

Cnv_Fine関数使用例

```
Real f  
f = Cnv_Fine(1)
```

3.7.31 Cnv_Flag関数

追従中止線に対する、追従状態を返します。

書式

Cnv_Flag (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

- 0: 正常に追従動作を実行しました。(キャンセル, 中止されていません)
- 1: ワークが追従中止線を超える見込みのため、追従動作実行をキャンセルしました。
- 2: ワークが追従中止線を超えたため、追従動作を中止しました。Z位置が指定の高さまで下がっていません。
- 3: ワークが追従中止線を超えたため、追従動作を中止しました。Z位置が指定の高さまで下がっています。
- 4: ワークが追従開始領域外のため、追従動作実行をキャンセルしました。

追従中止線を設定した場合のみ、0以外の戻り値を返します。

追従中止線の詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド"

注意

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

Cnv_Flag関数使用例

```
Print Cnv_Flag (1)
```

3.7.32 Cnv_LPulse関数

コンベヤーのトリガーによってラッチされたパルスを返します。

書式

Cnv_LPulse (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1~16)で指定します。

解説

ハードウェアトリガー配線またはCnv_Triggerによってラッチされた最新のコンベヤーのパルスを返します。

戻り値

指定したコンベヤーのラッチされたパルスをLong型数値で返します。

注意

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

参照

[Cnv_Trigger](#), [Cnv_Pulse関数](#)

Cnv_LPulse関数使用例

```
Print "Latched conveyor position: ", Cnv_LPulse(1)
```

3.7.33 Cnv_Mode

コンベヤートラッキングのトラッキングモードを設定します。

書式

Cnv_Mode (コンベヤー番号,モード番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

モード番号

- 0: 個数優先モード
- 1: 精度優先モード
- 2: 可変速コンベヤー対応モード

解説

コンベヤートラッキングのトラッキングモードを設定します。

Cnv_Modeは直線コンベヤーでのみ使用します。

トラッキング動作前にトラッキングモードを設定してください。設定がない場合、個数優先モードが設定されます。

- 個数優先モード: 精度はピッキング精度優先モードよりも劣りますが追従に要する時間が短いため、ワーク間隔の狭いコンベヤーシステムやコンベヤー速度の速いコンベヤーシステムに適したモードです。
- 精度優先モード: 追従に要する時間は個数優先モードよりも長くなりますが、精度が向上するため、小さめのワークを使うコンベヤーシステムに適したモードです。
- 可変速コンベヤー対応モード: ワークに接触して作業している最中に、コンベヤーを停止、または稼働させるようなコンベヤーシステムに適したモードです。

円形コンベヤーがサポートしているモード番号は、"0"のみです。"1"、または"2"を設定した場合は、ロボットはモード番号"0"と同じ動きをします。

注意

このコマンドは、コンベヤートラッキングオプションがインストールされている場合のみ使用できます。また、動作中にこのコマンドを使って、コンベヤートラッキングのモード変更はできません。

参照

[Cnv_Mode関数](#)

Cnv_Mode使用例

```
Cnv_Mode 1, 1
```

3.7.34 Cnv_Mode関数

コンベヤートラッキングのトラッキングモードを返します。

書式

Cnv_Mode (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

実数値0～2を返します。

- 0: 個数優先モード
- 1: 精度優先モード
- 2: 可変速コンベヤー対応モード

注意

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

参照

[Cnv_Mode](#)

Cnv_Mode関数使用例

```
Print Cnv_Mode (1)
```

3.7.35 Cnv_Name\$関数

指定したコンベヤーの名称を返します。

書式

Cnv_Name\$ (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値(1～16)で指定します。

戻り値

コンベヤー名を文字列で返します。

注意

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

参照

[Cnv_Number関数](#)

Cnv_Name\$関数使用例

```
Print "Conveyor 1 Name: ", Cnv_Name$(1)
```


3.7.36 Cnv_Number関数

指定したコンベヤー名称のコンベヤー番号を返します。

書式

Cnv_Number (コンベヤー名称)

パラメーター

コンベヤー名称

コンベヤー名称を文字列で指定します。

戻り値

コンベヤー番号を整数値で返します。

注意

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

参照

[Cnv_Name\\$関数](#)

Cnv_Number関数使用例

```
Integer cnvNum  
  
cnvNum = Cnv_Number("Main Conveyor")
```

3.7.37 Cnv_OffsetAngle

コンベヤーキューデータのオフセット値を設定します。

書式

Cnv_OffsetAngle コンベヤー番号 [, オフセット値]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

オフセット値

コンベヤーキューデータのオフセット値を実数値 (単位: degree)で指定します。省略可能です。省略すると、現在のオフセット値が表示されます。

解説

コンベヤーキューデータのオフセット値を設定します。

Cnv_OffsetAngleは円形コンベヤーでのみ使用します。

コンベヤートラッキングは使用するコンベヤー速度によっては追従遅れが発生することがあります。追従遅れが発生した場合、ロボットは追従遅れ量だけずれた位置をハンドリングします。Cnv_OffsetAngleはキューにオフセット値を与えることでこのずれを修正します。

注意

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

参照

[Cnv_OffsetAngle関数](#)

Cnv_OffsetAngle使用例

```
Cnv_OffsetAngle 1, 5
```

3.7.38 Cnv_OffsetAngle関数

コンベヤーキューデータのオフセット値を返します。

書式

Cnv_OffsetAngle (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

実数値(単位: degree)を返します。

注意

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

参照

[Cnv_OffsetAngle](#)

Cnv_OffsetAngle関数使用例

```
Real offsetAngle  
offsetAngle = Cnv_OffsetAngle (1)
```

3.7.39 Cnv_Point関数

センサー座標値をコンベヤー座標値に変換して返します。

書式

Cnv_Point (コンベヤー番号, センサーX座標値, センサーY座標値 [, センサーU座標値])

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16) で指定します。

センサーX座標値

センサーX座標値を実数値で指定します。

センサーY座標値

センサーY座標値を実数値で指定します。

センサーU座標値

センサーU座標値を実数値で指定します。省略可能です。

戻り値

コンベヤー座標値をポイントデータとして返します。

解説

Cnv_Point関数は、コンベヤーキューとして追加された座標値を、ポイントデータとして追加するために使います。ビジョンコンベヤーでは、センサーX座標値、センサーY座標値は、カメラからのビジョン座標値です。センサーコンベヤーでは、センサー位置がコンベヤー座標系の原点となります。

注意

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

参照

[Cnv_Speed関数](#)

Cnv_Point関数使用例

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
    VGet FindParts.Part.CameraXYU(i), found, x, y, u
    Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```

3.7.40 Cnv_PosErr関数

現在のトラッキング位置とターゲットとの位置偏差を返します。

書式

Cnv_PosErr (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

実数値 (単位: mm)を返します。

注意

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

参照

Cnv_PosErr関数使用例

```
Print "Conveyor 1 position error: ", Cnv_PosErr(1)
```

3.7.41 Cnv_PosErrOffset

現在のトラッキング位置とターゲットとの位置偏差を補正する値を設定します。

書式

Cnv_PosErrOffset コンベヤー番号, 補正值

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1~16)で指定します。

補正值

コンベヤー速度を予測する時間を実数値 (0~255, 単位: msec)で指定します。

解説

コンベヤートラッキング中にコンベヤーの停止と稼働を繰り返すと、コンベヤーの速度変化に対するロボットの追従遅れによってトラッキング位置とターゲットとの位置偏差が悪化します。

補正值で設定した時間後のコンベヤー速度を予測してコンベヤートラッキングすることで、位置偏差を改善できます。

Cnv_PosErrOffsetは可変速コンベヤー対応モードのみ使用可能です。個数優先モードと精度優先モードの場合は、補正值を設定しても位置偏差は改善できません。

注意

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

参照

[Cnv_Mode](#), [Cnv_PosErr](#)関数

Cnv_PosErrOffset使用例

```
Cnv_Mode 1, 2      ' 可変速コンベヤー対応モード
Cnv_PosErrOffset 1, 10  ' 補正值10msec
```

3.7.42 Cnv_Pulse関数

コンベヤーの現在位置のパルスを返します。

書式

Cnv_Pulse (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

指定したコンベヤーのパルスをLong型数値で返します。

注意

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

参照

[Cnv_Trigger](#), [Cnv_LPulse関数](#)

Cnv_Pulse関数使用例

```
Print "Current conveyor position: ", Cnv_Pulse(1)
```

3.7.43 Cnv_QueueAdd

コンベヤーキューデータにポイントデータを追加します。

書式

Cnv_QueueAdd コンベヤー番号, ポイントデータ [, ユーザーデータ]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16) で指定します。

ポイントデータ

コンベヤーキューデータに追加するポイントデータを指定します。

ユーザーデータ

ポイントデータと一緒に登録するユーザーデータを実数値で指定します。省略可能です。

解説

ポイントデータは指定したコンベヤーキューデータの最後尾に追加されます。現在ラッチされているコンベヤーやパルス位置も一緒に登録されます。

Cnv_QueueRejectで指定してある距離以下のキューデータがある場合、データは登録されませんが、エラーにはなりません。

キューデータの上限值は1000です。

注意

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

参照

[Cnv_RobotConveyor関数](#)

Cnv_QueueAdd使用例

```
Boolean found
Integer i, numFound
Real x, y, u

Cnv_Trigger 1
VRun FindParts
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
    VGet FindParts.Part.CameraXYU(i), found, x, y, u
    Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```


3.7.44 Cnv_QueueGet関数

指定したコンベヤーキューデータからポイントデータを返します。

書式

Cnv_QueueGet(コンベヤー番号 [, インデックス])

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

インデックス

取得するキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。)省略可能です。

戻り値

指定したコンベヤーキューからポイントデータを返します。

解説

Cnv_QueueGetは、コンベヤーキューからポイントデータを取得します。インデックスを省略した場合、キューデータの先頭データを返します。インデックスを設定した場合は、設定したインデックスのポイントデータを返します。

Cnv_QueueGetはコンベヤーキューからポイントデータを削除しません。削除するためにはCnv_QueueRemoveを使います。

コンベヤーを動作させながらワークをトラッキングしたい場合は、動作コマンドにCnv_QueueGetを入れる必要があります。

例:

```
Jump Cnv_QueueGet(1) ' ワークを追従
```

Cnv_QueueGetの戻り値をポイントに代入して、そのポイントにロボットを動作させ、ワークをトラッキングすることはできません。

```
P1 = Cnv_QueueGet(1)
Jump P1 ' ワークを追従しない
```

Cnv_QueueGetの戻り値をポイントに代入すると、その座標値は命令実行時のワーク位置になります。

注意

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

参照

[Cnv_QueueLen関数](#), [Cnv_QueueRemove](#)

Cnv_QueueGet関数使用例

```
' キューの先頭にJumpして追従
Jump Cnv_QueueGet(1)
On gripper
Wait .1
Jump place
```

```
Off gripper  
Wait .1  
Cnv_QueueRemove 1
```

3.7.45 Cnv_QueLen関数

指定したコンベヤーキューのデータ数を返します。

書式

Cnv_QueLen(コンベヤー番号 [, パラメーター番号])

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

パラメーター番号

キューデータをカウントするエリアを整数値で指定します。省略可能です。

定数	値	内容
CNV_QUELEN_ALL	0	キューデータの総数を返します。
CNV_QUELEN_UPSTREAM	1	追従開始領域より上流のキューデータ数を返します。
CNV_QUELEN_PICKUPAREA	2	追従開始領域内のキューデータ数を返します。
CNV_QUELEN_DOWNSTREAM	3	追従開始領域より下流のキューデータ数を返します。

戻り値

データ数を整数値で返します。

解説

有効なキューデータ数を返します。特に、追従開始領域内のデータ数を取得するのに有効です。

Wait命令の引数としても使うことが可能です。

注意

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

参照

[Cnv_QueGet関数](#)

Cnv_QueLen関数使用例

```
Do
  Do While Cnv_QueLen(1, CNV_QUELEN_DOWNSTREAM) > 0
    Cnv_QueRemove 1, 0
  Loop
  If Cnv_QueLen(1, CNV_QUELEN_PICKUPAREA) > 0 Then
    Jump Cnv_QueGet(1, 0) C0
    On gripper
    Wait .1
    Cnv_QueRemove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

3.7.46 Cnv_QueList

指定したコンベヤーキューデータの一覧を表示します。

書式

Cnv_QueList コンベヤー番号 [, 表示数]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

表示数

表示数データ数を整数値で指定します。省略可能です。省略すると、すべてのキューデータを表示します。

注意

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

参照

[Cnv_QueGet関数](#)

Cnv_QueList使用例

```
Cnv_QueList 1
```

3.7.47 Cnv_QueueMove

上流コンベヤーのキューデータを下流コンベヤーのキューに移動します。

書式

Cnv_QueueMove コンベヤー番号 [, インデックス] [, ユーザーデータ]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

インデックス

移動するキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。) 省略可能です。

ユーザーデータ

データと一緒に登録するユーザーデータを実数値で指定します。省略可能です。

解説

キューデータを下流コンベヤーのキューに移動します。インデックスを省略した場合、先頭 (インデックスが0)のキューデータを移動します。

注意

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

参照

[Cnv_QueueGet関数](#)

Cnv_QueueMove使用例

```
Cnv_QueueMove 1
```

3.7.48 Cnv_QueueReject

キューの2重登録防止のための最小距離を設定、表示します。

書式

Cnv_QueueReject コンベヤー番号 [, 2重登録防止距離]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

拒絶距離

2重登録防止距離の最小値を実数値 (単位: mm)で指定します。マイナス値を指定した場合は、0 mmが設定されます。省略可能です。省略すると、現在の2重登録防止距離が表示されます。

解説

キューの2重登録防止のための最小距離を設定します。ビジョンシステムによりスキャンされ、1回以上検出されていても登録されるのは1度です。Cnv_QueueRejectは2重登録防止のシステムフィルターとなります。デフォルトは0 mmです。

注意

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

参照

[Cnv_QueueReject関数](#)

Cnv_QueueReject使用例

```
Cnv_QueueReject 1, 20
```

3.7.49 Cnv_QueueReject関数

キューの2重登録防止距離を返します。

書式

Cnv_QueueReject (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

実数値 (単位: mm)を返します。

注意

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

参照

[Cnv_QueueReject](#)

Cnv_QueueReject関数使用例

```
Real rejectDist  
  
RejectDist = Cnv_QueueReject(1)
```

3.7.50 Cnv_QueueRemove

コンベヤーキューデータからキューデータを削除します。

書式

Cnv_QueueRemove コンベヤー番号 [, インデックス | All]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

インデックス

削除する先頭のインデックスを整数値、またはすべて削除する場合はAllで指定します。省略可能です。

解説

コンベヤーキューデータから1つ以上のキューデータを削除します。キューデータを使い終わったときに、キューデータを削除します。

注意

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

参照

[Cnv_QueueAdd](#)

Cnv_QueueRemove使用例

```
Jump Cnv_QueueGet(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1

' コンベヤーからデータを削除
Cnv_QueueRemove 1
```


3.7.51 Cnv_QueUserData

キューエントリに関連したユーザーデータの設定、表示をします。

書式

Cnv_QueUserData コンベヤー番号 [, インデックス] [, ユーザーデータ]

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16) で指定します。

インデックス

キューデータのインデックスを整数値で指定します。省略可能です。

ユーザーデータ

ユーザーデータを実数値で指定します。省略可能です。

解説

コンベヤーキューのユーザーデータを設定します。パラメーターのユーザーデータは省略可能ですが、通常のオペレーションでは必要です。

注意

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

参照

[Cnv_QueUserData関数](#)

Cnv_QueUserData使用例

```
Cnv_QueUserData 1, 1, angle
```

3.7.52 Cnv_QueueUserData関数

キューエントリに関連したユーザーデータを返します。

書式

Cnv_QueueUserData (コンベヤー番号 [, インデックス])

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

インデックス

キューデータのインデックスを整数値で指定します。省略可能です。

戻り値

実数値を返します。

注意

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

参照

[Cnv_QueueUserData](#)

Cnv_QueueUserData関数使用例

```
' キューに追加
Cnv_QueueAdd 1, Cnv_Point(1, x, y), angle

' キューから削除
angle = Cnv_QueueUserData(1) ' デフォルトのインデックス"0"になる
Jump Cnv_QueueGet(1) :U(angle)
Cnv_QueueRemove 1
```

3.7.53 Cnv_RobotConveyor関数

追従中のコンベヤー番号を返します。

書式

Cnv_RobotConveyor [(マニピュレーター番号)]

パラメーター

マニピュレーター番号

マニピュレーター番号を整数値で指定します。

戻り値

コンベヤー番号を整数値で返します。追従するコンベヤーがない場合は0を返します。

解説

複数のマニピュレーターを使用中の場合、マニピュレーターがどのコンベヤーを追従しているかを確認します。

注意

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

参照

Cnv_RobotConveyor関数使用例

```
Integer cnvNum
```

```
cnvNum = Cnv_RobotConveyor(1)
```

3.7.54 Cnv_Speed関数

コンベヤーの動作速度を返します。

書式

Cnv_Speed (コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

実数値 (単位: mm/s)を返します。円形コンベヤーでは、実数値 (単位: degree/s)を返します。

注意

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

参照

[Cnv_Pulse関数](#)

Cnv_Speed関数使用例

```
Print "Conveyor speed: ", Cnv_Speed(1)
```

3.7.55 Cnv_Trigger

次のCnv_QueueAddステートメントのために、コンベヤーの現在位置をラッチします。

書式

Cnv_Trigger コンベヤー番号

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

解説

指定したコンベヤーエンコーダーのパルス出力基板に、ハードウェアトリガーが配線されていない場合に使用します。Cnv_Triggerはソフトウェアトリガーとなります。

なお、VRunコマンドの処理時間にバラツキが発生する場合がありますので、VRunコマンド実施後に、Cnv_Triggerコマンドの使用をお勧めします。

注意

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

参照

[Cnv_QueueAdd](#)

Cnv_Trigger使用例

```
Boolean found
Integer i, numFound
Real x, y, u

VRun FindParts
Cnv_Trigger 1
VGet FindParts.Part.NumberFound, numFound
For i = 1 To numFound
    VGet FindParts.Part.CameraXYU(i), found, x, y, u
    Cnv_QueueAdd 1, Cnv_Point(1, x, y)
Next i
```

3.7.56 Cnv_Upstream

指定コンベヤーの上流限を設定します。

書式

Cnv_Upstream コンベヤー番号, 上流限

パラメーター

コンベヤー番号

コンベヤーの番号を式または数値 (1~16) で指定します。

上流限

追従開始領域の上流側の境界線を設定します。

解説

キャリブレーションウィザードで設定した上流限を変更できます。ただし、斜め上流限を使用している場合は Cnv_Upstream から上流限の値を変更することはできません。

注意

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

参照

[Cnv_Downstream](#)

Cnv_Upstream 使用例

```
Cnv_Upstream 1,200
```

3.7.57 Cnv_Upstream関数

指定コンベヤーの上流限設定値を返します。

書式

Cnv_Upstream(コンベヤー番号)

パラメーター

コンベヤー番号

コンベヤーの番号を整数値 (1～16)で指定します。

戻り値

- 直線コンベヤー: 実数値 (単位: mm)で返します。
- 円形コンベヤー: 実数値 (単位: deg)で返します。

注意

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

参照

[Cnv_Downstream](#)

Cnv_Upstream関数使用例

```
Print "Upstream limit: ", Cnv_Upstream(1)
```

3.7.58 CollisionDetect

現在のロボットの衝突検出機能(ロボット動作の異常の検出機能)をオン、またはオフします。

書式

(1) CollisionDetect 状態

(2) CollisionDetect 状態, 関節番号

(3) CollisionDetect

パラメーター

状態

- On: 衝突検出(ロボット動作の異常の検出)を有効にします。
- Off: 衝突検出(ロボット動作の異常の検出)を無効にします。

関節番号

スカラロボット (RSシリーズ含む)の場合、1から4まで、垂直6軸型ロボット (Nシリーズ含む)の場合、1から6までの関節番号を指定します。

結果

パラメーターが省略されているときは、現在のCollisionDetect状態を表示します。

解説

ロボットを動かしたい速度と、実際に動いている速度の差(速度偏差値)により、ロボット動作の異常を検出します。本機能で検出できる異常は、AとBに分類されます。

- A: ロボットアームやハンドの衝突や接触の発生
- B: 衝突や接触以外のロボット動作での異常

さらにBの異常は、パワーの状態により、以下のように分類されます。

- ハイパワー状態での異常: WeightやInertiaの過小設定によるトルク飽和
 - 多関節軸の複合動作や長尺物を振り回すことによるトルク飽和
 - 電源電圧低下によるトルク飽和
 - ハードウェアの異常やソフトウェアの誤作動による異常動作
- ローパワー時の異常:
 - ハードウェアの異常やソフトウェアの誤動作による異常動作
 - 仕様を超えたハンド質量や長尺物の保持によるローパワー時トルク飽和

衝突検出機能は、EPSON RC+ 7.0 Ver.7.2以降でサポートされる汎用ロボット (垂直6軸型ロボット, スカラロボット)に対応しています。X5シリーズなど、未サポートのロボットを接続した状態で本コマンドを使用すると、エラーが発生します。

本コマンドの実行は処理時間を要します。サイクルタイムが要求される場合は、コマンドの使用を最小限にしてください。

コマンドにより、全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オンです。(ファームウェアバージョンVer7.2.1.x以降の場合はオン、Ver7.2.0.x以前の場合はオフ)

コントローラーの電源をオフするとデフォルト値に戻りますが、それ以外の場合は、本コマンドで明示的に設定しない限り、状態は変化しません。

衝突検出時には、以下のメッセージを出力し、ロボットを停止します。

- エラー5057 "ハイパワー状態で衝突を検知(ロボット動作の異常の検出)"

- エラー5058 "ローパワー状態で衝突を検知(ロボット動作の異常の検出)"

ハイパワーモード時のダメージ低減には、LimitTorqueコマンドによるトルク制限機能との併用、ローパワーモード時のダメージ低減には、LimitTorqueLPコマンドによるトルク制限機能との併用も効果的です。

以下の説明も参照してください。

"Epson RC+ ユーザーズガイド - 衝突検出機能(ロボット動作の異常の検出機能)"

参照

[LimitTorque](#), [LimitTorque関数](#), [LimitTorqueLP](#), [LimitTorqueLP関数](#)

CollisionDetect使用例

CollisionDetect On	' 全軸衝突検出をONとする
CollisionDetect Off, 5	' 第5関節のみOFFとする
CollisionDetect	' on, on, on, on, off, on と表示されます。

3.7.59 CollisionDetect関数

CollisionDetect命令の設定値を返します。

書式

CollisionDetect(関節番号)

パラメーター

関節番号

1から6の整数で指定します。

戻り値

CollisionDetect命令の設定値を整数値で返します。

- 1 = ON
- 0 = OFF

参照

[CollisionDetect](#)

CollisionDetect関数使用例

```
Print CollisionDetect(1) '第1関節のCollisionDetect値を表示
```

3.7.60 Cont

一時停止状態となったコントローラーを再開し、全タスクを継続実行する場合に使用します。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

Cont

解説

本命令をプログラムから実行するには、Epson RC+ の[セットアップ] - [システム設定] - [コントローラー] - [環境設定]にある[アドバンスタスク制御コマンドを有効]チェックボックスをチェックする必要があります。ただし、この設定を行ってもTrap SGCcloseで起動されるタスクからCont命令を実行することはできません。

Contコマンドは、Pauseステートメントの実行や、安全扉開入力により一時停止状態となったコントローラーを再開し、全タスクを継続実行する場合に使用します。[オペレーターウィンドウ]の[継続実行]ボタンやリモート入力の"Continue"と同様の機能を持ちます。

WaitRecover状態 (安全扉開後の復帰待ち状態)でContコマンドを実行した場合、すべてのロボットが励磁と復帰動作を行ったのち、プログラムの実行が再開します。

ロボットの励磁と復帰動作のみを行いたい場合はRecoverコマンドを使用してください。

注意

プログラムからContコマンドを実行する場合、コマンドの仕様を理解して、システムとして継続実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

[Pause](#), [Recover](#), [ContManualRecover](#)

Cont使用例

```
Function main
  Xqt 2, monitor, NoPause
  Do
    Jump P1
    Jump P2
  Loop
Fend

Function monitor
  Do
    If Sw(pswitch) = On then
      Pause
      Wait Sw(pswitch) = Off and Sw(cswitch) = On
      Cont
    EndIf
  Loop
Fend
```

3.7.61 ContManualRecover

安全扉閉後、Recover命令による復帰動作が完了した状態で、全タスクを継続実行する場合に使用します。本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。本命令で、継続実行を行う場合は、安全扉が開いた時点の位置への復帰動作（Recover命令）を実行しておく必要があります。

書式

ContManualRecover

解説

本命令をプログラムから実行するには、EPSON RC+ の[セットアップ]－[システム設定]－[コントローラー]－[環境設定]にある[アドバンスタスク制御コマンドを有効]チェックボックスをチェックする必要があります。ただし、この設定を行ってもTrap SGCcloseで起動されるタスクからContManualRecover命令を実行することはできません。

ContManualRecoverコマンドは、安全扉開入力により一時停止状態となったコントローラーがRecover命令により再開され、安全扉が開いた時点の位置へロボットが復帰動作を完了している場合に使用します。リモート入力の“ContinueManualRecover”と同様の機能を持ちます。

WaitRecover状態（安全扉閉後の復帰待ち状態）でContManualRecoverコマンドを実行した場合、Cont命令とは異なり、エラーとなります。

事前にRecoverコマンドにより、ロボットの励磁と復帰動作を行ってから使用してください。

注意

プログラムからContManualRecoverコマンドを実行する場合、コマンドの仕様を理解して、システムとして継続実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

[Pause](#), [Recover](#), [Cont](#), [Recover関数](#)

ContManualRecover使用例

```
Function main
  Xqt 2, monitor, NoPause
  Do
    Jump P1
    Jump P2
  Loop
Fend

Function monitor
  Do
    Wait SafetyOn = True      `安全扉のオープンを待つ
    Wait SafetyOn = False    `安全扉のクローズを待つ
    If (CtrlInfo(1) and &H2000) = &H2000 Then
      Recover
      ContManualRecover
    EndIf
  Loop
Fend
```

3.7.62 Copy

ファイルをコピーします。

書式

Copy コピー元, コピー先

パラメーター

コピー元

コピーするファイルのパスとファイル名を指定します。詳細は、ChDiskを参照してください。

コピー先

ソースファイルをコピーする先のパスと、コピーするファイル名を指定します。詳細は、ChDiskを参照してください。

解説

指定したソースファイルを、指定したコピー先ファイル名にコピーします。

コピー元とコピー先で同じパス名またはファイル名を共有することはできません。

コピー先がすでに存在する場合は、エラーになります。

注意

ネットワークパスが使用できます。

ファイル名には "*", "?"などのワイルドカードは使えません。

コマンドウィンドウで使った場合、引用符やカンマは省略することができます。

参照

[ChDir](#), [MkDir](#)

Copy使用例

コマンドウィンドウでの操作例

```
> copy TEST.DAT TEST2.DAT

> Copy TEST.DAT c:      'NG
!!エラー: 7203 アクセスが拒否されました。
> Copy TEST.DAT c:\      'OK
>
```

3.7.63 Cos関数

指定角度のコサインを返す関数です。

書式

Cos(数値)

パラメーター

数値

角度を数値 (ラジアン) で指定します。

戻り値

指定した数値のコサイン値 (実数値) を返します。

解説

Cosは、指定した角度 (ラジアン) のコサインを返します。指定する角度 (数値) は、ラジアンで指定してください。戻り値の範囲は、-1～1の間になります。

角度が度で与えられている場合は、DegToRad関数を使ってラジアンに変換しなければなりません。

参照

[Abs関数](#), [Atan関数](#), [Atan2関数](#), [Int関数](#), [Mod 演算子](#), [Not 演算子](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#)

Cos関数使用例

下記は、Cosを使った簡単なプログラム例です。

```
Function costest
  Real x
  Print "Please enter a value in radians"
  Input x
  Print "COS of ", x, " is ", Cos(x)
Fend
```

下記は、Cosをコマンドウィンドウから使った例です。

```
Display the cosine of 0.55:

>print cos(0.55)
0.852524522059506
>

Display cosine of 30 degrees:

>print cos(DegToRad(30))
0.866025403784439
>
```

3.7.64 CP

パスモーションを設定します。

書式

- (1) CP { On | Off }
- (2) 動作命令 目標座標 CP

パラメーター

On | Off

- On: パスモーションを有効にします。
- Off: パスモーションを解除します。

解説

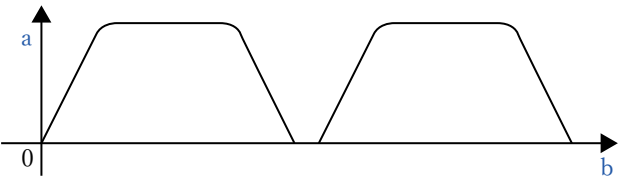
パスモーションは下記の命令で使います。

Arc, Arc3, Go, Jump, Jump3, Jump3CP, JumpTLZ, Move

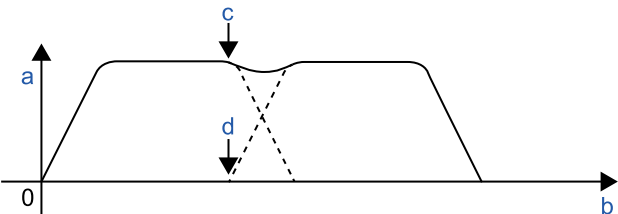
CP Onすると、各動作命令でCPパラメーターを指定するかしないかにかかわらず、動作命令は減速開始と同時に次のステートメントが実行されます。この結果、減速動作中に次の動作を開始すると動作軌道が合成されます。

CP Offすると、各動作命令でCPパラメーターを指定した場合にのみ、この機能は有効になります。

通常動作



パスモーション



記号	説明
a	速度
b	時間
c	減速開始
d	加速開始

CP動作 (Arc, Arc3, Jump3, Jump3CP, JumpTLZ, Move)同士、またはPTP動作 (Go, Jump)同士は、CP Onにより動作軌道が合成されます。
しかし、CP動作とPTP動作の連続軌道の場合は動作軌道が合成されずに、一度減速します。
コントローラーの環境設定から、CP On時CP動作とPTP動作を接続する場合、CP動作とPTP動作の動作軌道が合

成されるようになります。動作加減速度の設定によっては、過速度エラーや過加速度エラーが発生することがあります。その場合は、加減速度を調整するか、無効にしてください。

また、垂直6軸型ロボット (Nシリーズ含む)での手首特異点を目標値とするCP動作は、次動作と動作軌道が合成されず、一度減速します。

パスモーション有効時にHereを使用する場合、Hereを指定した動作の前で動作軌道が合成されずに、減速します。下記のような使い方をした場合、P1で減速して停止したあとに次の動作を開始します。

```
Go P1 CP
Go Here+X(100)
```

停止させずに動作させたい場合は、Hereの位置を事前に計算するようにプログラムを修正してください。

```
Go P1 CP
Go P1+X(100)
```

下記の場合はCP Offになります。

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

参照

CP関数, [Arc](#), [Arc3](#), [Go](#), [Jump](#), [Jump3](#), [Jump3CP](#), [JumpTLZ](#), [Move](#)

CP使用例

```
CP On
Move P1
Move P2
CP Off
```


3.7.65 CP関数

パスモーションの状態を返します。

書式

CP

戻り値

- 0 = パスモーションオフ
- 1 = パスモーションオン

参照

[CP](#)

CP関数使用例

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

3.7.66 CP_Offset

CP On時に、後続する動作命令を開始するオフセット時間を設定します。

書式

(1) CP_Offset [On [, OffsetTime]]

(2) CP_Offset Off

パラメーター

On | Off

- On: CP On時の動作開始オフセット機能を有効にします。省略した場合は、現在の設定を表示します。
- Off: CP On時の動作開始オフセット機能を無効にします。

OffsetTime

CP On時の動作開始オフセット時間を10～24 (単位: ms)の範囲の実数値で設定します。省略した場合、デフォルト値10 msが設定されます。

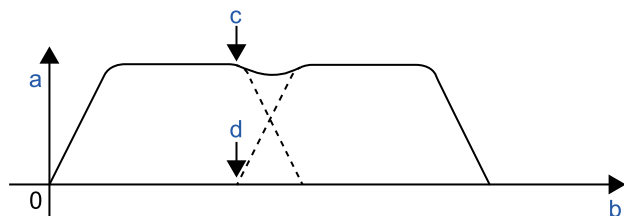
解説

CP_Offsetは以下の動作命令が対象となります。

Move, Arc, Arc3, CVMove

CP Onや動作命令にCPパラメーターを付加すると、前の動作の減速開始と同時に次のステートメントが実行されます。この結果、下図のように減速区間と次の動作の加速区間が重なるパスモーションになります。この時、前の動作の減速開始と次の動作の加速開始は、ステートメントの開始処理にかかわるオーバーヘッド時間があるため、厳密に一致しません。そのため、パスモーションのつなぎ目付近では速度が落ち込み、等速軌道になりません。これを改善するため、後続する動作ステートメント開始時間を前倒しするのがCP_Offsetの機能です。

パスモーション



記号	説明
a	速度
b	時間
c	減速開始
d	加速開始

CP_Offsetをオンにすると、OffsetTimeパラメーターに設定された時間分、後続する動作命令の開始処理が早められ、実際のロボットの減速開始と次の動作の加速開始が同期し、パスモーションの等速性能が改善します。

OffsetTimeパラメーターにはデフォルト値が設定されていますが、アプリケーションによって微調整してください。特に、後続する動作命令が"!並列処理!"を含む場合、動作開始にかかるオーバーヘッド時間が長くなるため、OffsetTimeをデフォルト値より大きな値 (16ms程度) に設定します。

CP_Offsetの設定時間 (OffsetTime)を調整するために、調整したい動作を実行中にTCPSpeedを用いてツール先端速度を観測します。適切なOffsetTimeが設定されると、動作のつなぎ目での速度が一定になります。OffsetTimeが大きすぎる場合TCPSpeedが上昇し、OffsetTimeが小さすぎる場合は、TCPSpeedが落ち込みます。CP_Offsetの調整は実機環境で行ってください。実コントローラーとシミュレーターでは動作開始の処理時間が異なるため、適切な調整ができません。

TCPSpeed計測プログラム例

```
Function main
  Motor On
  Power High

  SpeedS 250; AccelS 1500
  Speed 50; Accel 50, 50

  Go XY(300, 500, 500, 90, 0, 180)

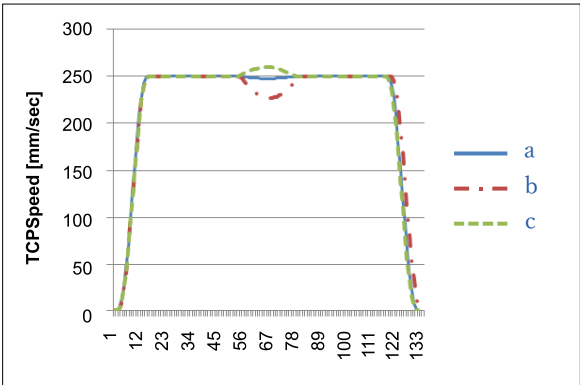
  CP_Offset On
  Xqt printTcPSpeed

  Move XY(0, 500, 500, 90, 0, 180) CP
  Move XY(-300, 500, 500, 90, 0, 180)

  Quit printTcPSpeed
  CP_Offset Off
Fend

Function printTcPSpeed
  Do
    Print TCPSpeed
  Loop
Fend
```

OffsetTime調整例



記号	説明
a	適切なOffsetTime
b	OffsetTime=0
c	OffsetTime過大

本命令はPTP動作を対象としません。PTP動作の場合は、通常通りのパスモーションとなります。

下記の場合はCP_Offsetはオフになります。

- コントローラー起動時

- Motor On実行
- SFree, SLock, Brake 実行
- Reset, Reset Error実行
- 停止ボタン, Quit All実行などによるタスクの終了

参照

[CP_Offset関数](#), [CP](#), [Move](#), [Arc](#), [Arc3](#), [CVMove](#)

CP_Offset使用例

```
CP_Offset On
Move P1
Move P2
CP_Offset Off
```

3.7.67 CP_Offset関数

CP On時に、後続する動作命令を開始するオフセット時間を返します。

書式

CP_Offset

戻り値

動作の開始オフセット時間を表す実数

参照

[CP_Offset](#)

CP_Offset関数使用例

```
If CP_Offset = 0 Then  
    Print "CP_Offset is off"  
EndIf
```

3.7.68 Ctr関数

カウンターのカウント値を返す関数です。

書式

Ctr(入力ビット番号)

パラメーター

入力ビット番号

カウンターに定義されている入力ビットの番号です。同時に有効にできるカウンターの最大数は、16個までです。

戻り値

カウンターの現在カウント値 (0～65535の整数)

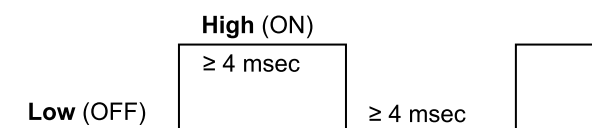
解説

Ctrは、CTResetステートメントと一緒に使うことで、I/O入力をカウントとして使えるように設定します。

カウンターに設定されたI/O入力が入力オフからオンに変わるたびに、その入力は、カウンター上のカウントを1つずつ増加させます。

Ctr関数は、指定するカウンター入力の現在のカウンター値を得るために使えます。どのI/O入力でもカウンターに指定することができます。しかし、同時に有効にできるカウンターの数は、16個までです。

カウンターパルス入力のタイミングチャート



参照

[CTReset](#)

Ctr関数使用例

下記は、I/O入力のカウンターの値を得るプログラムの一例です。

```
CTReset 3 '入力3のカウンターを0にリセット
On 0      '出力スイッチをオン
Wait Ctr(3) >= 5
Off 0     '入力3の入力サイクルが5になった場合はスイッチをオフ (出力0オフ)
```

3.7.69 CTRreset

指定した入力カウンターの値をリセットします。また、入力をカウンター入力と設定します。

書式

CTRreset(入力ビット番号)

パラメーター

入力ビット番号

カウンターとして設定される入力ビットの番号です。有効な入力ビットの番号を整数で入力します。使用可能なカウンターの数は、16までです。

解説

CTRresetは、Ctr関数と一緒に使い、入力をカウンターとして使えるようにします。CTRresetは、指定された入力ビットをカウンターとして設定し、そのカウンターをスタートします。もし、指定された入力ですでにカウンターに設定されている場合は、リセットして再スタートします。

注意

- 電源をオフした場合のカウンター

電源をオフするとすべてのカウンター値は解除されます。

- Ctr関数を使う場合

Ctr関数を使って、現在のI/O入力のカウンター値を知ることができます。

参照

[Ctr関数](#)

CTRreset使用例

下記は、I/Oの入力カウンター値を得るプログラムの一例です。

```
CTRreset 3    '入力3のカウンターを0にリセット
On 0          '出力スイッチをオン
Wait Ctr(3) >= 5
Off 0         '入力3の入力サイクルが5になった場合はスイッチをオフ（出力0オフ）
```

3.7.70 CtrlDev関数

現在の制御デバイスの番号を返します。

書式

CtrlDev

戻り値

- 21: PC
- 22: リモートI/O
- 26: リモートイーサネット
- 20: TP4
- 19: リアルタイム動作指令

参照

[CtrlInfo関数](#)

CtrlDev関数使用例

```
Print "The current control device is: ", CtrlDev
```


3.7.71 CtrlInfo関数

コントローラーの情報を返します。

書式

CtrlInfo (インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値で指定します。

解説

CtrlInfo関数で取得可能な情報は次の表のとおりです。

インデックス	ビット	値	解説
0	N/A		旧互換のために残されています。 コントローラーのファームウェアバージョンを取得するにはインデックス9を使用してください。
1	コントローラー状態		
	0	&H1	Ready状態
	1	&H2	Start状態
	2	&H4	Pause状態
	3-7		未定義
	8	&H100	非常停止状態
	9	&H200	安全扉開状態
	10	&H400	エラー状態
	11	&H800	重大エラー状態
	12	&H1000	ワーニング状態
	13	&H2000	WaitRecover状態（安全扉開からの復帰待ち）
	14	&H4000	Recover 状態（安全扉開からの復帰動作中）
	15-31		未定義
2	0	&H1	TPのEnableスイッチがオン
	1-31		未定義
3	0	&H1	TEACHモード回路異常検出
	1	&H2	安全扉回路異常検出

インデックス	ビット	値	解説
	2	&H4	非常停止回路異常検出
	3-31		未定義
4	N/A		0 - 実ランモード 1 - ドライランモード
5	N/A		制御デバイス: 21 - RC+ 22 - リモート 26 - リモートイーサネット 29 - リモートRS232C 20 - TP4 19 - リアルタイム動作指令
6	N/A		設定されているロボットの台数
7	N/A		操作モード: 0 - Programingモード 1 - AUTO モード
8	N/A		未定義
9	N/A		コントローラーのファームウェアバージョン メジャー番号*1000000 + マイナー番号*10000 + レビジョン番号*100 + ビルド番号 例 1.6.2.4 の場合: 1060204
10	N/A		ハードディスクのSMARTステータス 0 : SMARTステータス正常 1 : SMARTステータス異常 SMARTステータスが異常の場合、ハードディスクが故障する可能性があるので、すみやかにデータのバックアップを行い、新しいハードディスクに交換してください。 RAIDオプション使用時は、SMARTステータスは使用できません。常に正常が返ります。
15	N/A		DC電源電圧値 入力電圧の100倍した値が得られます。 例 48.01Vの場合、4801として取得されます。 DC電源に対応していない機種では、エラーが返ります。
16	N/A		PLCベンダータイプ 0: None 1: Allen Bradley 2: CODESYS

戻り値

必要なLong整数式の値を返します。

参照

[RobotInfo関数](#), [TaskInfo関数](#)

CtrlInfo関数使用例

```
Print "The number of robots is: ", CtrlInfo(6)
```

3.7.72 CtrlPref関数

Epson RC+のシステム設定のコントローラ環境設定情報を取得します。

書式

CtrlPref (インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値で指定します。

インデックス	解説
0	Resetで出力ポートをOFF
1	非常停止で出力ポートをOFF
2	関節が非励磁状態での移動命令を許可
3	I/O出力命令でWalkの実行を停止
4	ドライラン
5	仮想I/Oモード
6	コントローラ状態保存時にプロジェクトも保存
7	安全扉開で全タスクを中断
8	安全扉開放時の位置へ自動復帰
9	Windowsの起動を待たずにスタート
10	メインファンクション (MainXX)起動時にグローバル変数を初期化
11	バックグラウンドタスクを有効
12	アドバンスタスク制御コマンドを有効
13	CP On時にCP動作とPTP動作を接続
14	コントローラ起動時に自動LJMを有効
15	TEACHモード時にLJMを無効
16	ポイントフラグのチェックを無効
17	Teachモード時イネーブルスイッチオフでMotor Off
18	部品消耗管理を有効
19	ForcePowerLow信号OFFでLowパワー
20	ForcePowerLow信号変化時、タスクを一時停止

インデックス	解説
21	T2モードの実行を禁止
22	Ethernet接続認証を無効
23	Epson RC+ Express Editionの接続を禁止
24	動作軌跡とパルス動作にXYLimを適用
25	RC+ API用タスク数

戻り値

インデックス = 25: RC+API用のタスク数を表す整数値

それ以外: 0 = 無効, 1 = 有効

解説

Epson RC+のセットアップメニューの [システム設定] - [コントローラー] - [環境設定] の設定値を取得したい場合、本コマンドを使用します。

参照

[CtrlInfo関数](#)

CtrlPref関数使用例

```
Integer i
For i = 0 To 15
Print "i = " + Str$(i) + ": " + Str$(CtrlPref(i))
Next
```

3.7.73 CurDir\$関数

現在のカレントディレクトリーを文字列で返します。

書式

CurDir\$

戻り値

ドライブ名とパス名の文字列を返します。

参照

[ChDir](#), [CurDrive\\$関数](#), [CurDisk\\$関数](#)

CurDir\$関数使用例

```
Print "The current directory is: ", CurDir$
```

3.7.74 CurDisk\$関数

現在のディスク名を文字列で返します。

書式

CurDisk\$

戻り値

ディスク名の文字列を返します。

参照

[ChDisk](#), [CurDir\\$関数](#), [CurDrive\\$関数](#)

CurDisk\$関数使用例

```
Print "The current disk is: ", CurDisk$
```

3.7.75 CurDrive\$関数

現在のドライブ名を文字列で返します。

書式

CurDrive\$

戻り値

ドライブ名の文字列を返します。

参照

[ChDrive](#), [CurDir\\$関数](#), [CurDisk\\$関数](#)

CurDrive\$関数使用例

```
Print "The current drive is: ", CurDrive$
```


3.7.76 CurPos関数

ロボットの現在の動作目標位置を返します。

書式

CurPos

戻り値

ロボットの現在の動作目標位置を返します。

参照

[InPos関数](#), [FindPos関数](#), [RealPos関数](#)

CurPos関数使用例

```
Function main
    Xqt showPosition
    Do
        Jump P0
        Jump P1
    Loop
Fend

Function showPosition
    Do
        P99 = CurPos
        Print CX(P99), CY(P99)
    Loop
Fend
```

3.7.77 Curve

CVMove命令による自由曲線CP動作で使用する動作軌跡のファイルを作成します。与えられたポイントをすべて通るような曲線が動作軌跡となります。

書式

Curve ファイル名,動作曲線の開閉,モード指定,座標軸数 [,補間方式],連続ポイントデータ指定

パラメーター

ファイル名

ポイントデータが保存されるファイル名を文字列で指定します。拡張子は".CVT"固定です。拡張子を省略した場合、.CVTの拡張子が自動的に追加されます。Curve命令が実行されると、ファイルが作成されます。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

動作曲線の開閉

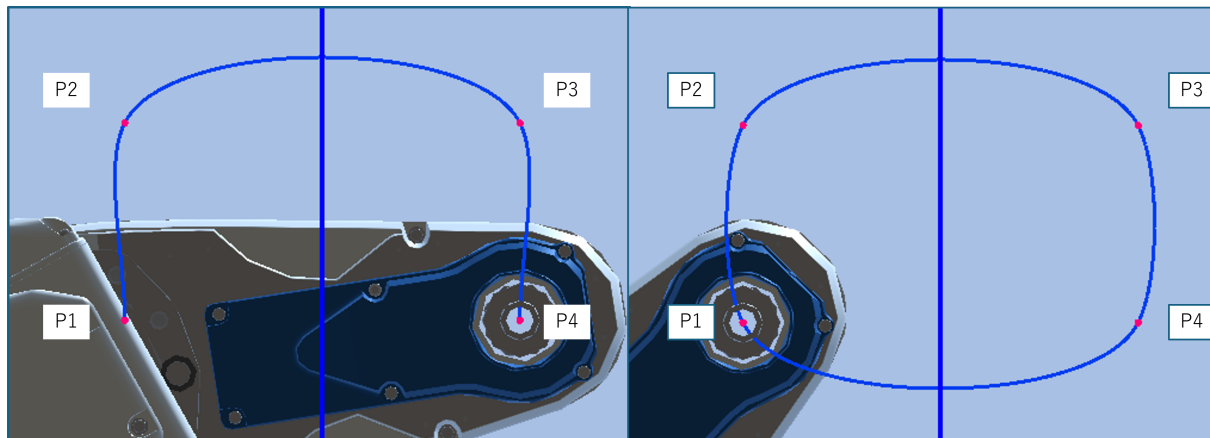
曲線動作の終了時における動作曲線の開閉を指定します。このパラメーターは、下記のいずれかの文字を指定します。

- C -生成する曲線が閉曲線
- O -生成する曲線が開曲線

開曲線を指定すると、Curve命令は、連続ポイントデータの最後のポイントでアームが停止します。閉曲線を指定すると、Curve命令は、最終ポイントを通過後も動作を継続し、アームを連続ポイントデータの始点に戻して動作を停止します。

閉曲線を指定すると、開始ポイントと最終ポイントの接線方向が一致するようになります。

下の図はP1～P4を入力とした場合の閉曲線と開曲線を比較したものです。閉区間の場合、P1まで動作します。また、動作の開始と終了が滑らかにつながります。



モード指定

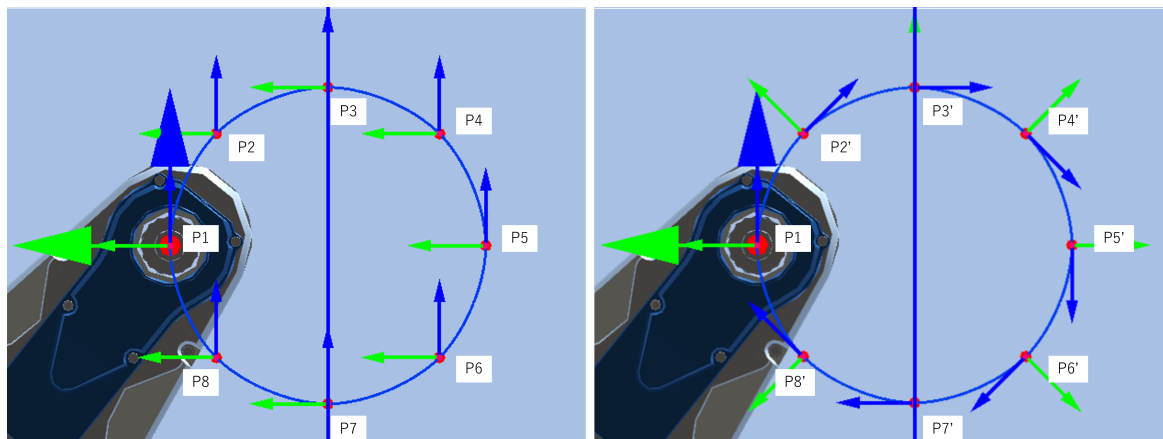
0～3ビット目を用いて姿勢補正するかどうか（ツールの姿勢を曲線の接線方向に自動的に補間するかどうか）を指定します。4～7ビット目でECP番号が指定できます。

モード指定		姿勢補正	ECP番号
16進数	10進数		
&H00	0	しない	0
&H10	16		1
&H20	32		2
...	...		...
&HA0	160		10
&HB0	176		11
&HC0	192		12
&HD0	208		13
&HE0	224		14
&HF0	240		15
&H02	2	する	0
&H12	18		1
&H22	34		2
...	...		...
&HA2	162		10
&HB2	178		11
&HC2	194		12
&HD2	210		13
&HE2	226		14
&HF2	242		15

姿勢補正を指定する場合は、アームは、連続ポイントデータの始点の姿勢のみを使います。途中のポイントのUVW座標は無視されます。姿勢補正を有効にした場合は、自由曲線の接線方向から見た時に、ツールの姿勢が常に一定になるように姿勢を維持します。これは、カッターのように、継続して接線方向への制御が必要なツールのときに指定します。

ワークの外周をなぞるようなポイントデータに対して、閉曲線と姿勢補正を同時に選択した場合、最終ポイントでは初期姿勢と同一姿勢になりますが、J6Flagまたはボールねじの回転数が異なります。

下の図は水平多関節型ロボットでの例になります。閉曲線かつ姿勢補正なしでP1～P8を指定した場合、P1～P8までを通過します。一方で、閉曲線と姿勢補正を同時に選択した場合、曲線の接線方向に従ってツール姿勢が補正されます。それにより、ツールはP1,P2'～P8'を通過します。P2～P8で指定した姿勢は無視されます。閉曲線のため、自由曲線の開始位置までツールは動作しますが、CVMove動作前後でボールねじがU軸に対して1回転した状態になります。



左:姿勢補正無し/右:姿勢補正有り

ECPを使用する場合は、4～7ビット目にECP番号を指定してください。

ECPは、ロボットでワークを把持し、ロボット周辺に固定されたツールを用いてワークの稜線に沿って軌跡制御させたい場合に有効です。外部制御点(ECP)座標系(オプション)についての基本的な説明は以下を参照ください。

"Epson RC+ ユーザーズガイド - SPEL+言語 - 座標系 - 外部制御点(ECP)座標系(オプション)"

ポイントデータに含まれる付加軸の位置を考慮した自由曲線を生成する場合、8ビット目を"1"に指定してください。例えば、姿勢補正およびECPを使用せず、付加軸位置を考慮した自由曲線を生成する場合には、"&H100"を指定します。

付加軸で自由曲線を生成する場合は、ロボット座標系とは関係なくS関節、T関節それぞれ独立に連続ポイントデータをつなぎます。

ただし、付加軸がPG軸で構成されている場合は、連続ポイントによる自由曲線は生成されず、最終ポイントに動作するデータが作成されます。

モード指定のビットと機能の対応は以下の通りです。

ビット	8	7	6	5	4	3	2	1	0
データ	付加軸	ECP番号			姿勢補正				

座標軸数

曲線動作中に制御される座標軸数を2, 3, 4, 6の整数で指定します。

- 2 - 姿勢を含まないXY平面で自由曲線を生成します。(垂直6軸型 (Nシリーズを含む)以外)
- 3 - 姿勢を含まないXYZ空間で自由曲線を生成します。(垂直6軸型 (Nシリーズを含む)以外)
- 4 - 姿勢を含むXYZ空間で自由曲線を生成します。(垂直6軸型 (Nシリーズを含む)以外)
- 6 - 姿勢を含むXYZ空間で自由曲線を生成します。(垂直6軸型 (Nシリーズを含む)のみ)

制御対象に選択されていない軸は、前回のエンコーダーパルス位置を保持し、Curve動作中は動作しません。例えば、水平多関節型ロボットで座標軸数に2を指定してカーブファイルを作成した場合、指定したポイントのZ座標によらず、CVMove動作中にZ軸は動作しません。

補間方式

曲線の補間方式を指定します。IM_CMPTBLを指定した場合、Ver8.1.1より前のファームウェアと互換性のある補間方式となります。IM_NORMALを指定した場合、新しい補間方式となります。省略可能で、省略した場合IM_CMPTBLで実行されます。

新しい補間方式では下記のようなメリットがあります。

- カーブファイル作成時間が短くなります
- 動作速度の調整機能により、姿勢変化の大きい動作でもエラーが発生しづらくなります。
- 直線区間を作成しやすくなります
- 姿勢補間がなめらかになります

通常はIM_NORMALを指定してください。補間方式の違いについては解説を参照してください。

連続ポイントデータ指定 { ポイントデータ | ポイント番号(初め:終わり) } [,出力コマンド] ...

このパラメーターそれぞれのポイントデータをカンマ(,)で区切って指定します。ポイントデータに抜けがなく、昇順、または降順に並んでいる場合は、2つのポイント番号をコロンで結合してP(1:5)のように指定できます。ポイントデータはすべて同一の座標系で定義されている必要があります。異なるローカル座標系で定義されたポイントが含まれる場合はエラーになります。また、垂直6軸型ロボットの場合、ポイントデータのハンド姿勢、肘姿勢、手首姿勢はすべて一致している必要があります。水平多関節型ロボットの場合、ハンド姿勢はすべて一致している必要があります。

動作に同期して、途中でI/Oの出力ポートをオン、またはオフする場合は、カンマ(,)で区切って出力命令を記述することができます。

通常は、連続ポイントデータは、下記のようにカンマで区切って指定します。

```
Curve "MyFile", 0, 0, 4, P1, P2, P3, P4
```

または、コロンを使って下記のように指定します

```
Curve "MyFile", 0, 0, 4, P(1:4)
```

上記の例では、P1、P2、P3、P4を使って曲線を指定しています。出力コマンドは、省略可能で、曲線動作中に出力操作を制御するときに使います。このコマンドは、I/OまたはメモリーI/Oのオンオフを指定します。出力命令は、直前の、連続ポイントデータの特定のポイントを、アームが通過した後に実行されます。1つのCurveステートメントに含むことのできる出力コマンドの数は、最大16個です。下記の例では、アームがP2を通過した後に"On 2"コマンドが実行され、その後アームは、P3からP10までのすべてのポイントを通過します。

```
Curve "MyFile", C, 0, 4, P1, P2, ON 2, P(3:10)
```

解説

この命令は、指定されたポイントデータによって、ロボットアームが自由曲線CP動作を行うためのファイルを作成し、そのデータをコントローラーのファイルに保存します。この命令によって作成されたデータは、CVMove命令によるCP動作実行時に使用されます。

Curve命令は指定された各ポイントのX, Y, Z, U, V, W座標値を独立に、3次スプライン関数を用いて計算し、それにより軌跡を生成します。よって、ポイント間の間隔が大きかったり、姿勢の変化が大きい場合には、生成される軌跡は想定しづらくなります。

軌跡を重視したい区間や姿勢が急激に変化する区間ではポイントを細かく指定してください。

動作時の速度・加減速度は、Curve命令前に指定しておく必要はありません。CVMove実行前ならSpeedSやAccelSなどの命令を使って変更できます。

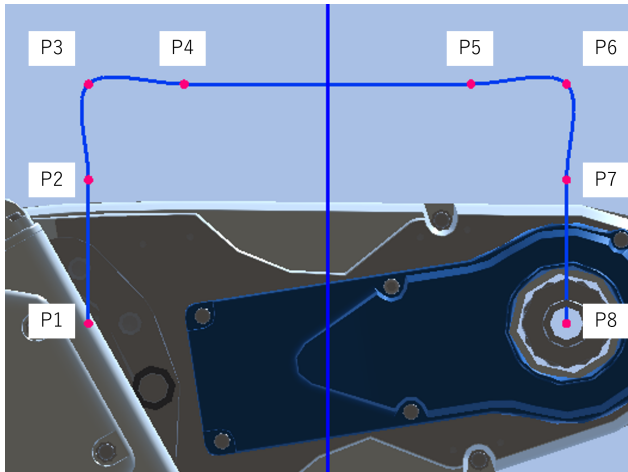
ポイントデータがローカル座標系で指定されている場合、ローカル座標系を基準にカーブファイルが作成されます。Curve命令実行後にLocal命令でローカル座標系を変更した場合、CVMove命令も変更後のローカル座標系に合わせてオフセットされます。

モード指定にECPを指定した場合、ECP座標を基準にカーブファイルが作成されます。Curve命令実行後にECPSets命令でECP座標を変更した場合、CVMove命令も変更後のECP座標に合わせてオフセットされます。

Ver8.1.1以降のファームウェアでは、曲線の補間方式を追加しています。通常は補間方式にIM_NORMALを指定してください。

補間方式がIM_NORMALのとき、隣り合う4点が直線上に並んでいる場合、真ん中の区間は必ず直線になります。また、ポイントデータの中のあるポイントを変更した場合、変更した点の前後2点間の経路が変わります。

下の図はP1~P8をカーブファイルに指定した場合の自由曲線になります。P3,P4,P5,P6は直線上に配置されているため、P4-P5間は必ず直線になります。P5を変更した場合は、P3-P4,P4-P5,P5-P6,P6-P7の区間の経路が変わります。



また、Ver8.1.1以降のファームウェアでは、動作速度の調整機能が有効です。そのため、姿勢が急激に変化する区間ではTCP速度を低下させ可能な限り加速度エラーが発生しないように調整します。この調整機能は、補間方式にIM_NORMALを指定した場合のみ有効です。

注意

■ 開曲線におけるポイント群のポイント数の範囲

開曲線では、3～1000ポイントを指定してください。

■ 閉曲線におけるポイント群のポイント数の範囲

RC800系コントローラーの場合、閉曲線では、3～1000ポイントを指定してください。

■ ポイント数が多い場合、処理時間が長くなります

Ver8.1.1より前のファームウェア、または、Ver8.1.1以降のファームウェアで補間方式にIM_CMPTBLを指定した場合、最大ポイント数でCurveコマンドを実行すると、開曲線では数秒程度、閉曲線では数十秒程度の時間がかかります。

処理時間が問題になる場合は補間方式にIM_NORMALを指定してください。

■ ファイルの互換性

Ver.7.5.1以降のファームウェアで作成されたファイルは、それより前のバージョンのファームウェアで使用できません。また、Ver.7.5.1より前のファームウェアで作成されたファイルは、Ver.7.5.1以降のファームウェアで使用できます。

Ver.8.1.1以降のファームウェアで作成されたファイルは、それより前のバージョンのファームウェアで使用できません。また、Ver.8.1.1より前のファームウェアで作成されたファイルは、Ver.8.1.1以降のファームウェアで使用できます。

起こりやすいエラー

■ 可動範囲外で、アームを動作させようとしたとき

Curve命令は、設定された曲線の動作範囲のチェックを行うことができません。これは、ロボットアームの動作中にユーザーが設定した曲線の軌跡が、可動範囲外に出るかもしれないことを意味します。このような場合、動作命令実行中に「動作範囲外」のエラーになります。

■ ポイントの間隔が不均一な場合

ポイント間隔が不均一な場合、生成される曲線の加速度が異常に高くなることがあります。このような場合、異常加速度エラーが発生することがあります。

CVMove実行中に異常加速度エラーが発生する場合、補間方式をIM_NORMALに変更してください。

参照

[AccelS関数](#), [Arc](#), [Arc3](#), [CVMove](#), [ECP](#), [Move](#), [SpeedS](#)

Curve使用例1

下記のプログラムは水平多関節型ロボットを用いた場合の例です。mycurve.CVTという名前のカーブファイルを作成します。ロボットはP1～P7を通過する曲線をトレースします。動作実行中、P2を通過時に出力ポートをオンします。

自由曲線を設定

```
> curve "mycurve", 0, 0, 4, IM_NORMAL, P1, P2, On 2, P(3:7)
```

アームをP1へJump命令で動作

```
> jump P1
```

定義した自由曲線"mycurve"でアームを移動

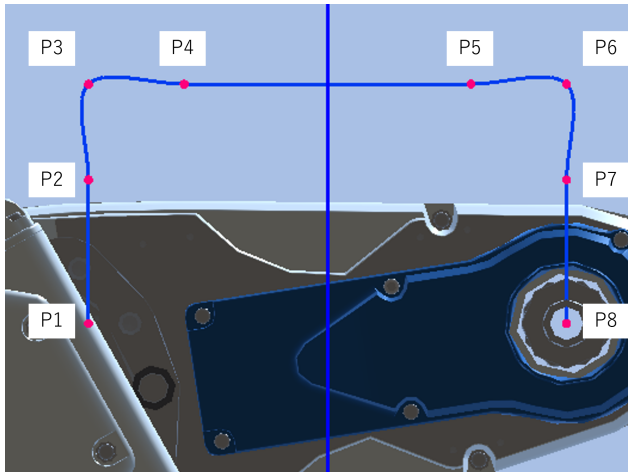
```
> cvmove "mycurve"
```

Curve使用例2 直線区間の作成

自由曲線動作を利用する際に、直線区間と曲線区間をギャップなく接続したい場合があります。4点を直線上に配置することで中間の2点間は完全に直線になりますが、端の区間は直線からズレます。この場合は、端の区間を短くすることでズレを低減できます。下記のプログラムは仮想コントローラー「GX8-C Sample」で試すことができます。

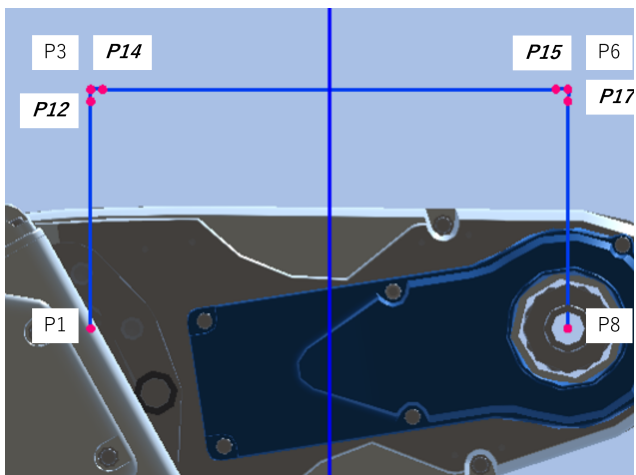
(1) コーナー部分の区間が長いケース P3,P4,P5,P6は直線上に並んでいるため、P4-P5の区間は直線になります。

```
Function Test2_1
    P1 = XY(100, 300, -50, 0)
    P2 = XY(100, 360, -50, 0)
    P3 = XY(100, 400, -50, 0)
    P4 = XY(60, 400, -50, 0)
    P5 = XY(-60, 400, -50, 0)
    P6 = XY(-100, 400, -50, 0)
    P7 = XY(-100, 360, -50, 0)
    P8 = XY(-100, 300, -50, 0)
    Motor On
    Power High
    Curve "Test2_1", 0, 0, 4, IM_NORMAL, P(1:8)
    Go P1
    CVMove "Test2_1"
Fend
```



(2) コーナー部分の区間が短いケース (1)のP2,P4,P5,P7の代わりにP12,P14,P15,P17を用いると、直線区間P14-P15を長く取ることができます。

```
Function Test2_2
  P1 = XY(100, 300, -50, 0)
  P12 = XY(100, 395, -50, 0)
  P3 = XY(100, 400, -50, 0)
  P14 = XY(95, 400, -50, 0)
  P15 = XY(-95, 400, -50, 0)
  P6 = XY(-100, 400, -50, 0)
  P17 = XY(-100, 395, -50, 0)
  P8 = XY(-100, 300, -50, 0)
  Motor On
  Power High
  Curve "Test2_2", 0, 0, 4, IM_NORMAL, P1, P12, P3, P14, P15, P6, P17, P8
  Go P1
  CVMove "Test2_2"
Fend
```



Curve使用例3 姿勢補正の有無

姿勢補正機能のサンプルです。カーブファイル「Test3_1」では姿勢補正なしでU座標を指定していないためツールは一定の姿勢で動作します。カーブファイル「Test3_2」では姿勢補正が有効なため、ツールは曲線の接線方向に対して一定の角度を保ったまま動作します。途中のポイントの姿勢は無視されます。下記のプログラムは仮想コントローラ「GX8-C Sample」で試すことができます。

(1) コーナー部分の区間が長いケース P3,P4,P5,P6は直線上に並んでいるため、P4-P5の区間は直線になります。

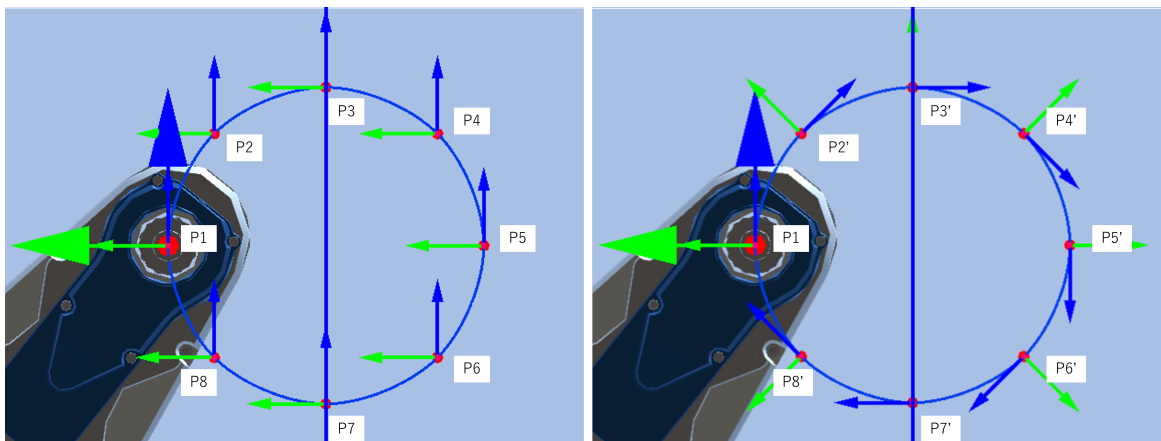
```
Function Test3
  Integer i
```



```

' 円周上にポイントを登録
Local 1, XY(0, 350, -50, 0)
For i = 0 To 8
    P(i) = XY(100 * Cos(2 * PI() * i / 8), 100 * Sin(2 * PI() * i / 8),
0, 0) /1
Next
Motor On
Power High
Curve "Test3_1", C, 0, 4, IM_NORMAL, P(0:8) '姿勢補正なし
Curve "Test3_2", C, 2, 4, IM_NORMAL, P(0:8) '姿勢補正あり
Go P0
CVMove "Test3_1"
Go P0
CVMove "Test3_2"
Fend

```



左:姿勢補正無し(Test3_1)/右:姿勢補正有り(Test3_2)

3.7.78 CVMove

Curve命令で定義した自由曲線CP動作を実行します。

書式

CVMove ファイル名[CP] [Till | Find] [SYNC]

パラメーター

ファイル名

Curve命令で作成され、コントローラーに保存されたファイル名を、文字列式または直接文字列で指定します。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

CP

最終ポイントの後のパスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

CVMoveは、ファイルのデータが設定する自由曲線CP動作を実行します。このファイルは、あらかじめCurve命令によって作成されていなければなりません。

ファイル名に拡張子がないと、自動的に".cvt"がつきます。

CVMoveによるCP動作の速度と加減速度は、SpeedSやAccelS命令によって変更することができます。

Curve命令において、ポイントデータがローカル座標系で指定されている場合、ローカル座標系を基準にカーブファイルが作成されます。Curve命令実行後に、Local命令でローカル座標系を変更した場合、CVMove命令も変更後のローカル座標系に合わせてオフセットされます。

Curve命令のモード指定にECPを指定した場合、ECP座標を基準にカーブファイルが作成されます。Curve命令実行後に、ECP命令でECP座標を変更した場合、Curve命令も変更後のECP座標に合わせてオフセットされます。

CVMove命令を実行したときの現在位置とカーブファイルで指定されている最初のポイントの位置が一致していない場合、カーブファイルの最初のポイントまでローパワーモードでPTP動作します。CVMove命令を実行する場合は、事前にカーブファイルの開始点までアームを動作してください。

現在位置は姿勢フラグも含めて一致している必要があります。そのため、垂直6軸型ロボットでは見かけ上同じ位置にいてもJ6関節が1回転することがあります。

CVMove実行時には、周辺装置との干渉に十分注意してください。特に、垂直6軸型ロボット (Nシリーズを含む) では隣り合うポイント間で急激に姿勢を変化させようとする、3次スプライン関数の性質上、その前後のポイントから姿勢の変化が開始され想定しない動作となる可能性があります。CVMove実行時には周辺装置との干渉に注意し、軌跡の確認を十分行ってください。ポイント間で姿勢が急激に変化する場合は、ポイント間の姿勢変化が少なくなるように、ポイントを細かく指定してください。

CVMove実行中に異常加速度エラーが発生する場合、Curve命令で作成する際の補間方式をIM_NORMALに変更してください。

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

参照

[AccelS関数](#), [Arc](#), [Arc3](#), [Curve](#), [Move](#), [SpeedS](#), [Till](#), [TillOn関数](#)

CVMove使用例

下記のプログラムは水平多関節型ロボットを用いた場合の例です。mycurve.CVTという名前のカーブファイルを作成します。ロボットはP1～P7を通過する曲線をトレースします。動作実行中、P2を通過時に出力ポートをオンします。

自由曲線を設定

```
> Curve "mycurve", 0, 0, 4, IM_NORMAL, P1, P2, On 2, P(3:7)
```

アームをP1へJump命令で動作

```
> Jump P1
```

定義した自由曲線"mycurve"でアームを移動

```
> CVMove "mycurve"
```

閉曲線と開曲線の違いや、姿勢補正の有無についての使用例はCurve命令の項目を参照してください。

3.7.79 CX, CY, CZ, CU, CV, CW, CR, CS, CT

ポイントデータの座標値を設定します。CV, CWは、垂直6軸型ロボット (Nシリーズを含む)のみ使用することができます。

CRは、Joint型ロボットのみ使用することができます。

CS, CTは、付加軸が設定されているロボットのみ使用することができます。

書式

CX(座標) = 値

CY(座標) = 値

CZ(座標) = 値

CU(座標) = 値

CV(座標) = 値

CW(座標) = 値

CR(座標) = 値

CS(座標) = 値

CT(座標) = 値

パラメーター

座標

P番号、P(式)、またはポイントラベルを指定します。

値

設定する座標値を実数値で指定します。

参照

[CX, CY, CZ, CU, CV, CW, CR, CS, CT関数](#)

CX, CY, CZ, CU, CV, CW, CR, CS, CT使用例

```
CX(pick) = 25.34
```

3.7.80 CX, CY, CZ, CU, CV, CW, CR, CS, CT関数

ポイントデータの座標値を取得します。

CV, CW関数は、垂直6軸型ロボット (Nシリーズを含む)のみ使用することができます。

CS, CT関数は、付加軸が設定されているロボットのみ使用することができます。

書式

CX(座標)

CY(座標)

CZ(座標)

CU(座標)

CV(座標)

CW(座標)

CR(座標)

CS(座標)

CT(座標)

パラメーター

座標

ポイントデータを指定します。

戻り値

指定された座標値を返します。

- CX, CY, CZの各関数の戻り値: 実数値 (単位:mm)です。
- CU, CV, CWの各関数の戻り値: 実数値 (単位:deg)です。
- CS, CTの各関数の戻り値: 実数値 (単位:mm)、もしくは実数値 (単位:deg)です。付加軸の設定により異なります。

解説

指定したポイントデータの座標値を取得します。

ロボットの現在位置の座標値を取得する場合は、パラメーターにHereを使います。

参照

[CX, CY, CZ, CU, CV, CW, CR, CS, CT](#)

CX, CY, CZ, CU, CV, CW, CR, CS, CT関数使用例

次の例ではポイント "pick"からX座標値を取り出し、変数xにセットします。

```
Function cptest
  Real x
  x = CX(pick)
  Print "The X Axis Coordinate of point 'pick' is", x
End
```

3.8 D

3.8.1 Date

日付の表示を行います。

書式

Date

結果

現在の日付を表示します。

参照

[Time, Date\\$関数](#)

Date使用例

コマンドウィンドウでの実行例

```
> Date  
2009/08/01
```

3.8.2 Date\$関数

現在の日付を返します。

書式

Date\$

戻り値

日付を文字列で返します。

書式は、yyyy/mm/dd [西暦年号/月/日]です。

参照

[Date, Time, Time\\$関数](#)

Date\$関数使用例

```
Print "Today's date: ", Date$
```

3.8.3 Declare

DLL (ダイナミックリンクライブラリー)で定義されている外部ファンクションを呼び出します。

書式

Declare ファンクション名, "DLLファイルパス", "DLL内ファンクション名" [, (引数リスト)] As関数型

パラメーター

ファンクション名

プログラムから呼び出すときのファンクション名を指定します。

DLLファイルパス

ライブラリーファイルのパスと名前を、コーテーションマーク (")で囲んだ文字列か、#defineで定義するマクロで指定します。

パスが指定されない場合はRC+ はカレントプロジェクトディレクトリーの中のファイルを検索します。見つからない場合は、ファイルはWindows system32ディレクトリーにあると仮定します。拡張子は省略できますが、省略した場合は、.DLLであると仮定されます。

DLL内ファンクション名

オプションパラメーターです。DLLの中の実際のファンクション名か、ファンクションインデックスを指定します。名前は大文字と小文字を区別します。ファンクション名はコーテーションマーク (")で囲んだ文字列で指定します。インデックスを使用する場合はインデックスの前に、#をつけます。省略した場合 "ファンクション名"パラメーターで指定されたファンクション名をDLL内ファンクション名として使用します。

引数リスト

DLL引数のリストです。引数は下記の書式を使用してください。省略可能です。

[{ByRef | ByVal}] 変数名[()] As変数の型

ByRef

呼び出すファンクションの変数を参照する場合は、ByRefを指定します。この場合、ファンクション内での引数の変更を呼び出し側の変数に反映させることができます。参照で渡された値は変更することができます。ByValを指定して変数を参照する場合、ファンクション宣言内でも変数をByRef指定する必要があります。

ByVal

デフォルト設定です。値 (ByVal)によってパラメーターは渡されます。値だけが渡されるので、ファンクションが戻るとき、その変数を変えることはできません。呼び出しているファンクションで、変数の値の変更をしないとき、指定します。省略可能です。

変数名

必須パラメーターです。引数を意味する変数名で、変数の名前をつけるときの規則にしたがい名前をつけます。配列変数を引数として使用する時には、ByRefを必ず指定してください。

変数の型

必須パラメーターです。引数の型を宣言します。

関数型

必須パラメーターです。関数型を宣言してください。

解説

現在のプログラムからDLLファンクションを呼び出すときに使います。Declareは、ファンクションの外で使用してください。

Declareステートメントは、コンパイル時にDLLファイルとファンクションが存在していることを確認します。

エラー2473「Epson RC+との連携機能に失敗しました(ビジーまたは未初期化)」が発生する場合の対応は、以下の通りです。

- Epson RC+がコントローラーに接続されていることを確認してください。DLLファンクションはPC上で実行されます。
- WindowsStatus関数を使用して、DLL呼び出しが利用可能であることを確認してください。

数値変数をByValで渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a As Long) As Long
VC++ long _stdcall MyDllFunc(long a);
```

文字列変数をByValで渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (a$ As String) As Long
VC++ long _stdcall MyDllFunc(char *a);
```

数値変数をByRefで渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a As Long) As Long
VC++ long _stdcall MyDllFunc(long *a);
```

文字列変数をByRefで渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a$ As String) As Long
VC++ long _stdcall MyDllFunc(char *a);
```

ByRef で文字列を渡すと、DLLの中で文字列を変更することができます。文字列は最長255文字まで使えます。最大文字数を超えないように注意してください。SPEL+は文字列変数のために、内部で固定の255文字領域を確保します。

数値配列をByRefで渡す

```
SPEL: Declare MyDLLFunc, "mystuff.dll", "MyDLLFunc", (ByRef a() As Long) As Long
VC++ long _stdcall MyDllFunc(long *a);
```

DLLファンクションからの戻り値

DLLファンクションは、文字列(String型)以外のあらゆるデータ型で戻り値を返すことができます。

文字列を返す必要がある場合は、上記の「文字列変数をByRefで渡す」を参考に、引数として文字列変数を指定して下さい。

戻り値に文字列変数を指定すると、エラー3614:「Declareコマンドで、戻り値に文字列を指定することはできません。」が発生します。

変数の型

Epson RC+ 8.0のデータ型とC/C++の変数型は、下表のように対応しています。

Epson RC+ 8.0に対応するデータがないため、C/C++のbyte型や構造体は使用することができません。

Epson RC+ 8.0とC/C++ データ型対応表

Epson RC+ 8.0	C/C++
Boolean	short
Byte	short
Short	short

Epson RC+ 8.0	C/C++
Integer	short
Long	int
Real	float
Double	double
String	char [256] * Nullを含む

プログラム例

```
Declare ReturnLong, "mystuff.dll", "ReturnLong", As Long

Function main
    Print "ReturnLong = ", ReturnLong
Fend
```

参照

[Function...Fend](#)

Declare使用例

- ' Declare命令で外部ファンクションを定義します。
- ' DLLファイルパスにフルパスを設定しない場合、カレントプロジェクトフォルダー、
- ' Windows System32フォルダーにDLLファイルを置くことができます。

```
Declare MyDLLTest, "mystuff.dll", "MyDLLTest" As Long
```

```
Function main
    Print MyDLLTest
Fend
```

- ' Declare命令で2つのInteger型引数を持つ外部ファンクションを定義します。

```
#define MYSTUFF "mystuff.dll"
```

```
Declare MyDLLCall, MYSTUFF, "MyTestFunc", (var1 As Integer, var2 As Integer) As Integer
```

- ' Declare命令で外部ファンクションをフルパス指定し、
- ' ファンクションをインデックスで指定して定義します。

```
Declare MyDLLTest, "c:\mydlls\mystuff.dll", "#1" As Long
```

3.8.4 DegToRad関数

角度をラジアンに変換します。

書式

DegToRad(角度)

パラメーター

角度

ラジアンに変換する度の値 (実数値)を指定します。

戻り値

Double型のラジアン値を返します。

参照

[Atan関数](#), [Atan2関数](#), [RadToDeg関数](#)

DegToRad関数使用例

```
s = Cos (DegToRad (x) )
```

3.8.5 Del

ファイルを削除します。

書式

Del ファイル名

パラメーター

ファイル名

削除するパスとファイル名を指定します。ファイル名は拡張子をつけます。詳細は、ChDiskを参照してください。

解説

指定したファイルを削除します。

Del使用例

コマンドウィンドウでの操作例

```
> Del TEST.PTS      'カレントディレクトリーからポイントファイルを削除
> Del c:\TEST.PTS   'NG
!! エラー: 7213 指定されたファイルが見つかりません。
> Del c:\TEST.PTS   'OK
```

3.8.6 DeleteDB

オープンしているデータベース内のテーブルからデータを削除します。

書式

DeleteDB #データベース番号, テーブル名[, 削除条件]

パラメーター

データベース番号

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

テーブル名

データ削除を行うテーブル名を指定します。

削除条件

削除条件を指定します。AND, ORを使用して複合条件を指定できます。削除条件を指定しない場合、テーブルの全データが削除されます。

解説

オープンされたデータベースの指定されたテーブルから削除条件に一致するデータを削除します。

オープンされたデータベースがExcelブックの場合、このコマンドは、実行できません。

注意

RC+ がインストールされているPCの接続が必要です

参照

[OpenDB](#), [CloseDB](#), [SelectDB関数](#), [UpdateDB](#)

3.8.7 DiffPoint関数

指定された2つのポイントの差分を返します。

書式

DiffPoint (ポイントデータ1, ポイントデータ2)

パラメーター

ポイントデータ1

1つ目のポイントデータを指定します。

ポイントデータ2

2つ目のポイントデータを指定します。

戻り値

ポイントデータ1から見たポイントデータ2の位置姿勢をポイントデータとして返します。

解説

ポイントデータ1を原点とした座標系における、ポイントデータ2の位置姿勢をポイントデータとして返します。
返されるポイントデータのローカル番号や、Handなどのフラグ情報はデフォルト値を返します。

2つのポイントデータについて、いずれかのポイントデータで、未定義の値がある場合、"0"として計算されます。

例えば、Point1を"XY (10,0,0,0,0,0): ST (10, 10)"として、Point2 を"XY (10,0,0,0,0,0)"と指定した場合、Point2では、SおよびTの値が定義されず、Point1では定義されるため、Point2のSおよびTの値を、"0"として計算した値を返します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

DiffPoint関数使用例

・ポイントP1から見たP2の位置姿勢を表示します。

```
Print DiffPoint (P1, P2)
```

・現在位置 (Here) から見たP1の位置姿勢を表示します。

```
Print DiffPoint (Here, P1)
```

3.8.8 DiffToolOrientation関数

指定された2つのポイントで実現されるそれぞれのツール姿勢の変化量を表すため、ツール座標系の各座標軸のなす角度 (単位: 度)を返します。

書式

DiffToolOrientation (ポイントデータ1, ポイントデータ2, 軸番号)

パラメーター

- ポイントデータ1
- 1つ目のポイントデータを指定します。
- ポイントデータ2
- 2つ目のポイントデータを指定します。
- 軸番号
- 角度変化量を求めるツール座標系の座標軸を指定します。

定数	値
COORD_X_PLUS	1: +X軸
COORD_Y_PLUS	2: +Y軸
COORD_Z_PLUS	3: +Z軸
COORD_ALL	4: 任意軸

戻り値

角度 (0～180度の実正数値)

解説

指定された2つのポイントデータのそれぞれで実現されるツール姿勢がなす姿勢変化量を、指定されたツール座標系の座標軸がなす角度 (0～180度の実正数値)によって返します。結果はポイントデータ1, 2の順番には影響を受けません。また、結果は2点間の原点の位置関係 (X, Y, Zの座標値)には影響を受けません。

COORD_ALLを指定した場合、任意軸回りの回転量を返します。任意軸とは、2つの姿勢 (U, V, W)があるとき、仮想的な軸 (1本の直線)回りに、1度の回転で移動可能である軸のことです。各軸に限定せず、総合的な回転角度を求める場合に使用します。

注意

- 使用可能なコントローラーについて
- T/VTシリーズでは、軸番号にCOORD_ALLを指定することはできません。

DiffToolOrientation関数使用例

```
'ポイントP1とP2のツール座標Z軸がなす角度を表示します。  
  
Print DiffToolOrientation(P1, P2, COORD_Z_PLUS)
```

3.8.9 DispDev

現在のディスプレイデバイスを設定します。

書式

DispDev (デバイスID)

パラメーター

デバイスID

指定するディスプレイデバイスのデバイスIDを指定します。

- 21 RC+
- 20 TP4

以下の定数を使用することも可能です。

- DEVID_SELF 21
- DEVID_TP 24
- DEVID_TP3 20

参照

[DispDev関数](#)

DispDev使用例

```
DispDev DEVID_TP
```


3.8.10 DispDev関数

現在のディスプレイデバイスを返します。

書式

DispDev

戻り値

デバイスIDが設定されているディスプレイデバイスの整数値を返します。

- 21 RC+
- 20 TP4

参照

[DispDev](#)

DispDev使用例

```
Print "The current display device is ", DispDev
```

3.8.11 Dist関数

2つのロボット座標間の距離を返します。

書式

Dist (座標1, 座標2)

パラメーター

座標1, 座標2

2つのロボット座標を指定します。

戻り値

2つの座標間の距離を実数値で返します。(単位: mm)

解説

付加軸を使用している場合でも、ロボットの移動距離のみを返します。例えば、付加軸を走行軸に使用するケースでも付加軸移動距離を考慮しません。

ジョイント型ロボットでは、本関数が返す値は意味を持ちません。

参照

[CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#)

Dist関数使用例

```
Real distance
```

```
distance = Dist(P1, P2)
```

3.8.12 Do...Loop

条件が一致している間、あるいは、不一致の場合は条件が一致するまで、DO...LOOP間を繰り返します。

書式

Do [{ While | Until } 条件式]

[ステートメント]

[Exit Do]

[ステートメント]

Loop

または、次の書式を使います。

Do

[ステートメント]

[Exit Do]

[ステートメント]

Loop [{ While | Until } 条件式]

Do Loop ステートメントの書式には、条件式とステートメントがあります。

条件式

TrueまたはFalseを表わす数字、または文字列式。条件式が空 (Null) の場合、条件はFalseとして扱われます。省略可能です。

ステートメント

条件が一致している間、あるいは、条件が一致するまで繰り返される、1つ以上のステートメント

解説

Do...Loopを抜けるもう1つの方法として、Do...Loop中、どこでも、いくつでもExit Doステートメントを挿入することができます。Exit Doは、If...Thenなど、いくつかの条件を評価した後によく用いられます。If...ThenでExit Doステートメントが用いられると、制御をLoopの次のステートメントに移します。

ネスティングされたDo...Loopステートメントの中で用いられると、Exit Doは、それが起こったループの1レベル上のループに制御を移します。

注意

- ループ文の中でXQT命令を頻繁に繰り返すような使い方はしないでください

Do...Loopなどループ文の中でXQT命令を頻繁に繰り返すような使い方はしないでください。コントローラーがハングアップ状態になる可能性があります。このような使い方をする場合は、Wait命令 (Wait 0.1) を追加してください。

- 空の無限ループ、および無限ループと同様の処理は、できる限りWait命令と併用してください

空のDo...Loop、および同様の処理は、システムに影響を与える可能性があるため、できるだけ行わないでください。コントローラーでは、無限ループタスクの検出を行っており、システムに影響が出るとコントローラーが判断した場合、エラー2556 (過剰なループを検出した) が発生することがあります。ループを必要とする演算や、I/O待ちを行う場合は、Wait命令 (Wait 0.1) などをループ処理内で実行し、CPUの占有を回避してください。

- Exit Do を使用せず、ネスト構造からループを抜けた場合

Exit Do命令以外のコマンド(Gosub命令、Goto命令、Call命令など)でループから抜けるプログラムを繰り返し実行すると、エラー2020が発生します。ループの途中で抜ける場合は、Exit Do 命令で抜けるようにしてください。

参照

[For...Next](#), [Select...Send](#)

Do...Loop使用例

```
Do While Not Lof(1)
  Line Input #1, tLine$
  Print tLine$
Loop
```

3.8.13 Double

Double型変数を宣言します。(8バイトの倍精度数)

書式

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

パラメーター

変数名

Double型と宣言する変数の名前を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Doubleは、指定した変数をDouble型に宣言します。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

Double型の有効桁数は14桁です。

参照

[Boolean](#), [Byte](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Double使用例

下記は、Doubleを使っていくつかの変数を宣言したプログラム例です。

```
Function doubletest
  Double var1
  Double A(10)          'Double型の1次元配列
  Double B(10, 10)      'Double型の2次元配列
  Double C(5, 5, 5)     'Double型の3次元配列
  Double arrayvar(10)
  Integer i
  Print "Please enter a Number:"
  Input var1
  Print "The variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter a Number:"
    Input arrayvar(i)
    Print "Value Entered was ", arrayvar(i)
  Next i
Fend
```

3.9 E

3.9.1 ECP

現在のECP (外部制御点)の選択または表示をします。

書式

(1) ECP ECP番号

(2) ECP

パラメーター

ECP番号

次に続く動作命令で、16あるECP設定 (整数値0～15)のうち、どれを使うかを指定します。省略可能です。

ECP 0は、ECP選択を無効にします。

結果

パラメーターを省略すると、現在設定されているECP番号を表示します。

解説

ECP命令は、ECP番号で指定した外部制御点を選択します。

注意

- このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。
- 電源オフによるECP選択への影響

選択されたECPは、主電源をオフすると、選択状態がクリアされます。

参照

[ECPSet](#)

ECP使用例

```
>ecpset 1, 100, 200, 0, 0  
>ecp 1
```

3.9.2 ECP関数

現在指定されているECP (外部制御点)番号を返します。

書式

ECP

戻り値

現在指定されているECP番号を整数値で返します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

[ECP](#)

ECP関数使用例

```
Integer savECP  
  
savECP = ECP  
ECP 2  
Call Dispense  
ECP savECP
```

3.9.3 ECPClr

外部制御点設定をクリア (未設定) します。

書式

ECPClr ECP番号

パラメーター

ECP番号

1から15の外部制御点のうち、クリア (未設定) する番号を整数値で指定します。(ECPの0番は初期設定値なので、クリアはできません。)

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLSet](#)

ECPClr使用例

```
ECPClr 1
```


3.9.4 ECPDef関数

外部制御点の設定状態を返します。

書式

ECPDef (ECP番号)

パラメーター

ECP番号

状態を呼び出したいECP番号を整数値で指定します。

戻り値

指定したECP番号の外部制御点が設定されている場合は"True"、設定されていない場合は"False"を返します。

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLClr](#), [TLSet](#)

ECPDef使用例

```
Function DisplayECPDef(ecpNum As Integer)

    If ECPDef(ecpNum) = False Then
        Print "ECP ", ecpNum, "is not defined"
    Else
        Print "ECP ", ecpNum, ": ",
        Print ECPSet(ecpNum)
    EndIf
End
```

3.9.5 ECPSet

外部制御点の定義または表示をします。

書式

(1) ECPSet ECP番号, ECPポイント指定

(2) ECPSet ECP番号

(3) ECPSet

パラメーター

ECP番号

外部制御点として定義する番号を、式または1～15の整数値で指定します。

ECPポイント指定

P番号またはP(式), ポイントラベル, ポイントデータで指定します。

戻り値

すべてのパラメーターを省略すると、現在のECP設定を表示します。

ECP番号だけを指定すると、指定のECP設定を表示します。

解説

外部制御点を設定します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

ECPSet使用例

```
ECPSet 1, P1  
ECPSet 2, 100, 200, 0, 0
```

3.9.6 ECPSet関数

指定したECP番号に割り当てられている外部制御点のポイントデータを返します。

書式

ECPSet (ECP番号)

パラメーター

ECP番号

ポイントデータを呼び出したいECP番号を整数値で指定します。

戻り値

指定したECP番号の外部制御点のポイントデータを返します。

注意

このコマンドは、外部制御点オプションがインストールされている場合のみ使用できます。

参照

[ECPSet](#)

ECPSet使用例

```
P1 = ECPSet (1)
```

3.9.7 ElapsedTime関数

タクトタイム計測用タイマーがスタートしてからの経過時間を単位秒で返します。

書式

ElapsedTime

戻り値

タクトタイム計測用タイマーの経過時間を実数値 (秒単位) で返します。タイマーの範囲は、0～約1.7E+31です。
タイマー解像度は、0.001秒です。

解説

タクトタイム計測用タイマーがスタートしてからの経過時間を返します。Tmr関数と異なり、プログラムが一時停止状態となっている時間は経過時間としてカウントされません。

タクトタイム計測用タイマーは ResetElapsedTimeでリセットできます。

```
Real overhead  
  
ResetElapsedTime  
overHead = ElapsedTime
```

注意

Runウィンドウからフォームを実行する場合、タクトタイム計測用タイマーはスタートしません。

タクトタイムを計測したい場合は、ファンクションを実行し、プログラムからGShowまたは、GShowDialogを使用してください。

参照

[ResetElapsedTime, Tmr関数](#)

ElapsedTime使用例

```
ResetElapsedTime      'タクトタイム計測用タイマーをリセット  
For i = 1 To 10        '10回実行  
    GoSub Cycle  
Next  
Print ElapsedTime / 10 'タクトタイムを計算して表示
```

3.9.8 Elbow

ポイントの肘姿勢を設定します。

書式

(1) Elbow ポイント指定 [, 設定値]

(2) Elbow

パラメーター

ポイント指定

P番号、P(式) またはポイントラベルを指定します。

設定値

整数または式を指定します。

- 1 = Above (/A)
- 2 = Below (/B)

戻り値

2つのパラメーターが両方省略されると、ロボットの現在位置の肘姿勢を表示します。

設定値パラメーターが省略されると、指定したポイントの肘姿勢を表示します。

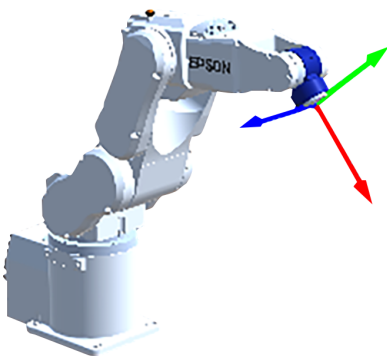
参照

[Elbow関数](#), [Hand](#), [J4Flag](#), [J6Flag](#), [Wrist](#)

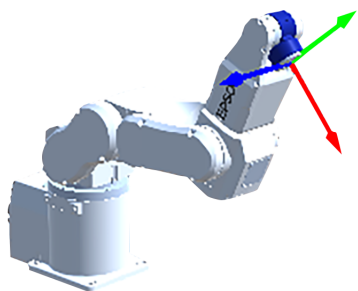
Elbow使用例

```
Elbow P0, Below  
Elbow pick, Above  
Elbow P(myPoint), myElbow
```

```
P1 = 0.000, 490.000, 515.000, 90.000, -40.000, 180.000
```



```
Elbow P1, Above  
Go P1
```



Elbow P1, Below
Go P1

3.9.9 Elbow関数

ポイントの肘姿勢を返します。

書式

Elbow [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置の肘姿勢を返します。

戻り値

- 1: Above (/A)
- 2: Below (/B)

参照

[Elbow](#), [Hand](#), [Wrist](#), [J4Flag](#), [J6Flag](#)

Elbow関数使用例

```
Print Elbow(pick)
Print Elbow(P1)
Print Elbow
Print Elbow(P1 + P2)
```

3.9.10 EncTemper

エンコーダーの現在温度を表示します。

書式

EncTemper[Joint number]

パラメーター

関節番号

関節番号を1~9の整数値または式で設定します。

付加軸のS軸は8、T軸は9になります。

解説

エンコーダーの現在温度 [°C]を表示します。

関節番号を省略した場合、すべての軸のエンコーダー温度を表示します。

エンコーダー温度取得に対応していないロボットや関節番号を指定した場合、-1000を表示します。

キーポイント

- 本コマンドでの温度取得に使用するセンサーは、校正されたものではありません。
- 相対的な変化を参照し、温度変化の傾向を把握するために使用してください。

参照

[EncTemper関数](#)

EncTemper使用例

全ての軸の温度を表示する例です

```
> EncTemper
  31.21          32.82      '第1関節 第2関節
  31.45          33.28      '第3関節 第4関節
  32.76          32.87      '第6関節 第5関節
 -1000.00       -1000.00    '第7関節 第8関節
 -1000.00                        '第9関節
```

第1関節の温度を表示する例です

```
> EncTemper 1
  31.21
```


3.9.11 EncTemper関数

エンコーダーの現在温度を返します。

書式

EncTemper(関節番号)

パラメーター

関節番号

関節番号を1~9の整数値、または式で設定します。

付加軸のS軸は8、T軸は9になります。

解説

エンコーダーの現在温度 [°C] を返します。

エンコーダー温度取得に対応していないロボットや関節番号を指定した場合、-1000を返します。



キーポイント

- 本コマンドでの温度取得に使用するセンサーは、校正されたものではありません。
- 相対的な変化を参照し、温度変化の傾向を把握するために使用してください。

参照

[EncTemper](#)

EncTemper関数使用例

第1関節のエンコーダー温度を取得して表示する例です。

```
> Print EncTemper(1)
-1000
```

3.9.12 Eof関数

ファイルのEOF (オープンしたファイルのポインターが終端にある)を返します。

書式

Eof (ファイル番号)

パラメーター

ファイル番号

30～63の整数値、または式で指定します。

戻り値

ファイルがEOFの場合は"True"、それ以外は"False"を返します。

解説

ファイルを読み出しモードで開いている場合に機能します。

AOpen, WOpenコマンドで開いたファイルの場合には、エラーになります。

参照

[Lof関数](#)

Eof関数使用例

```
Integer fileNum
String data$

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Do While Not Eof(fileNum)
    Line Input #fileNum, data$
    Print "data = ", data$
Loop
Close #fileNum
```

3.9.13 Era関数

エラーが発生した関節の番号を返します。

書式

Era [(タスク番号)]

パラメーター

タスク番号

0～32のタスク番号を整数で指定します。省略または"0"の場合は現在のタスクです。

戻り値

エラーが発生した関節番号を下記の0～9の整数値で知らせます。

- 0 - 現在のエラーは、関節が原因ではありません。
- 1 - 現在のエラーは、第1関節が原因です。
- 2 - 現在のエラーは、第2関節が原因です。
- 3 - 現在のエラーは、第3関節が原因です。
- 4 - 現在のエラーは、第4関節が原因です。
- 5 - 現在のエラーは、第5関節が原因です。
- 6 - 現在のエラーは、第6関節が原因です。
- 7 - 現在のエラーは、第7関節が原因です。
- 8 - 現在のエラーは、第8関節 (付加軸S)が原因です。
- 9 - 現在のエラーは、第9関節 (付加軸T)が原因です。

解説

Eraは、エラーが発生したとき、そのエラーがどの関節で発生したかを見つけ、その関節番号を知らせます。もし、関節が現在のエラーの原因でないときは、"0"を返します。

自動運転モード(AUTO)の通常タスクとNoPauseタスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。NoEmgAbortタスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、「エラー 2261」が発生します。タスク終了前に情報を取得するにはOnErrを使用してください。

参照

[Erl関数](#), [Err関数](#), [ErrMsg\\$関数](#), [Ert関数](#), [OnErr](#), [Trap](#) (ユーザー定義トリガー)

Era関数使用例

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
Exit Function
eHandler:
Print "The Error code is ", Err
Print "The Error Message is ", ErrMsg$(Err)
errTask = Ert
If errTask > 0 Then
  Print "Task number in which error occurred is ", errTask
  Print "The line where the error occurred is Line ", Erl(errTask)
  If Era(errTask) > 0 Then
    Print "Joint which caused the error is ", Era(errTask)
  EndIf
```

```
EndIf  
Fend
```

3.9.14 EResume

エラー処理ルーチンの終了後に、プログラムの実行を再開します。

書式

EResume [{ラベル | Next}]

解説

■ EResume

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、エラーの原因となったステートメントからプログラムの実行が再開されます。

呼び出されたファンクション内でエラーが発生した場合、エラー処理ルーチンを含むファンクション内のCallステートメントからプログラムの実行が再開されます。

■ EResume Next

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、エラーの原因となったステートメントの次のステートメントから、プログラムの実行が再開されます。

呼び出されたファンクション内でエラーが発生した場合、エラー処理ルーチンを含むファンクションが呼び出した最後のCallステートメントの次のステートメントから、プログラムの実行が再開されます。

■ EResume {ラベル}

エラー処理ルーチンと同じファンクション内でエラーが発生した場合、指定したラベルを含むステートメントから、プログラムの実行を再開します。

参照

[OnErr](#)

EResume使用例

```
Function main
  Integer retry

  OnErr GoTo eHandler
  Do
    RunCycle
  Loop
  Exit Function

eHandler:
  Select Err
  Case MyError
    retry = retry + 1
    If retry < 3 Then
      EResume '再実行
    Else
      Print "MyError has occurred ", retry, " times"
    EndIf
  Send
Fend
```

3.9.15 Erf\$関数

エラーが発生した関数名を返します。

書式

Erf\$ [(タスク番号)]

パラメーター

タスク番号

タスク番号を0～32の整数値で指定します。省略または"0"の場合は現在のタスクです。

戻り値

最後のエラーが発生した関数名を返します。

解説

Erf\$は、OnErrorとともに使います。Erf\$は、エラーが発生した関数名を返します。Erf\$を Err, Ert, Erl, Eraなどと組み合わせることで、発生したエラーについてより詳しい情報を集めることができます。

自動運転モード(AUTO) の通常タスクとNoPauseタスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。NoEmgAbortタスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、「エラー 2261」が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [ErrMsg\\$関数](#), [Ert関数](#), [OnError](#), [Trap](#) (ユーザー定義トリガー)

Erf\$関数使用例

下記は、次の内容を調べる簡単なプログラムです。

- どのタスクでエラーが発生したか (Ert関数)
- どの関数でエラーが発生したか (Erf\$関数)
- どこで発生したか (Erl関数)
- どの関節でエラーが発生したか (Era関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "Function at which error occurred is ", Erf$(errTask)
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
EndFunction
```

3.9.16 Erl関数

エラーが発生した行番号を返します。

書式

Erl [(タスク番号)]

パラメーター

タスク番号

タスク番号を0～32の整数値で指定します。省略または"0"の場合は現在のタスクです。

戻り値

最後のエラーが発生した行番号を返します。

解説

ErlはOnErrorとともに使います。Erlは、エラーが発生した行番号を返します。ErlをErr, Ert, Eraなどと組み合わせることで、発生したエラーについて詳しい情報を集めることができます。

自動運転モード(AUTO)の通常タスクとNoPauseタスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。NoEmgAbortタスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、「エラー 2261」が発生します。タスク終了前に情報を取得するにはOnErrorを使用してください。

参照

[Era関数](#), [Erf\\$関数](#), [Err関数](#), [ErrMsg\\$関数](#), [Ert関数](#), [OnError](#)

Erl関数使用例

下記は、次の内容を調べる簡単なプログラムです。

- どのタスクでエラーが発生したか (Ert関数)
- どこで発生したか (Erl関数)
- どんなエラーが発生したか (Err関数)
- どの関節でエラーが発生したか (Era関数)

```
Function main
  OnErr Goto eHandler
Do
  Call PickPlace
Loop
Exit Function
eHandler:
Print "The Error code is ", Err
Print "The Error Message is ", ErrMsg$(Err)
errTask = Ert
If errTask > 0 Then
  Print "Task number in which error occurred is ", errTask
  Print "The line where the error occurred is Line ", Erl(errTask)
  If Era(errTask) > 0 Then
    Print "Joint which caused the error is ", Era(errTask)
  EndIf
EndIf
Fend
```

3.9.17 Err関数

最新のエラーステータスを返します。

書式

Err [(タスク番号)]

パラメーター

タスク番号

タスク番号を0～32の整数値で指定します。"0"は、現在のタスクを指定します。省略可能です。

戻り値

エラーコードを整数値で返します。

解説

Err関数は、ユーザーに現在のエラーコードを知らせます。SPEL+のエラー対応機能とともにどのエラーが発生したか知らせ、適切に対応できるようにします。ErrはOnErrとともに用います。

コントローラーのエラーを取得するには、SysErr関数を使用します。

自動運転モード(AUTO)の通常タスクとNoPauseタスクでは、「自動運転中のエラー発生」になると実行を中断しタスク終了します。NoEmgAbortタスクやバックグラウンドタスクで本関数を使用するときに、すでに対象のタスクが終了している場合は、「エラー 2261」が発生します。タスク終了前に情報を取得するには OnErr を使用してください。

注意

プロジェクトプロパティのコンパイラオプションで「グローバル変数の宣言を各ファイルで必須」をチェックした場合はそれぞれのファイルで宣言が必要になります。

参照

[Era関数](#), [Erf\\$関数](#), [Erl関数](#), [ErrMsg\\$関数](#), [EResume](#), [Ert関数](#), [OnErr](#), [Return](#), [SysErr関数](#)

Err関数使用例

下記の例は、ポイントP0-P399があるかどうかをチェックする、簡単なユーティリティプログラムです。ポイントがなければ、それをユーザーに知らせるメッセージが画面に表示されます。各ポイントが定義されているか、CX命令を使ってテストしています。定義されていないポイントがあると、エラー処理に制御が移り、画面にそのポイントが定義されていないと表示します。

```
Function errtest
  Integer i, errnum
  Real x

  OnErr GoTo eHandle
  For i = 0 To 399
    x = CX(P(i))
  Next i
  Exit Function
,
,
'*****
'* Error Handler *
'*****
eHandle:
  errnum = Err
  '未定義ポイントを使用していないか確認
```



```
If errnum = 7007 Then
    Print "Point number P", i, " is undefined!"
Else
    Print "ERROR: Error number ", errnum, " Occurred."
EndIf
EResume Next
Fend
```

3.9.18 Errb関数

エラーが発生したロボットの番号を返します。

書式

Errb

戻り値

エラーが発生したロボット番号を返します。

解説

Errb関数は、エラーが発生したとき、そのエラーがどのロボットで発生したかを見つけ、そのロボット番号を知らせます。もしロボットが現在のエラーの原因でないときは、"0"を返します。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [ErrMsg\\$関数](#), [OnError](#), [Trap](#) (ユーザー定義トリガー)

Errb関数使用例

次のプログラム例では下記の内容を表示します。

- どのタスクでエラーが発生したか (Ert関数)
- どこで発生したか (Erl関数)
- どんなエラーが発生したか (Err関数)
- どの関節でエラーが発生したか (Era関数)
- どのロボットでエラーが発生したか (Errb関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
    Print "Robot number in which error occurred is ", errb
  EndIf
Fend
```

3.9.19 ErrMsg\$関数

指定したエラー番号のエラーメッセージを返します。

書式

ErrMsg\$(エラー番号, 言語番号)

パラメーター

エラー番号

メッセージを返すエラー番号を整数値で指定します。

言語番号

言語を次の整数値で指定します。省略可能です。

- 0 - 英語
- 1 - 日本語
- 2 - ドイツ語
- 3 - フランス語
- 4 - 中国語 (簡体字)
- 5 - 中国語 (繁体字)
- 6 - スペイン語

省略した場合、英語が指定されます。

戻り値

エラーコード表のエラーメッセージを返します。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [Ert関数](#), [OnErr](#), [Trap](#) (ユーザー定義トリガー)

ErrMsg\$関数使用例

下記は、次の内容を調べる簡単なプログラムです。

- どのタスクでエラーが発生したか (Ert関数)
- どこで発生したか (Erl関数)
- 関節でエラーが発生したか (Era関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
Fend
```

3.9.20 Error

ユーザー定義エラーを発生させます。

書式

(1) Error タスク番号, エラー番号

(2) Error エラー番号

パラメーター

タスク番号

タスク番号を0～32の整数値で指定します。"0"は、現在のタスクを指定します。省略可能です。

エラー番号

エラー番号を整数値で指定します。ユーザーエラー番号は8000～8999です。

解説

Errorステートメントは、システムまたはユーザー定義エラーを発生させます。Epson RC+ 開発環境のユーザーエラーエディターを使って、ユーザーエラーのラベルと記述を定義できます。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [OnErr](#)

Error使用例

```
#define ER_VAC 8000

If Sw(vacuum) = Off Then
    Error ER_VAC
EndIf
```

3.9.21 ErrorOn関数

コントローラーのエラー状態を返します。

書式

ErrorOn

戻り値

エラー状態のときは"True"、それ以外のときは"False"を返します。

解説

本関数は、NoEmgAbortタスク (Xqt 時にNoEmgAbortを指定して開始された特別なタスク) とバックグラウンドタスクでのみ使用します。

参照

[ErrorOn関数](#), [SafetyOn関数](#), [SysErr関数](#), [Wait](#), [Xqt](#)

ErrorOn関数使用例

下記例は、コントローラーがエラー状態になるのを監視し、エラーが発生した場合、エラー番号に応じてI/Oをオンオフするプログラムです。

注意

- Forcedフラグ

本プログラム例では、On/Off命令にForcedフラグを指定しています。

エラー発生中、および、非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

- エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

```
Function main

Xqt ErrorMonitor, NoEmgAbort
:
:
Fend

Function ErrorMonitor
  Wait ErrorOn
  If 4000 < SysErr Then
    Print "Mortion Error = ", SysErr
    Off 10, Forced
    On 12, Forced
  Else
    Print "Other Error = ", SysErr
    Off 11, Forced
    On 13, Forced
  EndIf
Fend
```

3.9.22 Ert関数

エラーが発生したタスク番号を返します。

書式

Ert

戻り値

エラーが発生したタスク番号を返します。

解説

Ert関数は、どのタスクでエラーが発生したかを取得するために使います。

エラーが発生したタスクなし (0), 通常タスク (1~32), バックグラウンドタスク (65~80), TRAPタスク (257~267)の番号を返します。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [ErrMsg\\$関数](#), [OnErr](#), [Trap](#) (ユーザー定義トリガー)

Ert関数使用例

次のプログラム例では下記の内容を表示します。

- どのタスクでエラーが発生したか (Ert関数)
- どこで発生したか (Erl関数)
- どの関節でエラーが発生したか (Era関数)

```
Function main
  OnErr Goto eHandler
  Do
    Call PickPlace
  Loop
  Exit Function
eHandler:
  Print "The Error code is ", Err
  Print "The Error Message is ", ErrMsg$(Err)
  errTask = Ert
  If errTask > 0 Then
    Print "Task number in which error occurred is ", errTask
    Print "The line where the error occurred is Line ", Erl(errTask)
    If Era(errTask) > 0 Then
      Print "Joint which caused the error is ", Era(errTask)
    EndIf
  EndIf
End
Fend
```

3.9.23 EStopOn関数

非常停止状態を返します。

書式

EstopOn

戻り値

非常停止状態のときは"True"、それ以外のときは"False"を返します。

解説

本関数は、NoEmgAbort タスク (Xqt 時に NoEmgAbortを指定して開始された特別なタスク)でのみ使用します。

参照

[ErrorOn関数](#), [SafetyOn関数](#), [Wait](#), [Xqt](#)

EstopOn関数使用例

下記例は、コントローラーが非常停止になるのを監視し、非常停止が発生した場合I/Oをオンオフするプログラムです。

注意

- Forcedフラグ

本プログラム例では、On/Off命令にForcedフラグを指定しています。

エラー発生中、および、非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

- エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

- 非常停止で出力ポートをOFF

本例のように、非常停止後もI/Oのオンオフをタスクが行う場合は、[セットアップ]-[システム設定]-[コントローラー]-[環境設定]-[非常停止で出力ポートをOFF]のチェックをはずすことを推奨します。この設定がチェックされたままの場合、コントローラーによるI/OのオフとタスクによるI/Oのオンのどちらが先に実行されるか保証されません。

```
Function main
    Xqt EStopMonitor, NoEmgAbort
    :
    :
Fend

Function EStopMonitor
    Wait EStopOn
    Print "EStop !!!"
    Off 10, Forced
    On 12, Forced
Fend
```

3.9.24 Eval関数

コマンドウィンドウのステートメントを実行し、エラー状態を返します。

書式

Eval(コマンド [, コマンドの出力結果])

パラメーター

コマンド

実行するコマンドを文字列で指定します。

コマンドの出力結果

コマンドの出力結果を格納する文字列変数を指定します。省略可能です。コマンドがエラーとなった場合は
"!Error: エラーコード"が返されます。出力結果が255文字を超えた場合、超えた結果は切り捨てられます。

戻り値

コマンドの実行によって戻されるエラーコードを返します。

コマンド実行がエラーとなっても、本関数そのものはエラーとなりません。また、システムログにもエラーは残りません。

コマンドが正常終了した場合は"0"を返します。

解説

Eval関数を使用すると、TCP/IPなどの通信ポートから任意のコマンドを実行できます。

実行できるコマンドは、コマンドウィンドウから実行可能なコマンドです。

コマンド文字列中にダブルクォーテーション(")を使用できます。

本関数の実行には、通常のステートメント実行より処理時間がかかります。

コマンドからの戻り値を得るためには、コマンドの出力結果パラメーターを使います。例えば、"Print Sw(1)"というコマンドに対し、コマンドの出力結果は、"1"あるいは"0"が返されます。

参照

エラーメッセージ一覧

Eval関数使用例

下記の例では、RS-232Cで読まれるコマンドをどのように実行するか示しています。コマンドが実行された後、エラーコードがホストに返されます。例えば、ホストからは"motor on"などのコマンドが送られます。

```
Integer errCode
String cmd$

OpenCom #1
Do
    Line Input #1, cmd$
    errCode = Eval(cmd$)
    Print #1, errCode
Loop
```


3.9.25 Exit

ループまたはファンクションを強制終了します。

書式

Exit { Do | For | Function }

解説

Exit ステートメントの書式には、下記があります。

Exit Do

Do...Loopステートメントから抜けます。これは、Do...Loop ステートメント内でのみ使うことができます。

Exit Doで、制御は、Loopステートメントの次のステートメントに移ります。ネスティングされた

Do...Loop ステートメント内で用いると、Exit Doのあるループの1レベル上のループに制御が移ります。

Exit For

Forループから抜けます。For...Nextループ内でのみ使うことができます。Exit For は、Nextステートメント

の次のステートメントに制御を移します。ネスティングされたForループで使われると、Exit Forのあるループの1レベル上のループに制御が移ります。

Exit Function

任意の位置でファンクションから抜きたいときに使います。プログラムの実行は、ファンクションを呼び出したステートメントの次のステートメントに継続します。

参照

[Do...Loop](#), [For...Next](#), [Function...Fend](#)

Exit使用例

```
For i = 1 To 10
  If Sw(1) = On Then
    Exit For
  EndIf
  Jump P(i)
Next i
```

3.9.26 ExportPoints

ポイントファイルを指定のパスにエクスポートします。

書式

ExportPointsファイル名, 保存先

パラメーター

ファイル名

エクスポートしたい特定のファイルを表す文字列式拡張子は、".pts"です。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

保存先

保存先のパスとファイル名を指定します。拡張子は、".pts"です。詳細は、ChDiskを参照してください。

解説

ExportPointsは、指定したポイントファイルを、PC上のフォルダーにコピーします。PC上のフォルダーに同じファイルが既に存在していた場合は上書きされます。

起こりやすいエラー

- 保存先パスが存在しない場合

指定した保存先パスが存在しない場合、エラーが発生します。

- 指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。

参照

[ChDir](#), [LoadPoints](#), [SavePoints](#), [FileExists](#)関数, [FolderExists](#)関数

ExportPoints使用例

```
Function main
  LoadPoints "robot1.pts"
  :
  SavePoints "robot1.pts"
  If FolderExists("c:\mypoints\") Then
    ExportPoints "robot1.pts", "c:\mypoints\model1.pts"
  EndIf
Fend
```

3.10 F

3.10.1 FbusIO_GetBusStatus関数

指定したフィールドバスの状態を返します。

書式

FbusIO_GetBusStatus(バス番号)

パラメーター

バス番号

フィールドバスシステム番号を表す整数式この番号は必ず16で、コントローラーのPC側にあるフィールドバスマスターボードに接続されているバスのIDです。

戻り値

- 0 - OK
- 1 - 未接続
- 2 - パワーオフ

解説

FbusIO_GetBusStatus関数では、フィールドバスの状態を確認することができます。

注意

このコマンドは、フィールドバスマスターオプションが有効のときのみ使用できます。

参照

[FbusIO_GetDeviceStatus関数](#), [FbusIO_SendMsg](#)

FbusIO_GetBusStatus関数使用例

```
Long sts  
sts = FbusIO_GetBusStatus(16)
```

3.10.2 FbusIO_GetDeviceStatus関数

指定したフィールドバスデバイスの状態を返します。

書式

FbusIO_GetDeviceStatus(バス番号, デバイスID)

パラメーター

バス番号

フィールドバスシステム番号を表す整数式この番号は必ず16で、コントローラーのPC側にあるフィールドバスマスターボードに接続されているバスのIDです。

デバイスID

デバイスのフィールドバスIDを表す整数式

戻り値

- 0 - OK
- 1 - 未接続
- 2 - パワーオフ
- 3 - 同期エラー: デバイスが初期化中、またはデバイスのボーレートが正しくありません。

解説

FbusIO_GetDeviceStatus関数では、フィールドバスデバイスの状態を確認することができます。

注意

このコマンドは、フィールドバスマスターオプションが有効時のみ使用できます。

参照

[FbusIO_GetBusStatus関数](#), [FbusIO_SendMsg](#)

FbusIO_GetDeviceStatus関数使用例

```
Long sts  
sts = FbusIO_GetDeviceStatus(16, 10)
```

3.10.3 FbusIO_SendMsg

フィールドバスI/Oデバイスに、メッセージを送り、回答を返します。

書式

FbusIO_SendMsg(バス番号, デバイスID, msgParam, sendData(), recvData())

パラメーター

バス番号

フィールドバスシステム番号を表す整数式この番号は必ず16で、コントローラーのPC側にあるフィールドバスマスターボードに接続されているバスのIDです。

デバイスID

デバイスのフィールドバスIDを表す整数式

msgParam

メッセージパラメーターを表す整数式DeviceNetには使用できません。

sendData

デバイスに送信されるデータをByte型の配列で指定します。この配列の次元は、送信されるバイト数の次元と同じでなくてはなりません。データを送信しない場合は、"0"を指定します。

recvData

デバイスから受信されるデータをByte型の配列で指定します。この配列は自動的に受信したバイト数に応じた次元数に変換されます。

解説

FBusIO_SendMsg は、フィールドバスI/O デバイスに対するクエリーとして用います。サポートするメッセージについては、機器メーカーにお問合わせください。

注意

このコマンドは、フィールドバスマスターオプションが有効時のみ使用できます。

参照

[FbusIO_GetBusStatus関数](#), [FbusIO_GetDeviceStatus関数](#)

FbusIO_SendMsg使用例

```
' DeviceNet デバイスに明示メッセージを送る
Byte sendData(5)
Byte recvData(0)
Integer i

sendData(0) = &H0E ' コマンド
sendData(1) = 1     ' クラス
sendData(3) = 1     ' インスタンス
sendData(5) = 7     ' 属性
' DeviceNet のmsgParamは"0"
FbusIO_SendMsg 16, 1, 0, sendData(), recvData()
' 回答を表示
For i = 0 to UBound(recvData)
  Print recvData(i)
Next i
```

```
' Profibus デバイスにメッセージを送る
Byte recvData(0)
Integer i

' msgParamはサービス番号
FbusIO_SendMsg 16, 1, 56, 0, recvData()
' 回答を表示
For i = 0 to UBound(recvData)
    Print recvData(i)
Next i
```

3.10.4 FileDateTime\$関数

ファイルの日付と時間を返します。

書式

FileDateTime\$(ファイル名)

パラメーター

ファイル名

確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDiskを参照してください。

注意

ネットワークパスが使用できます。

戻り値

ファイルの最終修正日時を下記のフォーマットで返します。

月/日/西暦年号 時間:分:秒

参照

[FileExists関数](#), [FileLen関数](#)

FileDateTime\$関数使用例

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```

3.10.5 FileExists関数

ファイルの有無をチェックします。

書式

FileExists(ファイル名)

パラメーター

ファイル名

確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDiskを参照してください。

注意

ネットワークパスが使用できます。

戻り値

- ファイルが存在する場合: True
- ファイルが存在しない場合: False

参照

[FolderExists関数](#), [FileLen関数](#), [FileDateTime\\$関数](#)

FileExists関数使用例

```
String myPath$
myPath$ = "c:\TEST\TEST.DAT"

If FileExists(myPath$) Then
    Print "Last access date and time: ", FileDateTime$(myPath$)
    Print "Size: ", FileLen(myPath$)
EndIf
```


3.10.6 FileLen関数

ファイルサイズを返します。

書式

FileLen(ファイル名)

パラメーター

ファイル名

確認するファイル名を文字列で指定します。ドライブ名とパス名を含みます。ファイル名だけを指定した場合、カレントディレクトリーにあるファイルを指します。詳細は、ChDiskを参照してください。

注意

ネットワークパスが使用できます。

戻り値

ファイルのバイト数を返します。

参照

[FileDateTime\\$関数](#), [FileExists関数](#)

FileLen関数使用例

```
String myPath$  
myPath$ = "c:\TEST\TEST.DAT"  
  
If FileExists(myPath$) Then  
    Print "Last access date and time: ", FileDateTime$(myPath$)  
    Print "Size: ", FileLen(myPath$)  
EndIf
```

3.10.7 Find

動作命令中に座標を保存するための条件の設定と表示をします。

書式

Find [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子(=, <>, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

- 関数: Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
- 変数: Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

- 演算子: And, Or, Xor

[例]

```
Sense Sw(5) = On  
Sense Sw(5) = On And Sw(6) = Off
```

解説

Find ステートメントは、単独で記述するか、動作命令ステートメントの修飾子として記述してください。

Find条件式には、上記の関数を1つ以上含んでいなくてはなりません。

Find条件式に変数が含まれたときは、その値は、Find条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。複数のFindステートメントを記述することもできます。その場合、最後に実行されたFind 条件が有効となります。

パラメーターを省略すると、現在のFind設定が表示されます。

注意

- コントローラー電源オン時のFind 設定

コントローラー電源オン時のFind 条件は未設定です。電源オン後は、条件を設定してから使用してください。

- Find 条件成立をチェックするPosFound関数

Find修飾子を用いた動作コマンドを実行した後で、Find条件が成立したかどうかを、PosFound関数を用いてチェックすることができます。

- イベント条件式で変数を使用する場合

- 使用できる変数型は整数型(Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が0.01秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。

- システム内で使用できる変数待ちの数に制限があります。1システム内で使用できる変数待ちの個数は最大64個です (Waitなどのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数をByrefで参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。

参照

[FindPos関数](#), [Go](#), [Jump](#), [PosFound関数](#), [SF_GetStatus関数](#)

Find使用例

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```

3.10.8 FindPos関数

動作命令実行中にFindで保存した座標を返します。

書式

FindPos

戻り値

動作コマンド実行中にFindで保存した座標を返します。

参照

[Find](#), [Go](#), [Jump](#), [PosFound関数](#), [CurPos関数](#), [InPos関数](#)

FindPos関数使用例

```
Find Sw(5) = On
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Cannot find the sensor signal."
EndIf
```

3.10.9 Fine

目標位置の位置決め終了判断範囲の設定、表示をします。

書式

(1) Fine 第1関節設定値, 第2関節設定値, 第3関節設定値, 第4関節設定値 [, 第5関節設定値, 第6関節設定値] [, 第7関節設定値] [, 第8関節設定値, 第9関節設定値]

(2) Fine

パラメーター

第1関節設定値

第1関節の位置決め許容範囲を、0～65535の整数で指定します。

第2関節設定値

第2関節の位置決め許容範囲を、0～65535の整数で指定します。

第3関節設定値

第3関節の位置決め許容範囲を、0～65535の整数で指定します。

第4関節設定値

第4関節の位置決め許容範囲を、0～65535の整数で指定します。

第5関節設定値

第5関節の位置決め許容範囲を、0～65535の整数で指定します。省略可能です。垂直6軸型ロボット (Nシリーズを含む)のみに使用します。

第6関節設定値

第6関節の位置決め許容範囲を、0～65535の整数で指定します。省略可能です。垂直6軸型ロボット (Nシリーズを含む)のみに使用します。

第7関節設定値

第7関節の位置決め許容範囲を、0～65535の整数で指定します。省略可能です。ジョイント型7軸ロボットのみに使用します。

第8関節設定値

第8関節の位置決め許容範囲を、0～65535の整数で指定します。省略可能です。付加軸S関節のみに使用します。

第9関節設定値

第9関節の位置決め許容範囲を、0～65535の整数で指定します。省略可能です。付加軸T関節のみに使用します。

*C8, C12シリーズの場合、位置決め許容範囲は0～131070の整数です。

結果

パラメーターを指定しないときは、Fineは、各関節の現在の設定値を表示します。

解説

Fineは、各関節に対し、動作終了時に位置決めを確認する動作命令で、指定位置に対して位置決め終了と判断する許容範囲を指定します。

この位置決め終了判断は、CPUからサーボシステムへの、目標座標パルスの送信が終わってから始まります。サーボのディレイにより、その段階ではロボットは目標位置に到達していません。判断は、各関節の設定された許容範囲まで、毎数ミリ秒ごとに行われます。すべての関節が指定された許容範囲内にくると位置決め終了となります。位置決めが終了すると、プログラム制御は次のステートメントに移りますが、サーボシステムは引き続き目標位置へロボットを制御します。

Fine命令で比較的大きい範囲が指定されると、位置決めは比較的早い段階で終了し、次のステートメントに進みます。

Fineのデフォルト設定は、ロボットタイプにより異なります。詳細は以下のマニュアルを参照してください。
"マニピュレーターマニュアル"

注意

■ サイクルタイムとFine命令

Fine値自体は、アームの加速または減速制御に影響しませんが、細かいFine値が設定されると、サーボがその許容範囲を達成するまで時間 (数ミリ秒) がかかります。システムのサイクルタイムは、その時間だけ遅くなります。アームがFine命令の設定する許容範囲内に位置決めされると、CPUは次の命令を実行します。

■ Fineの初期化 (Motor On, SLock, SFree などの命令による)

下記のいずれかのコマンドが使われると、Fineの設定値は、デフォルト値に初期化されます。
SLock, SFree, Motor

これらのコマンドが実行された後では、かならずFine値を再設定してください。

起こりやすいエラー

Fineによる位置決めが2秒以内に終了しないと、エラーコード4024になります。このエラーは、通常、サーボシステムのバランス調整が必要という意味です。

参照

[Accel](#), [AccelR](#), [AccelS](#), [Arc](#), [Arc3](#), [Go](#), [Jump](#), [Move](#), [Speed](#), [SpeedR](#), [SpeedS](#), [Pulse](#), [FineDist](#), [FineStatus](#)関数

Fine使用例

下記のプログラムは、プログラムファンクションからFineを実行する例と、モニターウィンドウからFineを実行する例です。

```
Function finetest
    Fine 5, 5, 5, 5      '精度を+/-5パルスに設定
    Go P1
    Go P2
Fend

> Fine 10, 10, 10, 10
>
> Fine
10, 10, 10, 10
```

3.10.10 Fine関数

指定した関節のFine設定を返します。

書式

Fine(関節番号)

パラメーター

関節番号

Fine設定を取得する関節番号を整数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

実数値を返します。

参照

[Accel](#), [AccelS](#), [Arc](#), [Arc3](#), [Go](#), [Jump](#), [Move](#), [Speed](#), [SpeedS](#), [Pulse](#)

Fine関数使用例

下記はFine関数を使用したプログラム例です。

```
Function finetst
  Integer a
  a = Fine(1)
Fend
```

3.10.11 FineDist

目標位置の位置決め終了判断範囲の設定、表示をします。設定値の単位はmmです。

書式

(1) FineDist 設定値

(2) FineDist

パラメーター

設定値

位置決め許容範囲は0.001[mm]～10[mm]です。

結果

パラメーターを指定しない場合、FineDistは現在の設定値を表示します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

- FineとFineDistについて

FineとFineDistの違いは、ロボットの動作終了時の位置決め判断の単位です。

- Fineはpulseで位置決め判断値を設定でき、軸ごとに位置決め判断を行います。
- FineDistはmmで位置決め判断値を設定でき、ツール番号0の座標系で位置決め判断を行います。

FineとFineDistの両方を使用することはできません。下記のようにFineとFineDistの2つをプログラムに使用した場合、FineDistで位置決め判断を行います。

(FineとFineDistの順番を逆にした場合は、Fineで位置決め判断を行います。)

```
Function test
  Fine 5, 5, 5, 5
  FineDist 0.1

  Go P1
  Go P2
Fend
```

注意

- FineDistの初期化 (Motor On, SLock, SFreeなどの命令による)

下記のいずれかのコマンドが使われると、FineDistの設定値は、デフォルト値に初期化され、位置決め判断はFineで行います。

SLock, SFree, Motor

これらのコマンドが実行された後では、かならずFineDist値を再設定してください。

起こりやすいエラー

FineDistによる位置決めが2秒以内に終了しないと、エラーコード4024になります。このエラーは、通常、サーボシステムのバランス調整が必要という意味です。

参照

[Accel](#), [AccelR](#), [AccelS](#), [Arc](#), [Arc3](#), [Go](#), [Jump](#), [Move](#), [Speed](#), [SpeedR](#), [SpeedS](#), [Pulse](#), [Fine](#), [FineStatus](#)関数

FineDist使用例

下記のプログラムは、プログラムファンクションからFineDistを実行する例と、モニターウィンドウからFineを実行する例です。

```
Function fineDistttest
    Fine 0.1    '精度を+/-0.1mmに設定
    Go P1
    Go P2
Fend

> FineDist 0.1
>
> FineDist
0.1
```

3.10.12 FineStatus関数

Fineを使用しているか、FineDistを使用しているかを整数値で返します。

書式

FineStatus

戻り値

Fineを使用しているか、FineDistを使用しているかを整数値で返します。

- 0 = Fine使用中
- 1 = FineDist使用中

参照

[Fine](#), [FineDist](#)

FineStatus関数使用例

```
Print FineStatus
```

3.10.13 Fix関数

実数値から整数部分を取り出します。

書式

Fix(数値)

パラメーター

数値

実数値を指定します。

戻り値

パラメーターで設定した実数値の整数の値を返します。

参照

[Int関数](#)

Fix関数使用例

```
>print Fix(1.123)
1
>
```

3.10.14 Flush

バッファをファイルへ書き込みます。

書式

Flush #ファイル番号

パラメーター

ファイル番号

30～63の整数値、または式で指定します。

解説

バッファを現在開いているファイルへ書き込みます。

ROpenコマンドで開いたファイルの場合は使用できません。

Flush使用例

```
Integer fileNum, i

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
For i = 0 To 100
    Print #fileNum, i
Next i
Flush #fileNum
Close #fileNum
```

3.10.15 FmtStr

数値式、または日付/時刻式をフォーマットします。

書式

FmtStr 書式指定する式, 書式指定文字列, 出力文字列変数

パラメーター

書式指定する式

書式指定する数値式、または日付/時刻式を指定します。

日付/時刻式は、"yyyy/mm/dd hh:nn:ss"で指定してください。

書式指定文字列

書式指定文字列を指定します。

出力文字列変数

出力文字列変数を指定します。

解説

書式設定文字列にしたがって、書式設定された文字列を返します。

数値書式指定子

None

フォーマットなしでそのまま番号を表示します。

(0)

桁位置を示す桁表示子です。数値またはゼロを表示します。数値式に含まれる10進数を、書式指定文字列の"0"のところに表示します。それ以外の場合は、その位置に"0"を表示します。数値の桁数が、書式指定文字列の(小数点の前後いずれかにおける)"0"の数より少ない場合は、先頭ないしは後方の"0"をいくつか表示します。もし、数値の小数点の右側の桁数が、書式指定文字列の小数点の右側にある"0"の数より多い場合は、"0"の数に合わせて数値の桁数を丸めます。また、数値の小数点左側の桁数が、書式指定文字列の小数点左側にある"0"の数より多い場合は、その余分な桁をそのまま表示します。

(#)

桁位置を示す桁表示子です。数値を示すか、何も表示しないかのいずれかを行います。数値式の10進数を書式指定文字列の"#"の位置に表示しますが、それ以外の場合は、その位置に何も表示しません。この文字(#)は、0桁表示子のような役目をしますが、数値の桁数が書式指定文字列の小数点の左右にある"#"の数と同じか、それより少ない場合は、先頭ないしは後方の"0"は表示されません。

(.)

小数点位置を表示し、小数点の左右にいくつの桁を表示するか設定します。ローカルによっては、カンマで小数点を示すものもあります。書式指定文字列中、この(.)の左に数字しかない場合は、1より小さい数字が小数点をあたりに始まり、表示されます。先頭のゼロを表示するには、"0"を小数点の左に使います。書式指定されて出力されてくる値で小数点を示す文字は、使用するWindowsが認識する数値フォーマットによります。

(,)

数値を1000で区切る千桁区切り子です。ローカルによっては、ピリオドを用います。小数点の左に4桁以上ある数値で、千ずつ区切るのに使います。標準的な使い方は、書式指定文字列が左右に桁表示子(0 または #)をとまう千桁区切り子(,)を含んでいると指定されます。小数の指定があるなしにかかわらず、小数点のすぐ左に、千桁区切り子が2つ並んでいる場合、または単独で使われている場合、「その数値を1000で割って、必要ならば四捨五入せよ」という書式指定を意味します。例えば、書式指定文字列"##0,,"は、1億を"100"と表示せよ、という意味になります。百万より小さい数値は"0"と表示されます。小数点のすぐ左隣

以外の位置に千桁区切り子が2つ並んで置かれると、単に千桁を区切るために使われます。書式指定された出力値で実際に千桁区切り子として使われる文字は、使用するWindowsが認識する数値フォーマットによります。

日付/時間書式指定子

(:)

時刻区切りです。ローカルによっては、他の文字が用いられることがあります。時刻区切りは、時間値が書式指定されたとき、時、分、秒の値を区切ります。書式指定された出力で実際に使われる文字は、Windowsの設定によって異なります。

(/)

日付区切り子です。ローカルによっては、他の文字が用いられることがあります。日付区切り子は、日付値が書式指定されたとき、日、月、年の値を区切ります。書式指定された出力で実際に使われる文字は、Windowsの設定によって異なります。

c

日付をddddで、時間をttttで、この順序で表示します。日付シリアル番号に分数部分がないと日付だけを、また、時間情報に整数部分がないと時間情報だけを、表示します。

d

日にちを先頭にゼロをつけないで表示します。(1~31)

dd

日にちを先頭にゼロをつけて表示します。(01~31)

ddd

曜日を略して表示します。(Sun~Sat)

dddd

曜日を略さないで表示します。(Sunday~Saturday)

dddddd

Windowsのショートデータ表示設定のフォーマットにしたがって、日付、月、年をすべて表示します。Windowsでは、ショートデータフォーマットの設定は、デフォルトで、m/d/yyとなっています。

dddddd

日付のシリアル数値を、日付、月、年として、Windowsのロングデータ表示設定のフォーマットにしたがって表示します。Windowsでは、ロングデータフォーマットの設定は、デフォルトでmmmm dd, yyyyとなっています。

w

曜日を数値で表示します。(1: 日曜日~7: 土曜日)

ww

1年のうち第何週目かを、数値で表示します。(1~54)

m

月を先頭にゼロをつけない数値で表示します。(1~12)

"h"または"hh"の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、"n"または"nn"を使用してください。

mm

月を先頭にゼロをつけた数値で表示します。(01~12)

"h"または"hh"の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、"n"または"nn"を使用してください。

mmm

月名を略して表示します。(Jan~Dec)

mmmm

月名を略さないで表示します。(January～December)

q

四半期を数値で表示します。(1～4)

y

1年のうち何番目の日かを、数値で表示します。(1～366)

yy

年号を2桁で表示します。(00～99)

yyyy

年号を4桁で表示します。(100～9999)

h

24時間表記の時刻を、先頭にゼロをつけないで表示します。(0～23)

hh

24時間表記の時刻を、先頭にゼロをつけて表示します。(00～23)

n

時刻の分を、先頭にゼロをつけないで表示します。(0～59)

nn

時刻の分を、先頭にゼロをつけて表示します。(00～59)

s

時刻の秒を、先頭にゼロをつけないで表示します。(0～59)

ss

時刻の秒を、先頭にゼロをつけて表示します。(00～59)

ttttt

Windowsが設定する時刻区切り子フォーマットにしたがい、時刻(時, 分, 秒)を表示します。"先頭のゼロ"オプションを選択すると、10:00am/pm以前の時刻に関しては、先頭にゼロをつけた表記で表示します。Windowsでは、時刻フォーマットのデフォルトは、h:nn:ssになっています。

AM/PM

12時間表記で時間を表示し、午前午後をAM/PM (大文字表記)で表示します。

am/pm

12時間表記で時間を表示し、午前午後をam/pm (小文字表記)で表示します。

A/P

12時間表記で時間を表示し、午前午後をA/P (大文字表記)で表示します。

a/p

12時間表記で時間を表示し、午前午後をa/p (小文字表記)で表示します。

AMPM

12時間表記で時間を表示します。午前時刻は、AMを文字列で、午後時刻はPMを文字列で、それぞれWindows内のフォーマット設定にしたがって表示します。AM/PMは、大文字でも小文字でも使えますが、Windowsの設定と指定文字列が一致していなければいけません。Windowsでは、デフォルトは、AM/PMとなっています。

注意

- 数値書式指定子と日付/時間書式指定子の混在

数値書式指定子と日付/時間書式指定子を混在して指定した場合は、エラーとなります。

参照[Left\\$関数](#), [Right\\$関数](#), [Str\\$関数](#)**FmtStr使用例**

```
Function SaveData

    String d$, f$, t$

    ' 月, 日, 時間, 分の形式でファイル名を作成
    d$ = Date$
    t$ = Time$
    d$ = d$ + " " + t$
    FmtStr d$, "mmddhhnn", f$
    f$ = f$ + ".dat"
    WOpen f$ as #30
    Print #30, "data"
    Close #30
Fend
```


3.10.16 FmtStr\$関数

数値式、または日付/時刻式をフォーマットします。

書式

FmtStr\$ (書式指定する式, 書式指定文字列)

パラメーター

書式指定する式

書式指定する数値式または日付/時刻式を指定します。

日付/時刻式は、"yyyy/mm/dd hh:nn:ss"で指定してください。

書式指定文字列

書式指定文字列を指定します。

戻り値

書式指定した文字列を返します。

解説

書式設定文字列にしたがって、書式設定された文字列を返します。

数値書式指定子

None

フォーマットなしでそのまま番号を表示します。

(0)

桁位置を示す桁表示子です。数値またはゼロを表示します。数値式に含まれる10進数を、書式指定文字列の"0"のところに表示します。それ以外の場合は、その位置に"0"を表示します。数値の桁数が、書式指定文字列の(小数点の前後いずれかにおける)"0"の数より少ない場合は、先頭ないしは後方の"0"をいくつか表示します。もし、数値の小数点の右側の桁数が、書式指定文字列の小数点の右側にある"0"の数より多い場合は、"0"の数に合わせて数値の桁数を丸めます。また、数値の小数点左側の桁数が、書式指定文字列の小数点左側にある"0"の数より多い場合は、その余分な桁をそのまま表示します。

(#)

桁位置を示す桁表示子です。数値を示すか、何も表示しないかのいずれかを行います。数値式の10進数を書式指定文字列の"#"の位置に表示しますが、それ以外の場合は、その位置に何も表示しません。この文字(#)は、0桁表示子のような役目をしますが、数値の桁数が書式指定文字列の小数点の左右にある"#"の数と同じか、それより少ない場合は、先頭ないしは後方の"0"は表示されません。

(.)

小数点位置を表示し、小数点の左右にいくつの桁を表示するか設定します。ローカルによっては、カンマで小数点を示すものもあります。書式指定文字列中、この(.)の左に数字しかない場合は、1より小さい数字が小数点をあたりに始まり、表示されます。先頭のゼロを表示するには、"0"を小数点の左に使います。書式指定されて出力されてくる値で小数点を示す文字は、使用するWindowsが認識する数値フォーマットによります。

(,)

数値を1000で区切る千桁区切り子です。ローカルによっては、ピリオドを用います。小数点の左に4桁以上ある数値で、千ずつ区切るのに使います。標準的な使い方は、書式指定文字列が左右に桁表示子(0 または #)をとまう千桁区切り子(,)を含んでいると指定されます。小数の指定があるなしにかかわらず、小数点のすぐ左に、千桁区切り子が2つ並んでいる場合、または単独で使われている場合、「その数値を1000で割って、必要ならば四捨五入せよ」という書式指定を意味します。例えば、書式指定文字列"##0,,"は、1億を"100"と表示せよ、という意味になります。百万より小さい数値は"0"と表示されます。小数点のすぐ左隣

以外の位置に千桁区切り子が2つ並んで置かれると、単に千桁を区切るために使われます。書式指定された出力値で実際に千桁区切り子として使われる文字は、使用するWindowsが認識する数値フォーマットによります。

日付/時間書式指定子

(:)

時刻区切りです。ローカルによっては、他の文字が用いられることがあります。時刻区切りは、時間値が書式指定されたとき、時、分、秒の値を区切ります。書式指定された出力で実際に使われる文字は、Windowsの設定によって異なります。

(/)

日付区切り子です。ローカルによっては、他の文字が用いられることがあります。日付区切り子は、日付値が書式指定されたとき、日、月、年の値を区切ります。書式指定された出力で実際に使われる文字は、Windowsの設定によって異なります。

c

日付をddddで、時間をttttで、この順序で表示します。日付シリアル番号に分数部分がないと日付だけを、また、時間情報に整数部分がないと時間情報だけを、表示します。

d

日にちを先頭にゼロをつけないで表示します。(1~31)

dd

日にちを先頭にゼロをつけて表示します。(01~31)

ddd

曜日を略して表示します。(Sun~Sat)

dddd

曜日を略さないで表示します。(Sunday~Saturday)

dddddd

Windowsのショートデータ表示設定のフォーマットにしたがって、日付、月、年をすべて表示します。Windowsでは、ショートデータフォーマットの設定は、デフォルトで、m/d/yyとなっています。

dddddd

日付のシリアル数値を、日付、月、年として、Windowsのロングデータ表示設定のフォーマットにしたがって表示します。Windowsでは、ロングデータフォーマットの設定は、デフォルトでmmmm dd, yyyyとなっています。

w

曜日を数値で表示します。(1: 日曜日~7: 土曜日)

ww

1年のうち第何週目かを、数値で表示します。(1~54)

m

月を先頭にゼロをつけない数値で表示します。(1~12)

"h"または"hh"の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、"n"または"nn"を使用してください。

mm

月を先頭にゼロをつけた数値で表示します。(01~12)

"h"または"hh"の直後に使用しても、時刻の分としては表示されません。時刻の分の表示は、"n"または"nn"を使用してください。

mmm

月名を略して表示します。(Jan~Dec)

mmmm

月名を略さないで表示します。(January～December)

q

四半期を数値で表示します。(1～4)

y

1年のうち何番目の日かを、数値で表示します。(1～366)

yy

年号を2桁で表示します。(00～99)

yyyy

年号を4桁で表示します。(100～9999)

h

24時間表記の時刻を、先頭にゼロをつけないで表示します。(0～23)

hh

24時間表記の時刻を、先頭にゼロをつけて表示します。(00～23)

n

時刻の分を、先頭にゼロをつけないで表示します。(0～59)

nn

時刻の分を、先頭にゼロをつけて表示します。(00～59)

s

時刻の秒を、先頭にゼロをつけないで表示します。(0～59)

ss

時刻の秒を、先頭にゼロをつけて表示します。(00～59)

ttttt

Windowsが設定する時刻区切り子フォーマットにしたがい、時刻(時, 分, 秒)を表示します。"先頭のゼロ"オプションを選択すると、10:00am/pm以前の時刻に関しては、先頭にゼロをつけた表記で表示します。Windowsでは、時刻フォーマットのデフォルトは、h:nn:ssになっています。

AM/PM

12時間表記で時間を表示し、午前午後をAM/PM (大文字表記)で表示します。

am/pm

12時間表記で時間を表示し、午前午後をam/pm (小文字表記)で表示します。

A/P

12時間表記で時間を表示し、午前午後をA/P (大文字表記)で表示します。

a/p

12時間表記で時間を表示し、午前午後をa/p (小文字表記)で表示します。

AMPM

12時間表記で時間を表示します。午前時刻は、AMを文字列で、午後時刻はPMを文字列で、それぞれWindows内のフォーマット設定にしたがって表示します。AM/PMは、大文字でも小文字でも使えますが、Windowsの設定と指定文字列が一致していなければいけません。Windowsでは、デフォルトは、AM/PMとなっています。

注意

- 数値書式指定子と日付/時間書式指定子の混在

数値書式指定子と日付/時間書式指定子を混在して指定した場合は、エラーとなります。

参照[Left\\$関数](#), [Right\\$関数](#), [Str\\$関数](#)**FmtStr\$使用例**

```
Function SendDateCode  
  
    String d$, f$  
  
    f$ = FmtStr$(10, "000.00")  
    OpenCom #1  
    Print #1, f$  
    CloseCom #1  
Fend
```

3.10.17 FolderExists関数

指定したフォルダーの有無をチェックします。

書式

FolderExists(パス名)

パラメーター

パス名

チェックするフォルダーのパス名を文字列で指定します。ドライブ名を含みます。パスの詳細は、ChDiskを参照してください。

注意

ディスクがPCの場合に実行可能です。

戻り値

- フォルダーが存在する場合: True
- フォルダーが存在しない場合: False

参照

[FileExists関数](#), [MkDir](#)

FolderExists関数使用例

```
If Not FolderExists("c:\TEST") Then  
    Mkdir "c:\TEST"  
EndIf
```

3.10.18 For...Next

For...Next間にある一連のステートメントを、指定の回数だけ、繰り返し実行します。

書式

For 変数名 = 初期値 To 終了値 [Step増分値] ステートメント Next [変数名]

パラメーター

変数名

繰り返しデータを代入する変数名を指定します。この変数は通常、整数値ですが、実数変数として定義することも可能です。

初期値

指定した変数にループの最初に代入する数値を指定します。

終了値

ループの終了を表わす値を指定します。この値がみたされると、For...Nextループは終了し、Next命令の次のステートメントを続けて実行します。

増分値

For...Nextループ中のNextステートメントが実行されるたびに増分する値を指定します。増分値は負の数にも設定できますが、この場合は、初期値>終了値でなければなりません。増分の指定がない場合は、自動的に"1"で増分します。省略可能です。

ステートメント

有効なSPEL+のステートメントであれば、どれでもFor...Nextループ中に挿入することができます。

解説

For...Nextは、ループ中にあるステートメントを指定された回数だけ繰り返します。ループの最初は、Forステートメントで、終わりはNextステートメントです。変数は、ループ内のステートメントが実行される回数を数えます。

初期値の数値は、カウンターの最初の値です。終了値変数と増分値が正しく設定されていれば、負の数値でも設定できます。

終了値は、カウンターの最終値です。この値に達するとループが終了し、プログラム制御はNext命令の次の命令に移行します。

Forステートメントの次のステートメントは、Next命令に達するまで、実行されます。カウンター変数(変数名)は、増分値パラメーターによって指定された値だけ増分していきます。もし増分値が設定されていない場合は、カウンターは、"1"ずつ増減します。

そのあと、カウンター変数(変数名)は、最終値と比較されます。カウンターが最終値よりも少ないか、同じ場合は、For命令の次のステートメントが再実行されます。カウンター変数(変数名)が、最終値よりも大きい場合は、実行は、For...Nextループの外に枝分かれし、Next命令の次の命令に継続します。

注意

■ 負の増分値

負の値を増分値に指定すると、カウンター変数は、ループのたびに減少するので、初期値は終了値よりも大きくしてください。

■ Nextの後の変数名は省略できます

Nextの後の変数名は省略できますが、ネスティングがある場合などは、変数名を記入しておいたほうが構造がわかりやすくなります。

- ループから抜けたときの変数の値は、終了値ではありません

```
Function forsample
  Integer i
  For i = 0 To 3
  Next
  Print i  '4と表示されます
Fend
```

- Exit Forを使用せず、ネスト構造からループを抜けた場合

Exit For命令以外のコマンド(Gosub命令、Goto命令、Call命令など)でループから抜けるプログラムを繰り返し実行すると、エラー2020が発生します。ループの途中で抜ける場合は、Exit For 命令で抜けるようにしてください。

- 空の無限ループ、および無限ループと同様の処理は、できる限りWait命令と併用してください

空のFor...Next、および同様の処理は、システムに影響を与える可能性があるため、できるだけ行わないでください。コントローラーでは、無限ループタスクの検出を行っており、システムに影響が出るとコントローラーが判断した場合、エラー2556 (過剰なループを検出した) が発生することがあります。

ループを必要とする演算や、I/O待ちを行う場合は、Wait命令 (Wait 0.1)などをループ処理内で実行し、CPUの占有を回避してください。

参照

[Do...Loop](#)

For...Next使用例

```
Function fornxt
  Integer counter
  For counter = 1 to 10
    Go Pctr
  Next counter

  For counter = 10 to 1 Step -1
    Go Pctr
  Next counter
Fend
```

3.10.19 FreeFile関数

現在使われていないファイル番号を返し、その番号を予約します。

書式

FreeFile

戻り値

30～63の整数を返します。

参照

[AOpen](#), [BOpen](#), [ROpen](#), [UOpen](#), [WOpen](#), [Close](#)

FreeFile関数使用例

```
Integer fileNum, i, j

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

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


3.10.20 Function...Fend

1つのファンクションは、Functionステートメントで始まりFendステートメントで終わる、実行可能なプログラムの最小単位となります。

書式

Function ファンクション名 [(引数リスト)] [As 型(関数)]

ステートメント

Fend

パラメーター

ファンクション名

Functionで始まりFend命令で終わるステートメントのグループに対する名前を、アルファベット64文字以内で指定します。アンダースコアを用いることも可能です。

引数リスト

呼び出されて、ファンクションに渡される引数の変数のリストです。複数の変数の場合は、カンマで区切ります。省略可能です。

引数の書式は下記のとおりです。

[{ByRef | ByVal}] 変数名[()] As 型(引数)

変数の渡し方は、ByRef/ByValのいずれかを指定します。省略した場合はByValを指定したとみなします。

ByRef

呼び出すファンクションの変数を参照する場合は、ByRefを指定します。この場合、ファンクション内での引数の変更を、呼び出し側の変数に反映させることができます。

ByVal

デフォルト設定です。呼び出すファンクションで参照する変数の値の変更をしない場合は、ByValを指定します。省略可能です。

変数名[()]

引数の変数名で、必須パラメーターです。変数名のつけ方の規則にしたがってください。配列変数を引数として使用する場合は、ByRefを必ず指定した上で、配列を示す"()"中抜き括弧を変数の後ろに付加してください。

As 型(引数)

必須パラメーターです。引数の型を宣言してください。

As 型(関数)

戻り値を取得したい場合に付加するパラメーターです。戻り値の型を宣言してください。

戻り値

ファンクション宣言の最後にAsで指定されたデータ型です (As 型(関数))。宣言がない場合は数値の0が戻り値になります。

解説

Functionステートメントは、SPEL+ステートメントグループの始まりを示します。1つのファンクションの終わりは、Fendステートメントで示します。FunctionとFendステートメントの間にあるステートメントは、すべてそのファンクションの一部と考えます。

FunctionとFendステートメントは、その間にあるすべてのステートメントを含む、それ自体が1つのファンクションである、入れ物のようなものだと考えられます。複数のファンクションを、1つのプログラムファイルに用いることも可能です。

戻り値を使用したい場合、ファンクション名と同名の変数に値を代入してから、関数を終了させてください。

ファンクションの呼び出し方法には以下があります。

1. Xqtの引数として指定する。別タスクとしてファンクションを呼び出します。Xqt は、指定されたファンクションを開始し、すぐに戻ります。
2. Callの引数として指定する。サブルーチンとしてファンクションを呼び出します。Call は指定されたファンクションに、プログラム制御を移します。
3. ファンクション名を記入する。Callによる呼び出しとみなされます。戻り値が文字列の場合はエラーになります。
4. コマンドの引数として指定する。サブルーチンとしてファンクションが実行され、その戻り値が呼び出し元のコマンドへの引数として代入されます。Quitなどタスク識別子を引数とするコマンドにタスク名として指定する場合は、引数や前後の括弧を記載しません。

参照

[Call](#), [Function...Fend](#), [Halt](#), [Quit](#), [Return](#), [Xqt](#)

Function...Fend使用例

[例 1] 下記は、1つのファイルに3つのファンクションを含む例です。task2とtask3は、mainから呼び出され、mainと同時に実行されます。

```
Function main
  Xqt 2, task2 'タスク2を同時に実行
  Xqt 3, task3 'タスク3を同時に実行
  .
  .
  .
Fend

Function task2
  Do
    On 1
    On 2
    Off 1
    Off 2
  Loop
Fend

Function task3
  Do
    On 10
    Wait 1
    Off 10
  Loop
Fend
```

[例 2] 下記は、周辺装置の圧力制御シーケンスを引数として与え、外部装置に送信したときの結果を戻り値として画面表示する場合の、ファンクション例です。

```
Function main
  Integer iResult
  Real Sequence1(200)
  .
  .
  iResult = PressureControl(ByRef Sequence1()) '引数は配列
  .
  Print "Result:", iResult
  .
Fend
```

```
Function PressureControl(ByRef Array1() As Real) As Integer
    .
    (Array1配列に基づいて、周辺装置を圧力制御)
    .
    PressureControl = 3      '戻り値
    .
    .
Fend
```

[例 3]

下記は、ファンクションの呼び出し方による違いの例です。

```
Function main
    ' 戻り値も引数も定義がないファンクション
    Print TaskInfo(SubFunction0, 0)      ' タスク名"SubFunction0"の情報を取得します。
    Print TaskInfo((SubFunction0), 0)    ' サブルーチンとして実行した後、戻り値の"0"が範囲
    ' 外となりエラーになります。

    ' 戻り値が定義されているファンクション
    Print TaskInfo(SubFunction1, 0)      ' タスク名"SubFunction0"の情報を取得します。
    Print TaskInfo((SubFunction1), 0)    ' サブルーチンとして実行した後、戻り値に相当する
    ' タスク番号の情報を取得します。

    ' 戻り値と引数の両方が定義されているファンクション
    Print TaskInfo(SubFunction2, 0)      ' タスク名"SubFunction2"の情報を取得します。
    Print TaskInfo(SubFunction2(0), 0)   ' サブルーチンとして実行した後、戻り値に相当する
    ' タスク番号の情報を取得します。

    ' 戻り値が文字列で定義されているファンクション
    Print TaskInfo(SubFunction1$, 0)     ' タスク名"SubFunction1$"の情報を取得します。
    Print TaskInfo((SubFunction1$), 0)   ' 第一引数のデータ型が異なるためビルドエラーにな
    ' ります。
Fend

Function SubFunction0
    .
Fend

Function SubFunction1 As Integer
    .
    SubFunction1 = 1
Fend

Function SubFunction2(i As Integer) As Integer
    .
    SubFunction2 = 1
Fend

Function SubFunction1$ As String
    .
    SubFunction1$ = "main"
Fend
```

3.11 G

3.11.1 GUIVer\$

接続中のEpson RC+のバージョンを取得します。

書式

GUIVer\$

パラメーター

なし

参照

なし

GUIVer\$ 使用例

```
String strVer$  
  
strVer$ = GUIVer$  
Print "RC+ Version = " + strVer$
```

3.11.2 GetIODef

入力, 出力, メモリーI/Oのビット, バイト, ワードのI/Oラベルとコメントを取得します。

書式

GetIODef 種別, インデックス, ByRef ラベル, ByRef コメント

パラメーター

- 種別 I/Oタイプを表す整数値
 - 1: InputBit
 - 2: InputByte
 - 3: InputWord
 - 4: OutputBit
 - 5: OutputByte
 - 6: OutputWord
 - 7: MemoryBit
 - 8: MemoryByte
 - 9: MemoryWord
 - 10: InputReal
 - 11: OutputReal
- インデックス: ビット、あるいはポート番号を表す整数値
- ラベル: 指定した種別, インデックスに該当するラベルを取得する文字列変数を指定します。
- コメント: 指定した種別, インデックスに該当するコメントを取得する文字列変数を指定します。

解説

各I/Oポイントのラベルやコメントを取得します。

参照

[IONumber関数](#), [IODef関数](#), [IOLabel\\$関数](#), [SetIODef](#)

GetIODef使用例

```
String strLabel$
String strDescript$

GetIODef 1, 0, ByRef strLabel$, ByRef strDescript$
Print "InputBit 0: Version = " + strLabel$
Print "InputBit 0: Description = " + strDescript$
```

3.11.3 GetProjectInfo

Epson RC+で開いているプロジェクト名とプロジェクトフォルダパスを取得します。

書式

GetProjectInfo ByRef プロジェクト名, ByRef プロジェクトフォルダパス

パラメーター

- プロジェクト名: 現在開いているプロジェクト名を取得する文字列変数を指定します。
- プロジェクトフォルダパス: プロジェクトフォルダのパスを取得する文字列変数を指定します。

解説

現在開いているプロジェクト情報を参照したい場合、本コマンドを使用します。

GetProjectInfo使用例

```
String strName$
String strPath$

GetProjectInfo ByRef strName$, ByRef strPath$
Print "Project name = " + strName$
Print "Project folder path = " + strPath$
```

3.11.4 GetSetNet

全TCP/IPポートのパラメーターを取得します。

書式

GetSetNet ByRef 返却する文字列配列変数

パラメーター

- 返却する文字列配列変数: 16個のTCP/IPポートのパラメーターを格納する文字列配列を指定します。

解説

16個のTCP/IPポートの設定を取得する際、本コマンドを使用します。

設定値の詳細は、以下を参照してください。

[SetNet](#)

参照

[SetNet](#)

GetSetNet使用例

```
String Info$(15)
Integer i

GetSetNet ByRef Info$()
For i = 0 To 15
    Print "i = " + Str$(i) + ": " + Info$(i)
Next
```

3.11.5 GetRobotInsideBox関数

進入検出エリアに進入しているロボットを返します。

書式

GetRobotInsideBox(エリア番号)

パラメーター

エリア番号

状態を返す進入検出エリアの番号 (1～15の整数)を指定します。

戻り値

エリア番号で指定された進入検出エリアに進入しているロボットをビットで返します。

ビット0がロボット1で以降順番にビット15がロボット16です。

ロボットで進入検出エリアが設定されていない場合は、該当ビットは常に0になります。

例えばロボット1とロボット3がエリアに進入している場合はビット0とビット2がOnするので5を返します。

参照

[Box, InsideBox関数](#)

GetRobotInsideBox関数使用例

下記はGetRobotInsideBox関数を使用したプログラムです。

進入検出エリアに1台もロボットが進入していない状態を待ちます。

```
Function WaitNoBox  
  
    Wait GetRobotInsideBox(1) = 0
```

進入検出エリアにロボット2だけが進入している状態を待ちます。

```
Function WaitInBoxRobot2  
  
    Wait GetRobotInsideBox(1) = &H2
```

下記のプログラムは、動作命令の並列処理内で使用する例です。動作実行中に特定の進入検出エリアに進入した時にI/Oをオンします。コントローラーに接続しているロボットが1台のケースです。

```
Function Main  
    Motor On  
    Power High  
    Speed 30; Accel 30, 30  
  
    Go P1 !D0; Wait GetRobotInsideBox(1) = 1; On 1!  
  
End
```

注意

D0を必ず記述してください。

3.11.6 GetRobotInsidePlane関数

進入検出平面に進入しているロボットを返します。

書式

GetRobotInsidePlane(平面番号)

パラメーター

平面番号

状態を返す進入検出平面の番号(1～15の整数)を指定します。

戻り値

平面番号で指定された進入検出平面に進入しているロボットをビットで返します。

ビット0がロボット1で以降順番にビット15がロボット16です。

ロボットで進入検出平面が設定されていない場合は、該当ビットは常に0になります。

例えば、ロボット1とロボット3が平面に進入している場合はビット0とビット2がOnするので5を返します。

参照

[InsidePlane関数](#), [Plane](#)

GetRobotInsidePlane関数使用例

下記はGetRobotInsidePlane関数を使用したプログラムです。

進入検出平面に1台もロボットが進入していない状態を待ちます。

```
Function WaitNoPlane
    Wait GetRobotInsidePlane(1) = 0
```

進入検出平面にロボット2だけが進入している状態を待ちます。

```
Function WaitInPlaneRobot2
    Wait GetRobotInsidePlane(1) = &H2
```

下記のプログラムは、動作命令の並列処理内で使用する例です。動作実行中に特定の進入検出平面に進入した時にI/Oをオンします。コントローラーに接続しているロボットが1台のケースです。

```
Function Main
    Motor On
    Power High
    Speed 30; Accel 30, 30

    Go P1 !D0; Wait GetRobotInsidePlane(1) = 1; On 1!

End
```

注意

D0を必ず記述してください。

3.11.7 Global

グローバル変数の宣言をします。グローバル変数はどこからでもアクセスできます。

書式

Global [Preserve|Static|NonStatic] データ型 変数名 [(配列変数の最大要素番号)] [, 変数名[(配列変数の最大要素番号)] ,...]

パラメーター

Preserve

Preserveを指定すると、変数の値を保持します。プロジェクトを変更すると、この値はクリアします。省略可能です。

データ型

Boolean, Byte, Double, Int32, Integer, Long, Real, Short, String, UByte, UInt32, UShortのいずれかのデータ型を指定します。

変数名

変数名を32文字以内で指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。

グローバル変数の場合、全要素数は最大100,000です。ただし、String型は最大10,000です。

バックアップ変数 (Global Preserve)の場合、全要素数は最大4,000です。ただし、String型は最大400です。

各最大要素番号は、全要素数が最大値を超えない範囲で指定します。

Static

Staticを指定すると、コントローラー起動後の初回のメインファンクション実行時に初期化されます。

NonStatic

NonStaticを指定すると、メインファンクション実行のたびに初期化されます。

解説

グローバル変数は、同じプロジェクト内であれば複数のファイルで共有できる変数です。Preserveオプションで宣言されない限り、Runウィンドウやオペレーターウィンドウからファンクションが起動されるたびにクリアされます。グローバル変数の初期化については、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - 変数の初期化"

Preserveオプションで宣言されると、コントローラーの電源を落としても値が保持されます。

保存したグローバル変数は、オプションのRC+ APIオプションで用いることも可能です。

グローバル変数名は、プログラム中で見分けやすくするため、下記例のように、接頭部 "g_" で始めてください。

Preserve, Static, NonStaticのいずれも宣言がない場合は、コントローラーの環境設定「ファンクション起動時にグローバル変数を初期化」に従います。

```
Global Long g_PartsCount
```

参照

[Boolean](#), [Byte](#), [Double](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Global使用例

下記は、異なる2つのプログラムファイルです。最初のファイルでは、いくつかのグローバル変数を定義し初期化しています。次のファイルで、それらのグローバル変数を使っています。

FILE1 (MAIN.PRG)

```
Global Integer g_Status
Global Real g_MaxValue

Function Main

    g_Status = 10
    g_MaxValue = 1.1
    .
    .
Fend
```

FILE2 (TEST.PRG)

```
Function Test

    Print "status1 =" , g_Status
    Print "MaxValue =" , g_MaxValue
    .
    .
Fend
```

3.11.8 Go

アームを、現在位置から指定位置までPTP動作で動かします。

書式

Go目標座標 [CP] [LJM [姿勢フラグ選択]] [Till | Find] [! 並列処理!] [SYNC]

パラメーター

目標座標

ポイントデータで目標位置を指定します。

CP

パスモーションを指定します。省略可能です。

LJM

目標座標をLJM関数で変換します。省略可能です。

姿勢フラグ選択

LJM関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

! 並列処理!

I/Oその他のコマンドを動作中に実行させたいときに、並列処理ステートメントを付け加えることができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

Goは、ロボットアームのすべての関節を、同時にPTP動作で動かします。下記は、Go命令に対する目標座標を指定する例です。

- 特定のポイント番号で指定する。例: Go P1
- 移動先を具体的な座標値で指定する。例: Go XY(50, 400, 0, 0)
- ポイント番号+オフセット値で指定する。例: Go P1 +X(50)
- ポイント番号+それと異なる座標値を明示して指定する。例: Go P1 :X(50)

動作の経路は、各関節がそれぞれ現在のポイントから目標座標までを補間するため、軌跡は予測できません。周辺装置との干渉に十分注意してください。

Go命令の動作速度は、Speed 命令で設定します。加減速度は、Accel 命令で設定します。

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

LJMパラメーターをつけた場合には、現在位置を参照位置として、目標位置をLJM関数で変換した位置に動作します。

Go LJM(P1, Here,1)を、Go P1 LJM 1と簡略して記述できます。

このとき元のポイントデータP1は変更されません。

LJMパラメーターは垂直6軸型ロボット (Nシリーズを含む)、およびRSシリーズロボットで有効です。

姿勢フラグ選択をデフォルト値で使用する場合は、省略が可能です。

Go P1 LJM

注意

■ Go と Moveの違い

Move と Goは、ともにロボットアームを動作させる命令です。両者の間の大きな違いは、GoがPTP動作させるのに対し、Moveは直線軌道でアームを移動させることです。目標ポイントに到着したときのアームの姿勢が重要であるときはGo 命令を用いますが、動作中のアームの軌道を制御することがより重要である場合は、Move 命令を使います。

■ Go と Jumpの違い

JumpとGoは、ともにロボットアームをPTP動作で移動させる命令です。しかし、Jumpには、Goにない機能が1つあります。Jumpでは、ロボットのハンドをまずLimZ値まで持ち上げてから、アームを水平移動させ、目標座標の上空にきたところで下降移動させる、という動きをさせることです。この動きのメリットは、障害物をより確実に避けることができる点と、さらに重要なのは、吸着, 配置動作による作業のサイクルタイムの向上にあります。

■ Goに適切な速度と加減速度指示を与える

Go 命令の動作速度と加減速度の設定は、Speed とAccel命令で行います。非常に重要なことですが、SpeedとAccel命令はGo命令と同じく、PTP動作に対する設定を行います。直線および円弧補間動作の速度, 加減速度の設定は、SpeedSとAccelS命令で行います。

■ Till 修飾子を用いるGoの用法

Till修飾子を用いることで、Go命令で指定された目標座標に到着する前に、そこに至る通過点で、ロボットを減速停止する条件を設定することができます。もしTill条件が成立しない場合は、ロボットはそのまま目標座標まで移動します。Tillを用いたGoの用法には、下記の2通りがあります。

• (1) GoとTill 修飾子

現在のTill条件が成立したかどうかチェックします。成立すれば、Goが指定した動作の完了を待たずにロボットを途中通過点で減速停止させることで、コマンドの実行は完了します。

• (2) GoとTill 修飾子、Sw (入力ビット番号) 修飾子、入力条件

この用法では、あらかじめ設定されたTillの既存の設定条件を用いるのではなく、Go命令と同じ行でTill条件を新たに設定することができます。設定する条件は、入力1つをチェックするというものですが、そのためにはSw 命令が必要になります。入力の状態がオンかオフかをチェックし、設定した条件に応じてアームを停止させることができます。この機能は、入力条件が成立すると動作が停止させられる「割込み」とほとんど同じ働きをします。もし入力条件がロボットの動作中に1度も成立しなければ、アームは指定された目標座標位置に到達します。

■ Find修飾子を用いるGoの用法

Find修飾子を用いることで、Go命令による動作中にロボットに1つの位置を記録するよう条件設定することができます。Findを用いたGoの用法には、下記の2通りがあります。

• (1) GoとFind 修飾子

現在のFind条件が成立したかどうかチェックします。成立すれば、現在の位置を特別なポイント、FindPosに保存します。

- (2) GoとFind 修飾子, Sw(入力ビット番号) 修飾子, 入力条件

この用法では、あらかじめ設定されたFindの既存の設定条件を用いるのではなく、Go命令と同じ行で新たにFind条件を設定することができます。設定条件は、1つの入力をチェックすることです。そのためにはSw命令が必要になります。入力の状態がオンかオフかをチェックし、現在の位置を特別なポイント、FindPosに保存します。

- Go 命令では停止前に必ず減速します

Go命令では、アームが動作の目標座標に停止する前に、必ず減速します。

起こりやすいエラー

- 可動範囲外にロボットを動作させようとしたとき

目標座標を直接座標で設定する場合、その座標位置がロボットの可動範囲内にあるか、必ず確認してください。ロボットの可動範囲内外が指定されていると、エラーが発生します。

参照

[!...! 並列処理](#), [Accel](#), [Find](#), [Jump](#), [Move](#), [Pass](#), [P# \(2. ポイント指定\)](#), [Pulse](#), [Speed](#), [Sw関数](#), [Till](#)

Go使用例

下記は、P0 からP10 にPTP 動作する簡単な例です。プログラム後半で、アームは、入力ビット2がオンになるまで、P2に向かってPTP 動作します。Goの途中で、入力ビット2がオンになると、アームはP2到着を待たずに減速停止し、次の命令を実行します。

```
Function sample
Integer i

Home
Go P0
Go P1
For i = 1 to 10
    Go P(i)
Next i
Go P2 Till Sw(2) = On
If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
EndIf
Fend
```

コマンドウィンドウからの操作例です。

>Go Here +X(50)	'現在の位置からX方向に50mm移動
>Go P1	'P1に移動
>Go P1 :U(30)	'P1ただしU=30の位置へ移動
>Go P1 /L	'左腕姿勢でP1へ移動
>Go XY(50, 450, 0, 30)	'X=50, Y=450, Z=0, U=30の位置へ移動

[その他の例]

Till Sw(1) = Off And Sw(2) = On	'入力1と2をTill条件として指定
Go P1 Till	'あらかじめ定義したTill条件を満たした場合は停止
Go P2 Till Sw(2) = On	'入力ビット2がオンになった場合は停止
Go P3 Till	'あらかじめ定義したTill条件を満たした場合は停止

3.11.9 GoSub...Return

GoSubは、プログラム制御をサブルーチンに移行します。サブルーチンの実行が終わると、制御はGoSub命令の次の行に戻ります。

書式

GoSub {ラベル}

{ラベル;} ステートメント

Return

パラメーター

ラベル

移行先をラベルで指定します。そのラベルがある行にプログラム実行は移行します。ラベル名は、32文字以内で指定してください。ただし、最初の文字には数字は使えません。必ずアルファベット文字を用いてください。

解説

GoSub命令は、指定のラベルにプログラム制御を移します。プログラムは移行先のステートメントを実行し、Return命令まで、そのまま移行先の行を実行していきます。GoSub命令は、サブルーチンの実行を行ってからReturnによって、GoSub命令の次の行に戻ります。サブルーチンは必ずReturnで終わらせてください。

起こりやすいエラー

- 存在しないステートメントへの移行

GoSub命令が存在しないラベルを移行先に指定すると、エラー3108が発生します。

- GoSubがないのにReturnが使われているとき

Return命令は、GoSub命令を実行したもとのプログラムへの復帰を行います。もし、GoSubがないのにReturnが使われているとエラー2383が発生します。GoSubなしでReturnが使われても、どこに復帰してよいかわからないので意味を持ちません。

参照

[GoTo](#), [OnErr](#), [Return](#)

GoSub使用例

下記は、GoSub命令を使って指定ラベルに移り、I/O命令をいくつか実行して復帰するという簡単な例です。

```
Function main
  Integer var1, var2

  GoSub checkio 'ラベルを使用してGoSubを実行
On 1
On 2
Exit Function

checkio: 'サブルーチンの開始位置
  var1 = In(0)
  var2 = In(1)
  If var1 = 1 And var2 = 1 Then
    On 1
  Else
    Off 1
  EndIf
```


Return	'サブルーチンの終了位置
Fend	

3.11.10 GoTo

GoTo命令は、プログラム制御を指定のラベルに移します。

書式

GoTo { ラベル }

パラメーター

ラベル

ラベルのある行にプログラム実行が移行します。ラベルは、32文字までで指定しますが、最初の文字に数字は使えません。必ずアルファベットを使用してください。

解説

GoTo命令は、指定のラベルにプログラム制御を移行します。プログラムは、移行先のステートメント行を実行し、その行以降を実行していきます。

注意

- GoToを多用しすぎる場合

1つのプログラム中で、GoTo命令を使い過ぎると、プログラムがわかりにくくなるので注意してください。通常は、できるだけGoTo命令は使用しないでください。実際には、どうしても使用をしなければならないGoToがありますが、ソースコードをあちこちに移行するようなGoToステートメントの使い方は、エラーやその他の問題の原因になります。

参照

[GoSub...Return, OnErr](#)

GoTo使用例

下記は、GoTo命令を使って、制御を行ラベルに移す簡単なプログラムの例です。

```
Function main

    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print "Input 1 was On, continuing cycle"
    .
    .
    Exit Function

mainAbort:
    Print "Input 1 was OFF, cycle aborted!"
Fend
```

3.12 H

3.12.1 Halt

指定された実行中のタスクを一時停止します。

書式

Halt タスク識別子

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。

タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1から32
- バックグラウンドタスク: 65から80
- トラップタスク: 257から267

解説

Haltは、タスク名またはタスク番号で指定された実行中のタスクを一時停止します。

停止したところから再開するときは、Resumeを使います。タスクの実行を完全に終了するときは、Quitを使います。タスクの状態を表示するには、Epson RC+ のツールバー上のタスクマネージャーのアイコンをクリックし、タスクマネージャーを起動します。

Haltは、指定されたタスクがNoPauseタスク, NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク), トラップタスク, バックグラウンドタスクの場合も、タスクを一時停止します。ただし、これらのタスクを一時停止させるには十分な検討が必要になります。一般的なケースでは特殊なタスクに対してHaltを実行しないことを推奨します。

参照

[Quit](#), [Resume](#), [Xqt](#)

Halt使用例

下記は、"flicker"という名前のファンクションをXqtで起動した後、Haltで一時停止し、Resumeで再開する例です。

```
Function main
  Xqt flicker          'flickerタスクを実行

  Do
    Wait 3             'flickeタスクを3秒間実行
    Halt flicker

    Wait 3             'flickerタスクを3秒間一時停止
    Resume flicker

  Loop
Fend

Function flicker
  Do
    On 1
```

```
Wait 0.2
Off 1
Wait 0.2
Loop
Fend
```

3.12.2 Hand

ポイントのハンド(腕系)姿勢を設定します。

書式

(1) Hand ポイント指定 [, Lefty | Righty]

(2) Hand

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

Lefty | Righty

ハンド(腕系)姿勢を指定します。

結果

2つのパラメーターが両方省略されると、ロボットの現在位置のハンド(腕系)姿勢を表示します。

Lefty | Rightyパラメーターが省略されると、指定したポイントのハンド(腕系)姿勢を表示します。

注意

Handコマンドは、ハンド(エンドエフェクター)を制御するコマンドではありません。

- Hand (本コマンド): マニピュレーターの腕が右腕系か左腕系かを指定します。|
- Hand_On, Hand_Off, Hand_On関数, Hand_Off関数, Hand_TW関数, Hand_Def関数, Hand_Type関数, Hand_Label\$関数, Hand_Number関数: マニピュレーター先端に取りつける、ハンド(エンドエフェクター)を制御します。

混同しないように注意してください。

ハンド制御コマンドについては、以下のマニュアルを参照してください。

"Hand機能"

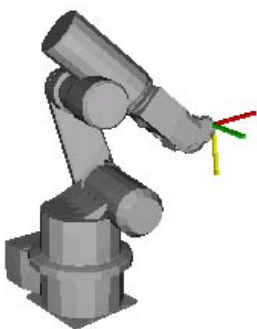
参照

[Elbow](#), [Hand関数](#), [J4Flag](#), [J6Flag](#), [Wrist](#), [J1Flag](#), [J2Flag](#)

Hand使用例

```
Hand P0, Lefty
Hand pick, Righty
Hand P(myPoint), myHand
```

```
P1 = -364.474, 120.952, 469.384, 72.414, 1.125, -79.991
```



Hand P1, Righty
Go P1



Hand P1, Lefty
Go P1

3.12.3 Hand関数

ポイントのハンド(腕系)姿勢を返します。

書式

Hand [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のハンド(腕系)姿勢を返します。

戻り値

- 1: Righty (/R)
- 2: Lefty (/L)

注意

Hand関数は、ハンド(エンドエフェクター)を制御する関数ではありません。

- Hand関数 (本関数): マニピュレーターの腕が右腕系か左腕系かを指定します。
- Hand_On, Hand_Off, Hand_On関数, Hand_Off関数, Hand_TW関数, Hand_Def関数, Hand_Type関数, Hand_Label\$関数, Hand_Number関数: マニピュレーター先端に取りつける、ハンド(エンドエフェクター)を制御します。

混同しないように注意してください。

ハンド制御コマンドについては、以下のマニュアルを参照してください。

"Hand機能"

参照

[Elbow](#), [Wrist](#), [J4Flag](#), [J6Flag](#), [J1Flag](#), [J2Flag](#)

Hand関数使用例

```
Print Hand(pick)
Print Hand(P1)
Print Hand
Print Hand(P1 + P2)
```

3.12.4 HealthCalcPeriod

部品消耗管理情報における「残月数」を演算する期間の設定と表示をします。

書式

(1) HealthCalcPeriod 期間設定値

(2) HealthCalcPeriod

パラメーター

期間設定値

演算する期間を整数値(1~7)で指定します。(単位: 日) デフォルト値は"7"です。

結果

パラメーターを省略すると、現在のHealthCalcPeriod設定値を表示します。

解説

部品消耗管理情報の残月数は過去の動作状況により定期的に演算されます。HealthCalcPeriodコマンドは、この演算に用いる動作期間の設定と表示をします。

期間を長くすると、期間内の変動によるばらつきの影響を抑えた「残月数」が演算されます。ただし、動作や速度を変更したときに、「残月数」が正しく表示されるまでの時間が長くなります。

HealthCalcPeriodの設定値は、実行したコントローラーで制御されるすべてのロボット, 関節, 部品に対して適用されます。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。T/VTシリーズでは常に1(日)が使用されます。

- 設定期間

HealthCalcPeriodコマンドで設定する期間は、コントローラーを起動している期間です。

実時間と異なることに注意してください。

- 「残り月数」および「消耗率」のクリア時の演算

Epson RC+ の「部品消耗管理情報」、HealthCtrlResetまたはHealthRBResetで「残り月数」および「消耗率」をクリアした後に最初に設定期間に達するまでの間は、HealthCalcPeriodの設定値にかかわらず、1日ごとに残月数の演算を行います。この期間は残月数のばらつきが大きくなります。

参照

[HealthCalcPeriod関数](#), [HealthCtrlInfo](#), [HealthRBInfo](#), [HealthCtrlReset](#), [HealthRBReset](#)

HealthCalcPeriod使用例

```
> HealthCalcPeriod 3
> HealthCalcPeriod
3
```


3.12.5 HealthCalcPeriod関数

現在の設定されている部品消耗管理情報における「残月数」を演算する期間を返します。

書式

HealthCalcPeriod

戻り値

演算する期間を整数値で返します。(単位: 日)

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

参照

[HealthCalcPeriod](#)

HealthCalcPeriod関数使用例

演算期間を表示する例です

```
Print "period is", HealthCalcPeriod
```

3.12.6 HealthCtrlAlarmOn関数

コントローラーに関する指定した種別の部品の部品消耗アラーム状態を返す関数です。

書式

HealthCtrlAlarmOn(部品種別)

パラメーター

部品種別

アラーム状態を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

指定した部品が部品消耗アラームを発生している場合はTrue、発生していない場合はFalseを返します。

部品消耗アラームは、HealthRateCtrlInfoで取得される部品消耗率が100%を超えた場合に発生します。

参照

[HealthCtrlInfo](#), [HealthRateCtrlInfo](#)関数

HealthCtrlAlarmOn関数使用例

コントローラーのバッテリー部品消耗アラームが発生しているかを判断する例です。

```
Function PrintAlarm
  If HealthCtrlAlarmOn(HEALTH_CONTROLLER_TYPE_BATTERY) = True Then
    Print "Controller Battery NG"
  Else
    Print "Controller Battery OK"
  EndIf
Fend
```

3.12.7 HealthCtrlInfo

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を表示します。

書式

HealthCtrlInfo 部品種別

パラメーター

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

解説

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を表示します。

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーをHealthCalcPeriodで設定された期間稼働することによって取得される消費率の変化量に基づいて演算されます。

注意

コントローラーをHealthCalcPeriodで設定された期間稼働することによって取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間がHealthCalcPeriodで設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーをHealthCalcPeriodで設定された期間の2倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

[HealthCtrlAlarmOn関数](#), [HealthRateCtrlInfo関数](#)

HealthCtrlInfo使用例

コントローラーのバッテリー交換推奨時期までの残り月数を表示する例です。

```
> HealthCtrlInfo HEALTH_CONTROLLER_TYPE_BATTERY
BATTERY                240.000
>
```

3.12.8 HealthCtrlInfo関数

コントローラーに関する指定した種別の部品の交換推奨時期までの残り月数を返す関数です。

書式

HealthCtrlInfo(部品種別)

パラメーター

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

交換推奨時期までの残り月数を実数で返します (単位: 月)。

解説

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて演算されます。

注意

コントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間がHealthCalcPeriodで設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーをHealthCalcPeriodで設定された期間の2倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照

[HealthCtrlAlarmOn関数](#), [HealthRateCtrlInfo関数](#)

HealthCtrlInfo関数使用例

交換推奨時期までの残り月数が1か月以下になったらアラームを出力する例です。

```
Function AlarmCheck
  Real month

  month = HealthCtrlInfo(HEALTH_CONTROLLER_TYPE_BATTERY)
  If month < 1 Then
    Print "Alarm ON"
  EndIf
Fend
```

3.12.9 HealthCtrlRateOffset

指定した部品種別の消耗率に対してオフセットを設定します。

書式

HealthCtrlRateOffset部品種別, オフセット値

パラメーター

部品種別

コントローラーに関する部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

オフセット値

消耗率に対してオフセットする整数値を指定します。(単位: %)

解説

指定した部品種別に対して、消耗率のオフセットを設定します。

参照

[HealthRBAAlarmOn関数](#), [HealthRateRBInfo関数](#), [HealthRBInfo](#)

HealthCtrlRateOffset使用例

コントローラーのバッテリーの消耗率に対して10%加算する例です。

```
> HealthCtrlRateOffset HEALTH_CONTROLLER_TYPE_BATTERY,10
>
```

3.12.10 HealthCtrlReset

指定した部品種別の交換推奨時期までの残り月数、および消耗率をクリアします。

書式

HealthCtrlReset 部品種別

パラメーター

部品種別

コントローラーに関する部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

解説

指定した部品種別に対して、交換推奨時期までの残り月数、および消耗率をクリアします。警告も同時に解除します。

参照

[HealthCtrlAlarmOn関数](#), [HealthRateCtrlInfo関数](#), [HealthCtrlInfo](#)

HealthCtrlReset使用例

```
> HealthCtrlReset HEALTH_CONTROLLER_TYPE_BATTERY
>
```

3.12.11 HealthCtrlWarningEnable

コントローラーに関する指定した種別の部品の部品消耗アラームの通知を有効または無効にします。

書式

HealthCtrlWarningEnable 部品種別[, On/Off]

パラメーター

部品種別

コントローラーの部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

On/Off

- On: 部品消耗アラームの通知を有効にします。 - Off: 部品消耗アラームの通知を無効にします。

結果

On/Offパラメーターを省略すると、現在のOn/Off設定値が表示されます。

解説

指定した種別の部品の部品消耗アラームが発生した場合、部品消耗アラーム通知を行うかどうかを設定します。

注意

指定した部品の部品消耗アラームの通知を無効にした場合、交換推奨時期を経過しても部品消耗アラームが通知されません。本コマンドによる設定には十分注意して実行してください。

参照

[HealthCtrlAlarmOn関数](#)

HealthCtrlWarningEnable使用例

コントローラーの、バッテリーの部品消耗アラーム通知を無効にする例です。

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY, Off
```

コントローラーの、バッテリーの部品消耗アラーム通知設定を表示する例です。

```
> HealthCtrlWarningEnable HEALTH_CONTROLLER_TYPE_BATTERY
BATTERY Off
>
```

3.12.12 HealthCtrlWarningEnable関数

コントローラーに関する指定した種別の部品の部品消耗アラームの通知の設定状態を返す関数です。

書式

HealthCtrlWarningEnable(部品種別)

パラメーター

部品種別

コントローラーの部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

部品消耗アラームの設定値を整数値で返します。

- 1: On
- 0: Off

参照

[HealthCtrlAlarmOn関数](#)

HealthCtrlWarningEnable関数使用例

コントローラーのバッテリーの部品消耗アラームの通知設定状態を表示する例です。

```
Print HealthCtrlWarningEnable(HEALTH_CONTROLLER_TYPE_BATTERY )
```


3.12.13 HealthRateCtrlInfo関数

コントローラーに関する指定した種別の部品の消耗率を返す関数です。

書式

HealthRateCtrlInfo(部品種別)

パラメーター

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (1)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_CONTROLLER_TYPE_BATTERY	1	バッテリーを指定します。

戻り値

交換推奨時期を100%としたときの現在の部品消耗率を実数で返します(単位: %)。

解説

実稼働状況データに基づいて部品消耗率を演算します。

注意

交換推奨時期は、統計に基づいた部品交換を推奨する時期です。部品消耗率が100 %に達する前に、交換が必要になる場合もあります。また、100 %に達した場合でも、即座に使用できなくなるわけではありません。ただし、100%以降は故障の可能性が増加するため、早めの交換を推奨します。

参照

[HealthCtrlAlarmOn関数](#), [HealthCtrlInfo](#)

HealthRateCtrlInfo関数使用例

コントローラーバッテリーの部品消耗率が90 %以上になったらアラームを出力する例です。

```
Function AlarmCheck
  Real HealthRate

  HealthRate = HealthRateCtrlInfo (HEALTH_CONTROLLER_TYPE_BATTERY)
  If HealthRate > 90 Then
    Print "Alarm ON"
  EndIf
Fend
```

3.12.14 HealthRateRBInfo関数

ロボットに関する指定した種別の部品の消耗率を返す関数です。

書式

HealthRateRBInfo(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号

部品消耗率を返すロボット番号を整数値 (1-16)で指定します。

部品種別

部品消耗率を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

部品消耗率を返す関節を整数値 (1-9)で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

交換推奨時期を100 %としたときの、現在の部品消耗率を実数で返します (単位: %)。

ロボットの関節が指定した部品を使用していない場合には、-1を返します。

解説

実稼働状況データに基づいて部品消耗率を演算します。

注意

交換推奨時期は、統計に基づいた部品交換を推奨する時期です。部品消耗率が100 %に達する前に、交換が必要になる場合もあります。また、100 %に達した場合でも、即座に使用できなくなるわけではありません。ただし、100%以降は故障の可能性が増加するため、早めの交換を推奨します。

参照

[HealthRBAlarmOn関数](#), [HealthRBInfo](#)

HealthRateRBInfo関数使用例

ロボット1の第3関節の減速機の部品消耗率が90%以上になったらアラームを出力する例です。

```
Function AlarmCheck
Real HealthRate

HealthRate = HealthRateRBInfo(1, HEALTH_ROBOT_TYPE_GEAR, 3)
If HealthRate > 90 Then
```

```
Print "Alarm ON"  
EndIf  
Fend
```

3.12.15 HealthRBAlarmOn関数

ロボットに関する指定した種別の部品の部品消耗アラーム状態を返す関数です。

書式

HealthRBAlarmOn(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号

アラーム状態を返すロボット番号を整数値 (1-16) で指定します。

部品種別

アラーム状態を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

アラーム状態を返す関節を整数値 (1-9) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。本コマンドは、付加軸に対しては使用できません。

戻り値

指定した部品が部品消耗アラームを発生している場合はTrue、発生していない場合はFalseを返します。

部品消耗アラームは、HealthRateRBInfoで取得される部品消耗率が100%を超えた場合に発生します。

ロボットの関節が指定した部品を使用していない場合には、False(0)を返します。

参照

[HealthRBInfo](#), [HealthRateRBInfo関数](#)

HealthRBAlarmOn関数使用例

ロボット1の第3関節のグリス部品消耗アラームが発生しているかを判断する例です。

```
Function PrintAlarm4
If HealthRBAlarmOn(1, HEALTH_ROBOT_TYPE_GREASE, 3) = True Then
Print "Robot1 Joint3 Grease NG"
Else
Print "Robot1 Joint3 Grease OK"
EndIf
Fend
```

3.12.16 HealthRBAnalysis関数

あるサイクルのロボット動作に関して、指定した部品種別の使用可能月数を返す関数です。

書式

HealthRBAnalysis(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号

ロボット番号を整数値 (1-16)で指定します。

部品種別

ロボットに関する部品を整数値 (2-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

関節を整数値 (1-6)で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

使用可能月数を実数で返します。

指定の部品種別が、指定した関節で未実装の場合には、-1を返します。

解説

あるサイクルのロボット動作で、指定した部品種別の使用可能月数をシミュレートします。部品が新品の状態で24時間連続稼働した場合に何カ月使用できるか算出します。過去の使用状況は加味されません。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む)では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#)

HealthRBAnalysis関数使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
```

```
Motor Off
HealthRBStop 1

month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

3.12.17 HealthRBDistance

指定した関節のモーター駆動量 (回転量) を表示します。

書式

HealthRBDistance [ロボット番号] [,関節番号]

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節のモーター駆動量 (回転量) を算出し、表示します。過去の使用量は加味されません。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#)

HealthRBDistance使用例

スカラロボットの第1関節の駆動量を表示する例です。

```
> HealthRBDistance 1, 1
1.000
>
```

3.12.18 HealthRBDistance関数

指定した関節のモーター駆動量 (回転量) を返す関数です。

書式

HealthRBDistance([ロボット番号,] 関節番号)

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

駆動量を実数で返します。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節のモーター駆動量 (回転数) を返します。過去の使用量は加味されません。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#)

HealthRBDistance関数使用例

```
Function RobotPartAnalysis
  Real healthDistance

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthDistance = HealthRBDistance(1,1)
  Print "Distance =", Str$(healthDistance)
Fend
```


3.12.19 HealthRBIInfo

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を表示します。

書式

HealthRBIInfo ロボット番号, 部品種別[, 関節番号]

パラメーター

ロボット番号

交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16) で指定します。

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (0-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_ALL	0	全部品種別を指定します。
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

交換推奨時期までの残り月数を返す関節を整数値 (1-9) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を表示します。

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて演算されます。

ロボットの関節が指定した部品を使用していない場合には、-1を表示します。

注意

コントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間がHealthCalcPeriodで設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーをHealthCalcPeriodで設定された期間の2倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照[HealthRBAAlarmOn関数](#), [HealthRateRBInfo関数](#)**HealthRBInfo使用例**

ロボット1の、全関節の全部品の交換推奨時期までの残り月数を表示する例です。

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_ALL
BATTERY          240.000
BELT              -1.000,   -1.000,   38.689,   95.226
GREASE            -1.000,   -1.000,   21.130,   -1.000
MOTOR             240.000,  240.000,  240.000,  240.000
GEAR              240.000,  224.357,   -1.000,   -1.000
BALL_SCREW_SPLINE -1.000,   -1.000,  240.000,   -1.000
>
```

ロボット1の、全関節の減速機の交換推奨時期までの残り月数を表示する例です。

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_GEAR
GEAR              240.000,  224.357,   -1.000,   -1.000
>
```

ロボット1の、第2関節のモーターの交換推奨時期までの残り月数を表示する例です。

```
> HealthRBInfo 1, HEALTH_ROBOT_TYPE_MOTOR, 2
MOTOR             224.357
>
```

3.12.20 HealthRBIInfo関数

ロボットに関する指定した種別の部品の、交換推奨時期までの残り月数を返す関数です。

書式

HealthRBIInfo(ロボット番号, 部品種別, 関節番号)

パラメーター

ロボット番号

交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16) で指定します。

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

交換推奨時期までの残り月数を返す関節を整数値 (1-9) で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にも同じ値を返します。本コマンドは、付加軸に対しては使用できません。

戻り値

交換推奨時期までの残り月数を実数で返します (単位: 月)。

ロボットの関節が指定した部品を使用していない場合には、-1を返します。

解説

交換推奨時期までの月数は、過去の使用状況による部品消耗率、およびコントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて演算されます。

注意

コントローラーをHealthCalcPeriodで設定された期間稼働するごとに取得される消費率の変化量に基づいて交換推奨時期までの残り月数が演算されるため、以下のような場合には正常に演算されません。

- 稼働時間がHealthCalcPeriodで設定された期間に満たない状況で本コマンドを実行する場合。
- 長期間のロボット稼働停止期間を終えた後に本コマンドを実行する場合。
- 部品交換をして部品消耗のリセットを行った後に本コマンドを実行する場合。
- コントローラーの日時の設定を変更した場合。

このような場合は、コントローラーをHealthCalcPeriodで設定された期間の2倍以上稼働させた後で本コマンドを実行すると正しい値が表示されます。

参照[HealthRBArmOn関数](#), [HealthRateRBInfo関数](#)**HealthRBInfo関数使用例**

ロボット1の第3関節ボールねじスプラインの交換推奨時期までの残り月数が1か月以下になった場合にアラームを出力する例です。

```
Function AlarmCheck
  Real month

  month = HealthRBInfo(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  If month < 1 Then
    Print "Alarm ON"
  EndIf
Fend
```

3.12.21 HealthRBRateOffset

指定した部品種別の消耗率に対してオフセットを設定します。

書式

HealthRBRateOffset ロボット番号, 部品種別, 関節番号, オフセット値

パラメーター

ロボット番号

ロボット番号を整数値 (1-16)で指定します。

部品種別

ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

関節を整数値 (1-6)で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にもオフセットを設定します。本コマンドは、付加軸に対しては使用できません。

オフセット値

消耗率に対してオフセットする整数値を指定します。(単位: %)

解説

指定した部品種別、関節に対して、消耗率のオフセットを設定します。

参照

[HealthRBAAlarmOn関数](#), [HealthRateRBInfo関数](#), [HealthRBInfo](#)

HealthRBRateOffset使用例

ロボット1の第1関節の減速機の消耗率に対して10 %加算する例です。

```
> HealthRBRateOffset 1,HEALTH_ROBOT_TYPE_GEAR,1,10
>
```

3.12.22 HealthRBReset

指定した部品種別の交換推奨時期までの残り月数、および消耗率をクリアします。

書式

HealthRBReset ロボット番号, 部品種別, 関節番号

パラメーター

ロボット番号

ロボット番号を整数値 (1-16)で指定します。

部品種別

ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

交換推奨時期までの残り月数を返す関節を整数値 (1-6)で指定します。部品種別でバッテリーを選択した場合には、バッテリーはすべての関節で共通のため、どの関節を指定した場合にもクリアします。本コマンドは、付加軸に対しては使用できません。

解説

指定した部品種別、関節に対して、交換推奨時期までの残り月数、および消耗率をクリアします。

警告も同時に解除します。

参照

[HealthRBAlarmOn関数](#), [HealthRateRBInfo関数](#), [HealthRBInfo](#)

HealthRBReset使用例

```
> HealthRBReset 1,HEALTH_ROBOT_TYPE_GEAR,1
>
```

3.12.23 HealthRBSpeed

指定した関節の速度の平均値を表示します。

書式

HealthRBSpeed [ロボット番号] [, 関節番号]

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節の速度の絶対値の平均を表示します。結果は0～1までの実数値で表されます。最大平均速度値は "1" になります。

平均値が0.001以下の値になる場合は "0" になります。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#), [AvgSpeed](#)

HealthRBSpeed使用例

スカルロボットの第1関節の速度を表示する例です。

```
> HealthRBSpeed 1, 1
0.100
>
```

3.12.24 HealthRBSpeed関数

指定した関節の、速度の絶対値の平均値を返します。

書式

HealthRBSpeed ([ロボット番号,] 関節番号)

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

0～1までの実数値で返します。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節の速度の絶対値の平均値を返します。結果は0～1までの実数値で表されます。最大平均速度は1になります。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#), [AvgSpeed](#)

HealthRBSpeed関数使用例

```
Function RobotPartAnalysis
  Real healthSpeed

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthSpeed = HealthRBSpeed(1,1)
  Print "AveSpeed =", Str$(healthSpeed)
Fend
```


3.12.25 HealthRBStart

あるサイクルのロボット動作に対する部品の使用可能月数と要素の演算を開始します。

書式

HealthRBStart ロボット番号

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。

解説

指定されたロボット番号に対する部品の使用可能月数と要素 (トルク, 速度, 駆動量) の演算を開始します。

演算を開始している状態で、再度実行すると、前回の演算結果は初期化されます。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBAnalysis](#), [HealthRBStop](#), [HealthRBTRQ](#), [HealthRBSpeed](#), [HealthRBDistance](#)

HealthRBStart使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

3.12.26 HealthRBAalysis

あるサイクルのロボット動作に関して、指定した部品種別の使用可能月数をシミュレートし、表示します。

書式

HealthRBAalysis ロボット番号, 部品種別[, 関節番号]

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。

部品種別

ロボットに関する部品を整数値、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_ALL	0	全部品種別を指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

関節番号

関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

あるサイクルのロボット動作で、指定した部品種別の使用可能月数をシミュレートします。部品が新品の状態で24時間連続稼働した場合に何カ月使用できるか算出し、表示します。過去の使用状況は加味されません。

指定の部品種別が、指定した関節で未実装の場合には、-1を返します。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#)

HealthRBAalysis使用例

スカラロボットの、全関節の全部品の使用可能月数を表示する例です。

```
> HealthRBAalysis 1, HEALTH_ROBOT_TYPE_ALL
BELT          -1.000,  -1.000,  38.689,  95.226
GREASE        -1.000,  -1.000,  21.130,  -1.000
MOTOR         240.000, 240.000, 240.000, 240.000
GEAR          240.000, 224.357,  -1.000,  -1.000
BALL_SCREW_SPLINE -1.000,  -1.000, 240.000,  -1.000
>
```

スカラロボットの、全関節の減速機の使用可能月数を表示する例です。

```
> HealthRBAAnalysis 1, HEALTH_ROBOT_TYPE_GEAR
GEAR          240.000, 224.357, -1.000, -1.000
>
```

6軸ロボットの、第2関節のモーターの使用可能月数を表示する例です。

```
> HealthRBAAnalysis 1, HEALTH_ROBOT_TYPE_MOTOR, 2
MOTOR          224.357
>
```

3.12.27 HealthRBStop

あるサイクルのロボット動作に対する部品の使用可能月数と要素の演算を終了します。

書式

HealthRBStop ロボット番号

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。

解説

指定されたロボット番号に対する部品の使用可能月数と要素 (トルク, 速度, 駆動量) の演算を終了します。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。
- 演算開始して1時間過ぎると、自動的に終了します。自動的に終了後、このコマンドを実行するとエラーになります。
- HealthRBStartコマンドを実行せずに、このコマンドを実行するとエラーになります。
- HealthRBStop実行後にHealthRBStartを実行せずに、再度HealthRBStopを実行した場合にもエラーになります。

参照

[HealthRBAnalysis](#), [HealthRBStart](#), [HealthRBTRQ](#), [HealthRBSpeed](#), [HealthRBDistance](#)

HealthRBStop使用例

```
Function RobotPartAnalysis
  Real month

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  month = HealthRBAnalysis(1, HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE, 3)
  Print "Ball Screw Spline analysis =", Str$(month)
Fend
```

3.12.28 HealthRBTRQ

指定した関節の寿命影響トルク値を表示します。

書式

HealthRBTRQ [ロボット番号] [, 関節番号]

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。関節番号を指定しない場合は、全関節の値を返します。本コマンドは、付加軸に対しては使用できません。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節の寿命影響トルク値を表示します。結果は0～1までの実数値で表されます。最大トルク値は "1" になります。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#), [ATRQ](#)

HealthRBTRQ使用例

スカラロボットの第1関節の寿命影響トルク値を表示する例です。

```
> HealthRBTRQ 1, 1
0.020
>
```

3.12.29 HealthRBTRQ関数

指定した関節の寿命影響トルク値を返します。

書式

HealthRBTRQ ([ロボット番号,] 関節番号)

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。省略可能です。省略された場合は、カレントロボット番号を使用します。

関節番号

関節を整数値 (1-6) で指定します。本コマンドは、付加軸に対しては使用できません。

戻り値

0～1までの実数値で返します。

解説

HealthRBStartからHealthRBStopまでのロボット動作に対して、指定した関節の寿命影響トルク値を返します。結果は0～1までの実数値で表わされます。最大実効トルクは1になります。

注意

- Autoモードでは、機能しません。
- ドライラン (仮想コントローラー含む) では、機能しません。

参照

[HealthRBStart](#), [HealthRBStop](#), [ATRQ](#)

HealthRBTRQ関数使用例

```
Function RobotPartAnalysis
  Real healthTRQ

  Robot 1

  HealthRBStart 1
  Motor On
  Go P0
  Go P1
  Motor Off
  HealthRBStop 1

  healthTRQ = HealthRBTRQ(1,1)
  Print "Torque =", Str$(healthTRQ)
End
```

3.12.30 HealthRBWarningEnable

ロボットに関する指定した種別の部品の部品消耗アラームの通知を有効または無効にします。

書式

HealthRBWarningEnable ロボット番号, 部品種別[, On/Off]

パラメーター

ロボット番号

ロボット番号を整数値 (1-16) で指定します。

部品種別

ロボットに関する部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

On/Off

- On: 部品消耗アラームの通知を有効にします。
- Off: 部品消耗アラームの通知を無効にします。

結果

On/Offパラメーターを省略すると、現在のOn/Off設定値が表示されます。

解説

指定した種別の部品の部品消耗アラームが発生した場合、部品消耗アラーム通知を行うかどうかを設定します。

注意

指定した部品の部品消耗アラームの通知を無効にした場合、交換推奨時期を経過してもワーニングが通知されません。本コマンドによる設定には十分注意して実行してください。

参照

[HealthRBAlarmOn関数](#)

HealthRBWarningEnable使用例

スカラロボット1の、グリス部品の部品消耗アラーム通知を無効にする例です。

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE, Off
```

スカラロボット1の、グリス部品の部品消耗アラーム通知設定を表示する例です。

```
> HealthRBWarningEnable 1, HEALTH_ROBOT_TYPE_GREASE
GREASE Off
>
```

3.12.31 HealthRBWarningEnable関数

ロボットに関する指定した種別の部品の部品消耗アラームの通知の設定状態を返す関数です。

書式

HealthRBWarningEnable(ロボット番号, 部品種別)

パラメーター

ロボット番号

交換推奨時期までの残り月数を返すロボット番号を整数値 (1-16) で指定します。

部品種別

交換推奨時期までの残り月数を返す部品を整数値 (1-6)、または以下に示す定数で指定します。

定数	値	モード
HEALTH_ROBOT_TYPE_BATTERY	1	バッテリーを指定します。
HEALTH_ROBOT_TYPE_BELT	2	ベルトを指定します。
HEALTH_ROBOT_TYPE_GREASE	3	グリスを指定します。
HEALTH_ROBOT_TYPE_MOTOR	4	モーターを指定します。
HEALTH_ROBOT_TYPE_GEAR	5	減速機を指定します。
HEALTH_ROBOT_TYPE_BALL_SCREW_SPLINE	6	ボールねじスプラインを指定します。

戻り値

部品消耗アラームの設定値を整数値で返します。

- 1: On
- 0: Off

参照

[HealthRBAlarmOn関数](#)

HealthRBWarningEnable関数使用例

スカラロボット1のグリス部品の部品消耗アラームの通知設定状態を表示する例です。

```
Print HealthRBWarningEnable(1, HEALTH_ROBOT_TYPE_GREASE )
```


3.12.32 Here

現在位置をロボット座標として教示します。

書式

Here 座標

パラメーター

座標

P番号, P(式), ポイントラベルのいずれかを指定します。

注意

- Hereステートメントと並列処理

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

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

上記のような使い方はできません。

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

上記のようなプログラムに変更してください。

- Hereステートメントとマルチタスク

HereステートメントをXqtなどで実行したマルチタスク関数内で実行したとき、メインタスク内でMove, Goなどでロボットが動作中である場合、エラー停止します。

CurPosを使うことで、動作中の現在位置が取得できます。

例

```
Function Xqt_PrintHere  
Do  
Print CurPOS  
Wait 0.1  
Loop  
Fend  
Function main  
Xqt 10, Xqt_PrintHere  
Go P0  
Fend
```

- CPとの併用

バスモーション有効時にHereを使用する場合、Hereを指定した動作の前で動作軌道が合成されずに減速します。以下のような使い方をした場合、P1で減速して停止した後に、次の動作を開始します。

```
Go P1 CP  
Go Here+X(100)
```

停止せずに動作させたい場合は、Hereの位置を事前に計算するようにプログラムを修正してください。

```
Go P1 CP  
Go P1+X(100)
```

参照

[Here関数](#), [CurPos関数](#)

Here使用例

```
Here P1  
Here pick
```

3.12.33 Here関数

ロボットの現在位置を返します。

書式

Here

戻り値

ロボットの現在位置を返します。

解説

Here関数は使用中のマニピュレーターの現在位置を取得するために使います。

参照

[Here関数](#)

Here関数使用例

```
P1 = Here
```

3.12.34 Hex\$関数

16進数表現の数値を文字化して返します。

書式

Hex\$(数値)

パラメーター

数値

整数値を指定します。

戻り値

16進数表現の数値をASCII値の文字列として返します。

解説

Hex\$関数は、16進数表現の数値を文字化して返します。文字は0～9, A～Fで構成します。Hex\$はStat関数の結果確認に使用します。

参照

[Str\\$関数](#), [Stat関数](#), [Val関数](#)

Hex\$関数使用例

```
> print hex$(stat(0))
A00000
> print hex$(255)
FF
```

3.12.35 History

コマンドの実行履歴を表示します。

書式

History

結果

実行した順番に、コマンドの実行履歴を表示します。

解説

直近にコマンドウィンドウで実行したコマンドの実行履歴(最大100件)を表示します。

注意

このコマンドはコマンドウィンドウのみ実行可能です。

参照

[ClearHistory](#)

History使用例

```
> Motor On
> Go P1
> History
Motor On
Go P1
```

3.12.36 Hofs

エンコーダー原点からソフトウェア原点への補正パルスの設定と表示をします。

書式

(1) Hofs 第1関節設定値, 第2関節設定値, 第3関節設定値, 第4関節設定値 [, 第5関節設定値, 第6関節設定値] [, 第7関節設定値] [, 第8関節設定値, 第9関節設定値]

(2) Hofs

パラメーター

第1関節設定値

第1関節補正パルス値 (整数)を、式または数値で指定します。

第2関節設定値

第2関節補正パルス値 (整数)を、式または数値で指定します。

第3関節設定値

第3関節補正パルス値 (整数)を、式または数値で指定します。

第4関節設定値

第4関節補正パルス値 (整数)を、式または数値で指定します。

第5関節設定値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第5関節補正パルス値 (整数)を、式または数値で指定します。

第6関節設定値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第6関節補正パルス値 (整数)を、式または数値で指定します。

第7関節設定値

ジョイント型7軸ロボット用のパラメーターです。第7関節補正パルス値 (整数)を、式または数値で指定します。

第8関節設定値

付加軸S関節用のパラメーターです。第8関節 (付加軸S)補正パルス値 (整数)を、式または数値で指定します。

第9関節設定値

付加軸T関節用のパラメーターです。第9関節 (付加軸T)補正パルス値 (整数)を、式または数値で指定します。

戻り値

パラメーターの指定がないときは、現在のHofs設定値を表示します。

解説

Hofsは、原点補正パルス値を表示または設定します。Hofsは、エンコーダー原点 (Z相)からメカ原点までのオフセット値を設定します。

ロボット動作制御は、各関節に搭載されたエンコーダーの原点に基づいていますが、エンコーダー原点がロボットのメカ原点と必ずしも一致するわけではありません。そこで、メカ原点と一致するエンコーダーの位置をソフトウェア上の原点とするために、Hofsにより補正パルス量を設定します。

注意

- Hofs値は、必要な時以外、絶対に変更しないでください

Hofs値は工場出荷時に精密に設定されています。不必要にこの値を変更すると、位置決めエラーや思わぬ動作の原因になり危険です。必要な時以外は、Hofs値を絶対に変更しないようにしてください。

- JointAccuracyのリセット(関節精度補正対応機種のみ)

関節精度補正対応機種では、Hofsにより原点補正パルス値の設定を行うと、変更があった関節に対しては、JointAccuracyで設定した関節精度補正の補正值が"0"にリセットされます。JointAccuracyで設定した補正值をリセットしたくない場合は、HofsJointAccuracyを使用してください。

- Hofs値を自動的に演算するには

Hofs値を自動的に演算するためには、アームをキャリブレーションしたい位置まで動かし、Calibを実行します。そうすると、CalPlsパルス値およびキャリブレーション位置パルス値に基づき、コントローラーが自動的にHofs値を演算します。

- Hofsの保存と復元

Hofsは、[システム設定]メニュー-[システム設定]ダイアログ-[ロボット]-[キャリブレーション]にある[保存]および[読み込み]を使って保存および復元することができます。

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

Safety基板を搭載したコントローラーでは、コントローラーのHofs値と安全機能を実現するSafety基板のHofs値が一致する必要があります。

本コマンドを実行すると、コントローラーのHofs値のみが変更され、Safety基板設定と差異が生じるため、ワーニングが発生します。

このため、本コマンド実行後は、Safety基板設定を更新するために、安全機能マネージャーを起動してください。

詳細は、以下のマニュアルを参照してください。
"ロボットコントローラー 安全機能マニュアル"

参照

[Calib](#), [CalPls](#), [JointAccuracy](#), [HofsJointAccuracy](#), [Home](#), [Hordr](#), [MCal](#), [SysConfig](#)

Hofs使用例

下記は、モニターウィンドウでの簡単な使用例です。第1関節原点補正值を-545に、第2関節原点補正值を514に、第3関節と第4関節の原点補正值を0に設定し、その後、現在の原点補正值を表示します。

```
> hofs -545, 514, 0, 0  
  
> hofs  
-545, 514, 0, 0  
>
```

3.12.37 Hofs関数

各関節のエンコーダー原点からソフトウェア原点への補正パルス値を返す関数です。

書式

Hofs(関節番号)

パラメーター

関節番号

Hofs値を求める関節番号 (整数)を、式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

指定した関節の補正パルス値 (整数値、単位: パルス)を返します。

参照

[Calib](#), [CalPls](#), [Home](#), [Hordr](#), [MCal](#), [SysConfig](#)

Hofs関数使用例

下記はHofs関数を使ったプログラムの例です。

```
Function DisplayHofs
  Integer i

  Print "Hofs settings:"
  For i = 1 To 4
    Print "Joint ", i, " = ", Hofs(i)
  Next i
Fend
```


3.12.38 HofsJointAccuracy

関節精度補正の補正值を変更せずに、エンコーダー原点からソフトウェア原点への補正パルスの設定と表示をします。

書式

(1) HofsJointAccuracy 第1関節設定値, 第2関節設定値, 第3関節設定値, 第4関節設定値 [, 第5関節設定値, 第6関節設定値] [, 第7関節設定値] [, 第8関節設定値, 第9関節設定値]

(2) HofsJointAccuracy

パラメーター

第1関節設定値

第1関節補正パルス値 (整数)を、式または数値で指定します。

第2関節設定値

第2関節補正パルス値 (整数)を、式または数値で指定します。

第3関節設定値

第3関節補正パルス値 (整数)を、式または数値で指定します。

第4関節設定値

第4関節補正パルス値 (整数)を、式または数値で指定します。

第5関節設定値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第5関節補正パルス値 (整数)を、式または数値で指定します。

第6関節設定値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第6関節補正パルス値 (整数)を、式または数値で指定します。

第7関節設定値

ジョイント型7軸ロボット用のパラメーターです。第7関節補正パルス値 (整数)を、式または数値で指定します。

第8関節設定値

付加軸S関節用のパラメーターです。第8関節 (付加軸S)補正パルス値 (整数)を、式または数値で指定します。

第9関節設定値

付加軸T関節用のパラメーターです。第9関節 (付加軸T)補正パルス値 (整数)を、式または数値で指定します。

戻り値

パラメーターの指定がないときは、現在のHofs設定値を表示します。

解説

HofsJointAccuracyは、関節精度補正の補正值を変更せずに、原点補正パルス値を表示、または設定します。関節精度補正対応機種については、以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

ロボット動作制御は、各関節に搭載されたエンコーダーの原点に基づいていますが、エンコーダー原点がロボットのメカ原点と必ずしも一致するわけではありません。そこで、メカ原点と一致するエンコーダーの位置をソフトウェア上の原点とするために、Hofsにより補正パルス量を設定します。

Hofsは各関節について変更がある場合は、それぞれのJointAccuracyで設定した関節精度補正の補正値を "0" にリセットしますが、HofsJointAccuracyではリセットを行いません。関節精度補正の補正値を "0" にリセットしたくない場合は、HofsJointAccuracyを使用してください。

注意

- Hofs値は、必要な時以外、絶対に変更しないでください

Hofs値は、工場出荷時に精密に設定されています。不必要に この値を変更すると、位置決めエラーや思わぬ動作の原因になり危険です。必要な時以外は、Hofs値を絶対に変更しないようにしてください。

参照

[JointAccuracy](#), [Hofs](#)

HofsJointAccuracy使用例

下記は、コマンドウィンドウでの簡単な使用例です。第1関節原点補正値を "-545" に、第2関節原点補正値を "514" に、第3関節と第4関節の原点補正値を "0" に設定し、その後、現在の原点補正値を表示します。

HofsJointAccuracyの使用前後で関節精度補正の補正値は変化していません。

```
> JointAccuracy 1
1000, 420, 100, 240
> HofsJointAccuracy -545, 514, 0, 0

> HofsJointAccuracy
-545, 514, 0, 0
> JointAccuracy 1
1000, 420, 100, 240
>
```

3.12.39 Home

ロボットアームをユーザー定義のホーム位置まで移動します。

書式

Home

解説

HomeSetで指定されたホーム位置 (待機姿勢) まで、Hordrで設定された関節順序にしたがって、低速のPTP動作で移動します。

水平多関節型ロボット (RSシリーズを含む) では、通常、第3関節が1番最初にHomeSet位置まで戻ってから、第1関節, 第2関節, 第4関節が、同時にそれぞれのHomeSet座標位置まで戻ります。ホーム位置に戻る関節の順番は、Hordr命令で変更することができます。

注意

- Home 状態の出力

ロボットがホーム位置 (待機姿勢) にあるとき、コントローラーのHome出力はオンになります。

起こりやすいエラー

- HomeSet値を設定しないでHomeを実行した場合

HomeSet値を設定しないでHomeを実行すると、エラー2228が発生します。

参照

[HomeClr](#), [HomeDef関数](#), [Hordr](#), [HomeSet](#)

Home使用例

Home命令は、下記のようなプログラムで使用できます。

```
Function InitRobot
  Reset
  If Motor = Off Then
    Motor On
  EndIf
  Home
Fend
```

あるいは、コマンドウィンドウから下記のように発行します。

```
> home
>
```

3.12.40 HomeClr

ホーム位置の設定をクリアします。

書式

HomeClr

参照

[HomeDef関数](#), [HomeSet](#)

HomeClr使用例

HomeClr関数は、下記のようなプログラムで使用できます。

```
Function ClearHome
    If HomeDef = True Then
        HomeClr
    EndIf
Fend
```

3.12.41 HomeDef関数

ホーム位置が設定されたかどうか返します。

書式

HomeDef

戻り値

設定された場合は"True"、設定されていない場合は"False"を返す。

参照

[HomeClr](#), [HomeSet](#)

HomeDef関数使用例

HomeDef関数は、下記のようなプログラムで使用できます。

```
Function DisplayHomeSet

    Integer i

    If HomeDef = False Then
        Print "Home is not defined"
    Else
        Print "Home values:"
        For i = 1 To 4
            Print "J", i, " = ", HomeSet(i)
        Next i
    EndIf
Fend
```

3.12.42 HomeSet

ホーム位置 (待機姿勢)の姿勢パルスの設定と表示をします。

書式

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

(2) HomeSet

パラメーター

第1関節パルス値

第1関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第2関節パルス値

第2関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第3関節パルス値

第3関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第4関節パルス値

第4関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第5関節パルス値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第5関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第6関節パルス値

垂直6軸型ロボット (Nシリーズを含む)用のパラメーターです。第6関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第7関節パルス値

ジョイント型7軸ロボット用のパラメーターです。第7関節ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第8関節パルス値

付加軸S関節用のパラメーターです。第8関節 (付加軸S)ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

第9関節パルス値

付加軸T関節用のパラメーターです。第9関節 (付加軸T)ホームエンコーダーパルス値 (整数)を、式または数値で指定します。

結果

パラメーターを省略すると現在のホームパルス値を表示します。

解説

ユーザーは、各関節に対するパルス値を指定し、ホーム (待機)位置を設定できます。

起こりやすいエラー

- HomeSet 値を設定せずにHomeを実行したとき

HomeSet値を設定せずにHomeを実行すると、エラー2228が発生します。

- HomeSet 値を設定せずにHomeSet 値を表示させようとしたとき

HomeSet値を設定せずにホーム位置のパルス値を表示しようとする、エラー2228が発生します。

参照

[Home](#), [HomeClr](#), [HomeDef関数](#), [Hordr](#), [Pls関数](#)

HomeSet使用例

下記はコマンドウィンドウからの操作例です。

```
> homeset 0,0,0,0   'ホーム位置を0, 0, 0, 0に設定
> homeset
  0  0
  0  0

> home              'ロボットがホーム位置へ移動
```

Pls関数を使って、アームの現在位置をホーム位置と設定します。

```
> homeset Pls(1), Pls(2), Pls(3), Pls(4)
```

3.12.43 HomeSet関数

指定された関節のホーム位置 (待機姿勢) のパルス値を返す関数です。

書式

HomeSet(関節番号)

パラメーター

関節番号

HomeSet値を求める関節番号を、式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

指定関節のホームパルス値を返します。関節番号に"0"を指定すると、HomeSet値の設定がある場合は、"1"を、設定がない場合は"0"を返します。

参照

[HomeSet](#)

HomeSet関数使用例

下記は、HomeSet関数を使ったプログラムの例です。

```
Function DisplayHomeSet

  Integer i

  If HomeSet(0) = 0 Then
    Print "HomeSet is not defined"
  Else
    Print "HomeSet values:"
    For i = 1 To 4
      Print "J", i, " = ", HomeSet(i)
    Next i
  EndIf
Fend
```


3.12.44 Hordr

Home実行時の各関節の動作順序の指定と表示をします。

書式

(1) Hordr 設定値1, 設定値2, 設定値3, 設定値4 [, 設定値5] [, 設定値6] [, 設定値7] [, 設定値8][, 設定値9]

(2) Hordr

パラメーター

設定値1

第1ステップで復帰する関節をビットパターンで指定します。

設定値2

第2ステップで復帰する関節をビットパターンで指定します。

設定値3

第3ステップで復帰する関節をビットパターンで指定します。

設定値4

第4ステップで復帰する関節をビットパターンで指定します。

設定値5

第5ステップで復帰する関節をビットパターンで指定します。

設定値6

第6ステップで復帰する関節をビットパターンで指定します。

設定値7

第7ステップで復帰する関節をビットパターンで指定します。

設定値8

第8ステップで復帰する関節をビットパターンで指定します。

設定値9

第9ステップで復帰する関節をビットパターンで指定します。

結果

パラメーターを省略すると現在の復帰順序設定を表示します。

解説

Hordrは、Homeコマンド実行時の関節の復帰順序を設定します。どの関節が何番目にホーム位置 (待機姿勢)へ移動するかが決まります。

Hordr命令は、Home実行時の各関節の復帰順序を、ユーザーが変更できるようにします。復帰順序は、ロボットの型によって、4つ、6つ、あるいは9つのステップに分かれて設定されます。ユーザーはHordrにより各ステップで復帰する関節を指定することができます。各ステップで復帰する関節は、複数を指定することも可能です。理論的には全関節が同時に復帰することもあり得ます。しかし、水平多関節型ロボット (RSシリーズを含む4軸ロボット)の場合、一般的には、第3関節を最初にステップ1で移動させ、他の関節をそれ以降のステップで復帰させることをお勧めします。

Hordr命令では、各ステップに対しビットパターンが指定されなければなりません。各関節に対し、ビットパターンが定められています。あるステップで、ビットが"1"であれば、対応する関節がホーム位置 (待機姿勢)へ移動します。ビットを"0"にクリアすると、対応する関節は、そのステップにおいてはホーム位置 (待機姿勢)へ移動しません。各関節に対するビットパターンは次のように割り当てられています。

ビットパターン表

関節名	ビット番号	2進数表記
第1関節	bit 0	&B0001
第2関節	bit 1	&B0010
第3関節	bit 2	&B0100
第4関節	bit 3	&B1000
第5関節	bit 4	&B10000
第6関節	bit 5	&B100000
第7関節	bit 6	&B1000000
第8関節	bit 7	&B10000000
第9関節	bit 8	&B100000000

参照

[Home](#), [HomeSet](#)

Hordr使用例

下記はコマンドウィンドウからの水平多関節型ロボット (RSシリーズを含む4軸ロボット) の操作例です。

この例では、ロボットアームのホーム位置への復帰順序を2進数で次のように設定します

第1ステップで第3関節、第2ステップで第1関節、第3ステップで第2関節、第4ステップで第4関節がホーム位置へ復帰します。

```
>hordr  &B0100, &B0001, &B0010, &B1000
```

この例では、ロボットアームのホーム位置への復帰順序を10進数で次のように設定します 第1ステップで第3関節、第2ステップで第1関節、第2関節、第4関節が同時にホーム位置へ復帰します。

```
>hordr  4, 11, 0, 0
```

次の例では、現在の復帰順序を10進数で表示しています。

```
>hordr
4, 11, 0, 0
>
```

3.12.45 Hordr関数

Home実行時の、指定された動作順序において復帰する関節を返します。

書式

Hordr(設定値番号)

パラメーター

設定値番号

Hordr命令の動作順序を表わす整数値を指定します。

戻り値

Hordrステートメントで指定された動作順序で復帰する関節をビットパターンで返します。

参照

[Home](#), [HomeSet](#), [Hordr](#)

Hordr関数使用例

```
Integer a  
a = Hordr(1)
```

3.12.46 Hour

コントローラーの積算通電時間の表示をします。

書式

Hour

解説

コントローラーの電源を投入し、積算時間を表示します。(積算通電時間)積算時間の単位は時間です。

参照

[Time関数](#)

Hour使用例

次の例は、コマンドウィンドウから実行します。

```
> hour  
2560  
>
```

3.12.47 Hour関数

コントローラーの積算通電時間を返します。

書式

Hour

戻り値

コントローラーの積算通電時間を実数値で返します。

参照

[Time関数](#)

Hour関数使用例

```
Print "Number of controller operating hours: ", Hour
```

3.13 |

3.13.1 If...Then...Else...EndIf

指定された条件分岐にしたがって命令を実行します。

書式

(1) If条件式 Then

ステートメントT1

・
・

[ElseIf 条件式 Then]

ステートメントT1

・
・

[Else]

ステートメントF1

・
・

EndIf

(2) If 条件式 Then ステートメントT1 [;ステートメントT2...] [Else ステートメントF1 [;ステートメントF2...]]

パラメーター

条件式

真偽値 (True/Falseの値)が戻る有効な条件式です。真 (True)の場合は"0"以外の数値を、偽 (False)の場合は、"0"を返します。(下記の条件式例を参照してください。)

ステートメントT1

条件式の値が真 (True)のとき、つまり、条件が満たされたときに実行されます。(複数のステートメントを If...Then...Else形式のブロックの中に記述できます。)

ステートメントF1

条件式の値が偽 (False)のとき、つまり、条件が満たされなかったときに実行されます。(複数のステートメントを If...Then...Else形式のブロックの中に記述できます。)

解説

(1) If...Then...Else は、条件が満たされたときにステートメントT1以降を実行します。条件が満たされなかったときは、ステートメントF1以降を実行します。If...Then...Else命令のElse部分は省略可能です。Else部分を省略し、条件が満たされなかったときは、EndIf以降のステートメントが実行されます。If...Then...Else ステートメントのブロックは、Elseの有無にかかわらず、EndIfで閉じてください。

(2) If...Then...Elseは、ブロック以外の形式で用いることも可能です。If...Then...Elseのステートメントをすべて同じ行に記述する方法です。この方法でIf...Then...Elseを用いる場合には、EndIfは必要ではありません。この行で記述されている条件式が満たされれば (条件式に対して真値 (True)であれば)、ThenとElse間のステートメントが実行されます。条件式が満たされなければ (条件式に対して偽値 (False)であれば)、Elseに続くステートメントが実行されます。If...Then...ElseのElse部分は必須ではありません。条件式が満たされない場合 (条件式に対して偽値 (False)の場合)でElseキーワードがなければ、制御はプログラム中の次のステートメントに移行します。

注意

- 条件式サンプル

- $a = b$: aとbが等しい
- $a < b$: aがbより小さい
- $a \geq b$: aがbより大きいかまたは等しい
- $a <> b$: aとbが等しくない
- $a > b$: aがbより大きい
- $a \leq b$: aがbより小さいかまたは等しい

論理演算子And, Or, Xorなども使うことができます。

■ 条件式にTrueを使用する場合

定数 Trueの値は-1ですが、Boolean型なので、他の型の変数との比較条件に使用する場合注意が必要です。

```
Function main
  Integer i
  i = 3
  If i = True Then
    Print "i=TRUE"
  EndIf
Fend
```

上記プログラムを実行すると "i=TRUE"が表示されます。これは、Boolean型を含む条件判定は、"0"か"非0"かで行われるためです。

"i"の値が"0"でない場合は条件成立と判定され表示が行われます。プロジェクトプロパティ>コンパイラーで「変数型チェック」を有効にすると、コンパイル時にBoolean型と数式との比較をエラーとして検出できます。

参照

[Select...Send, Do...Loop](#)

If/Then/Else使用例

[1行のIf...Then...Else]

下記例は、特定の出力のオンオフを決定するために、入力をチェックする簡単な例です。このようなタスクは、常に動かしておくI/Oタスクなどに使用します。

```
Function main
  Do
    If Sw(0) = 1 Then On 1 Else Off 1
  Loop
Fend
```

[ブロック形式のIf...Then...Else]

下記は、いくつかの入力をチェックして、その入力の状態を出力する簡単な例です。

```
If Sw(0) = 1 Then Print "Input0 ON" Else Print "Input0 OFF"
'
If Sw(1) = 1 Then
  If Sw(2) = 1 Then
    Print "Input1 On and Input2 ON"
  Else
    Print "Input1 On and Input2 OFF"
  EndIf
Else
  If Sw(2) = 1 Then
    Print "Input1 Off and Input2 ON"
```

```
Else
  Print "Input1 Off and Input2 OFF"
EndIf
EndIf
```

[その他の書式の例]

```
If x = 10 And y = 3 Then GoTo Label50
If test <= 10 Then Print "Test Failed"
If Sw(0) = 1 Or Sw(1) = 1 Then Print "Everything OK"
```


3.13.2 ImportPoints

現在選択中のロボットのプロジェクトにポイントファイルをインポートします。

書式

ImportPoints ソースパス,ファイル名 [, ロボット番号]

パラメーター

ソースパス

カレントプロジェクトにインポートしたい特定のフォルダーやファイルを表す文字列式。拡張子は".pts"固定です。詳細は、ChDiskを参照してください。

ファイル名

カレントロボットのカレントプロジェクトへインポートさせたい特定のファイルを表す文字列式拡張子はptsです。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

ロボット番号

どのロボットがポイントファイルに関連するのかを指定する整数式省略可能です。ロボット番号=0の場合、ポイントファイルは共通ポイントファイルとしてインポートされます。省略された場合は、0(共通ポイントファイル)を使用します。

解説

ImportPointsは、現在選択中のプロジェクトにポイントファイルをコピーして、そのポイントファイルを現在選択中のロボットのファイルに追加します。追加されたポイントファイルはコンパイルされ、LoadPointsコマンドで読み込み可能になります。現在選択中のロボットに同じファイルが既に存在していた場合は上書きされ、再コンパイルされます。

起こりやすいエラー

- ファイルが存在しない場合

指定したソースパスが存在しない場合、エラーが発生します。

- 指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。

- 指定したファイルが現在のロボットのファイルでない場合

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。

参照

[LoadPoints](#), [Robot](#), [SavePoints](#)

ImportPoints使用例

```
Function main
  Robot 1
  ImportPoints "c:\mypoints\model1.pts", "robot1.pts", 1
  LoadPoints "robot1.pts"
Fend
```

3.13.3 In関数

指定された入力ポートの状態をバイト単位で返します。バイトポートは、8個の入力ビットで構成されます。

書式

In(バイトポート番号)

パラメーター

バイトポート番号

I/Oのバイトポートを指定します。

戻り値

0～255の整数を返します。戻り値は8ビットで、各ビットが入力ビット1個に対応します。

解説

Inでは、8個の入力ビットの値を同時に見ることができます。In命令は、8個のI/Oビットの状態を1つの変数の値として保存したり、Wait命令とともに用いることで、「2個以上のI/Oビットの状態が特定の条件に一致するまで待機させる」といった使い方ができます。

1度に8個の入力ビットをチェックすることができるので、戻り値は、0～255の範囲の整数値になります。下記表で、戻り値の各整数値が、それぞれの入力ビットにどのように対応しているか参照してください。

入力ビット表 (バイトポート0使用時)

戻り値	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

入力ビット表 (バイトポート2使用時)

戻り値	23	22	21	20	19	18	17	16
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

参照

[InBCD関数](#), [MemIn関数](#), [MemOff](#), [MemOn](#), [MemSw関数](#), [Off](#), [On](#), [OpBCD](#), [Oport関数](#), [Out](#), [Sw関数](#), [Wait](#)

In関数使用例

下記のプログラム例は、入力ビット20, 21, 22, 23がセンサーデバイスに接続されており、アプリケーションは、その各々から"準備OK"を示すオン信号が戻ってから始動する、という仮定に基づいて記述されています。プログラム例では、バイトポート2の8個の入力ビットの状態を取得し、入力ビット20, 21, 22, 23がすべてオンであることを

確認してから次に進んでいます。もし、いずれかの入力ビットがオンでないと（いずれかで"1"が返されると）、エラーメッセージが表示され、タスクは停止します。

```
Function main
  Integer var1
  var1 = In(2)  'バイトポート2の8個の入力ビットの状態を取得
  If (var1 And &B11110000) = &B11110000 Then
    Go P1
    Go P2
    '他の動作コマンドをここで実行
    '
    '
  Else
    Print "Error in initialization!"
    Print "Sensory Inputs not ready for cycle start"
    Print "Please check inputs 20,21,22 and 23 for"
    Print "proper state for cycle start and then"
    Print "start program again"
  EndIf
Fend
```

コマンドウィンドウからは、入力を設定することはできませんが、確認することはできます。下記の例では、入力ビット1, 5, 15がオンであることを前提にしています。その他の入力はすべてオフになっています。

```
> print In(0)
34
> print In(1)
128
> print In(2)
0
```

3.13.4 InBCD関数

指定された8ビットポートごとの入力状態を読み取り、2桁の2進化10進数 (BCD) として読み取り、10進数に変換して返す関数です。

書式

InBCD(ポート番号)

パラメーター

ポート番号

I/Oの入力バイトを指定します。

戻り値

入力ポート (0～99)の状態をBCD (0～9)で返します。

解説

InBCDは、BCDで同時に8個の入力ラインを読みます。InBCD命令のパラメーター"ポート番号"で、どのビットグループの状態を読むか指定します。例えば、ポート番号= 0 のとき0～7の入力ビットの状態を、ポート番号= 1 のとき8～15の入力ビットの状態を読む、ということになります。

8個の入力ビットの状態は、BCDで返されます。戻り値は、0～99の1または2桁の数値です。1桁目 (10の位)は、ポート番号で指定した8個の出力ビットのうち上位4ビットに対応します。2桁目 (1の位)は、ポート番号で指定した8個の出力ビットのうち下位4ビットに対応します。

各桁におけるBCDの有効値は0～99なので、I/Oのすべての組み合わせを満たすことができません。下記表でポート番号= 0 のとき、可能なI/O状態の組み合わせと戻り値を、一部示しています。

入力設定 (入力番号)

戻り値	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

BCDでは、10進数しか指定することができません。このことは、特定のポートですべての入力ビットが同時にオンであるようなとき、BCDを使うInBCD命令では有効な値を求めることは不可能だということを意味します。

InBCDでは最大戻り値は、"99"です。上記の表からもわかるように、"99"が戻り値のときには、すべての入力が入オンにはなりません。戻り値"99"のときオンの入力ビットは、0, 3, 4, 7で、その他のビットはすべてオフになっています。

注意

InBCDとInの違い

- InBCD関数は、指定された8ビットポートの状態を、上位4ビット、下位4ビットで分けた2桁のBCDデータとして読み取ります。よって、4ビットの状態が9 (&B1001) を超える &B1010～&B1111 の場合、エラーが発生します。
- In関数は、指定された8ビットポートの状態をそのまま10進数に変換して返します。

参照

[In関数](#), [MemOff](#), [MemOn](#), [MemOut](#), [MemSw関数](#), [Off](#), [On](#), [OpBCD](#), [Oport関数](#), [Out](#), [Sw関数](#), [Wait](#)

InBCD使用例

コマンドウィンドウからの簡単な操作例です。

入力ビット0, 4, 10, 16, 17, 18 がオンで、他の入力ビットはオフだと仮定しています。

```
> Print InBCD(0)
11
> Print InBCD(1)
04
> Print InBCD(2)
07
>
```

3.13.5 Inertia

ロボットの負荷イナーシャと偏心量を設定します。

書式

(1) Inertia [負荷イナーシャ [, 偏心量]]

(2) Inertia

パラメーター

負荷イナーシャ

ハンドおよびワークを含む、先端関節の中心周りの慣性モーメント (イナーシャ) (実数、単位: $\text{kg}\cdot\text{m}^2$) を、数値または式で指定します。省略可能 (負荷イナーシャのみを省略することは不可能) です。

偏心量

ハンドおよびワークの重心の先端関節中心からの距離 (実数、単位: mm) を、数値または式で指定します。省略可能です。

結果

パラメーターを省略すると、現在設定されているイナーシャパラメーターを表示します。

[偏心量] を省略すると、入力された [負荷イナーシャ] が設定され、デフォルト値の [偏心量] が設定されます。

[負荷イナーシャ] のみを省略することはできません。

イナーシャ値や偏心量に、実際より小さい値を指定すると、過剰な加速値および減速値が設定され、マニピュレーターに損傷を与えることがありますので注意してください。また、エラーの発生や衝撃の原因となり、機能が発揮できないだけでなく、部品の寿命が低下したり、ベルトの歯飛びによる位置ズレが発生したりする可能性があります。

解説

先端関節周りの慣性モーメントを指定するとき、Inertiaステートメントを使います。これにより先端関節の加減速およびサーボゲインを適正に補正できます。また、偏心量パラメーターで、先端関節中心からハンドとワークの重心までの距離を指定できます。

"負荷, イナーシャ, 偏心/オフセット測定ユーティリティ" による設定も可能です。

詳細は、以下のマニュアルに記載しています。

"Epson RC+ ユーザーズガイド - 負荷, イナーシャ, 偏心/オフセット測定ユーティリティ"

注意

- Inertiaの設定値は、電源オフでも変更されません

Inertiaの設定値は、一度設定するとコントローラーに記憶されます。電源をオフしても変更されません。

何も設定しない場合は、前回設定された値のままになります。

参照

Inertia関数

ハンド制御コマンドについては、以下のマニュアルを参照してください。

"Hand機能"

Inertia使用例

Inertia 0.02, 1

3.13.6 Inertia関数

イナーシャのパラメーター値を返します。

書式

Inertia(パラメーター番号)

パラメーター

パラメーター番号

下記の整数値を指定します。

- 0: ロボットがイナーシャパラメーターをサポートするときは"1"、サポートしないときは"0"を返します。
- 1: 負荷イナーシャを返します。(単位: $\text{kg}\cdot\text{m}^2$)
- 2: 偏心量を返します。(単位: mm)

戻り値

指定した設定を実数値で返します。

参照

[Inertia](#)

ハンド制御コマンドについては、以下のマニュアルを参照してください。

"Hand機能"

Inertia関数使用例

```
Real loadInertia, eccentricity

loadInertia = Inertia(1)
eccentricity = Inertia(2)
```


3.13.7 InPos関数

ロボットの位置決め状態を返します。

書式

InPos

戻り値

- 位置決めが完了した場合: True
- ロボット動作中: False

参照

[CurPos関数](#), [FindPos関数](#), [WaitPos](#)

InPos関数使用例

```
Function main

  P0 = XY(0, -100, 0, 0)
  P1 = XY(0, 100, 0, 0)

  Xqt MonitorPosition
  Do
    Jump P0
    Wait .5
    Jump P1
    Wait .5
  Loop

Fend

Function MonitorPosition

  Boolean oldInPos, pos

  Do
    Pos = InPos
    If pos <> oldInPos Then
      Print "InPos = ", pos
    EndIf
    oldInPos = pos
  Loop

Fend
```

3.13.8 Input

ディスプレイデバイスから入力を受け取り、変数に保存します。

書式

Input 変数名 [, 変数名, 変数名,...]

パラメーター

変数名

変数名を指定します。複数の変数を指定するときは、","で区切ります。この場合の","をデリミターと呼びます。

解説

ディスプレイデバイスからデータを受け取り、指定された変数に代入します。

命令の実行時には、ディスプレイデバイスにプロンプト"?"が表示されます。データを入力後、キーボードでリターンキーを押します。

注意

■ 数値入力の規則

数値入力をする場合、デリミター以外の非数値データがあると、その非数値データと、それ以降のデータは切り捨てられます。

■ 文字列入力の規則

文字列に代入するときは、数字とアルファベットが文字として扱われます。

■ その他の Input 命令に関する規則

- 複数の変数を代入先に指定するときは、それぞれに代入する数値データは、必ずデリミター","で区切ります。
- 数値変数と文字列変数を指定することができますが、入力データ型は代入先の変数型に適合していなければなりません。

起こりやすいエラー

■ 指定された変数の数と入力データの数が合わないとき

複数の変数を指定すると、入力データの数は指定された変数の数と一致しなければなりません。命令で指定された変数の数とキーボードから受け取った数値データの数が合わないとき、エラー2505になります。

参照

Input #, [Line Input](#), Line Input #, [Print](#), [String](#)

Input使用例

下記は、簡単なInputステートメントの例です。

```
Function InputNumbers
  Integer A, B, C

  Print "Please enter 1 number"
  Input A
  Print "Please enter 2 numbers separated by a comma"
  Input B, C
```

```
Print "A = ", A
Print "B = ", B, "C = ", C
Fend
```

上記のプログラムを走らせると、次のセッションが行われます。

```
Please enter 1 number
?-10000
Please enter 2 numbers separated by a comma
?25.1, -10000
A = -10000
B = 25 C = -10000
```

3.13.9 Input

ファイル, 通信ポート, データベース, またはデバイスから、文字列か数値をデータ入力し、変数に保存します。

書式

Input #ポート番号, 変数名 [, 変数名, 変数名,...]

パラメーター

ポート番号

ファイル, 通信ポート, データベース, またはデバイスを示すID番号です。ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C) またはOpenNet (TCP/IP)ステートメントで指定した番号です。データベース番号は、OpenDBステートメントで指定した番号です。

デバイスIDは以下の数値です。

- 21 RC+
- 20 TP4

変数名

データを受け取る変数名を指定します。

解説

Input # 命令は、ポート番号で指定したデバイスから数値または文字列データを受け取り、そのデータを指定する変数に入力します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

- 数値入力の規則

数値入力をする際、デリミター以外の非数値データがあると、その非数値データと、それ以降のデータは切り捨てられます。

- 文字列入力の規則

文字列に代入するときは、数字とアルファベットが文字として扱われます。

- 最大データ長

本命令が一度にあつかえる最大データ長は、256 Byteです。ただし、データベースが対象の場合は、4096 Byteが最大データ長になります。通信ポート(TCP/IP)が対象の場合は、1024 Byteが最大データ長になります。

- その他の Input # 命令に関する規則

- 複数の変数を代入先に指定するときは、それぞれに代入する数値データは、必ずデリミター (",")または空白 (" ")で区切ります。
- 複数の文字列変数、および数値変数と文字列変数の両方を指定するときには、数値データは、必ずデリミター (",")、または空白 (" ")で区切り、文字列データは必ずデリミター ("")で区切る必要があります。
- 入力データ型は代入先の変数型に適合していなければなりません。
- コントローラー間で、通信ポートを使って文字列変数, 数値変数をやりとりする例

- 送信側 (どちらのパターンでもOKです。)

```
Print #PortNum, "$Status,", InData, OutData
Print #PortNum, "$Status", ",", InData, OutData
```

- 受信側

```
Input #PortNum, Response$, InData, OutData
```

起こりやすいエラー

- 指定された変数の数と入力データの数合わないとき

命令で指定された変数の数とデバイスから受け取った数値データの数合わないとき、エラー2505になります。

参照

[Input](#), [Line Input](#), [Line Input #](#), [Print #](#), [Read](#), [ReadBin](#)

Input #使用例

下記はInput # ステートメントを使った簡単な例です。

```
Function GetData
  Integer A
  String B$

  OpenCom #1
  Print #1, "Send"
  Input #1, A      'ポート#1から数値を取得
  Input #1, B$     'ポート#1から文字列を取得
  CloseCom #1
Fend
```

3.13.10 InputBox

入力ダイアログボックスを表示します。ユーザーがテキストに入力するか、またはボタンをクリックするまで待ち、入力した内容を返します。

書式

InputBox プロンプト, タイトル, デフォルト, 入力文字列

パラメーター

プロンプト

ダイアログボックスに表示するメッセージの文字列

タイトル

ダイアログボックスのタイトルバーに表示する文字列

デフォルト

テキストボックスにデフォルトで表示する文字列デフォルト値を設定しない場合は、空白を設定します。(" ")

入力文字列

ユーザーが入力した文字列をセットする文字列型の変数ユーザーがキャンセルをクリックした場合は、この変数に "@" がセットされます。

解説

InputBox はダイアログボックスを表示し、ユーザーが[OK]ボタン、または[キャンセル]ボタンをクリックするまで待機します。パラメーターの入力文字列に、ユーザーが入力した文字列がセットされます。

参照

[MsgBox](#)

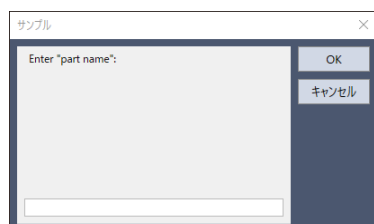
InputBox使用例

このプログラムはInputBoxステートメントの使用例です。

```
Function GetPartName$ As String
    String prompt$, title$, answer$

    prompt$ = "Enter " + Chr$(34) + "part name" + Chr$(34) + ":"
    title$ = "サンプル"
    InputBox prompt$, title$, "", answer$
    If answer$ <> "@" Then
        GetPartName$ = answer$
    EndIf
Fend
```

次の図は前記のプログラム例の結果です。



制限事項

- パラメーターのプロンプト、タイトル、およびデフォルトに、半角カンマ", "が含まれる場合、文字列が正しく表示できません。半角カンマを含まない文字列をご使用ください。

- 入力文字列が”@”の場合、キャンセルと判別できません。

3.13.11 InReal関数

2ワード (32ビット)の入力データを32ビット浮動小数点データ (IEEE754準拠)として読み込みます。

書式

InReal(ワードポート番号)

パラメーター

ワードポート番号

I/O入力ワードポートを指定します。

戻り値

入力ポートの状態をReal型実数で返します。

解説

ワードポート番号で指定した入力ワードポートから2入力ワードの値をIEEE754のReal型値として取得します。ワードポート番号パラメーターには、入力ワードラベルを使用できます。

本関数は、Wait命令、およびTill, Find, Senseの条件として使用できません。

参照

[In関数](#), [InW関数](#), [InBCD関数](#), [Out](#), [OutW](#), [OpBCD](#), [OutReal](#)

InW関数使用例

```
Real realVal  
  
realVal = InReal(32)
```


3.13.12 InsideBox関数

進入検出エリアの検出状態を返します。

書式

InsideBox(エリア番号 [, ロボット番号 | All])

パラメーター

エリア番号

状態を返す進入検出エリアの番号 (1～15の整数)を指定します。

ロボット番号

検出するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。Allを指定した場合は、1台のロボットでも進入している場合はTrueを返します。

戻り値

指定した進入検出エリアにロボットの先端位置が進入している場合はTrue、進入していない場合はFalseを返します。

参照

[Box](#), [BoxClr](#), [BoxDef関数](#), [GetRobotInsideBox関数](#), [InsidePlane関数](#)

注意

EPSON RC+ 5.0では、Wait命令と組み合わせてInsideBox関数の結果を待つことが可能でしたが、EPSON RC+ 6.0, RC+ 7.0, Epson RC+ 8.0ではWait命令と組み合わせることはできません。このような場合InsideBox関数の代わりに、GetRobotInsideBox関数を使用してください。

対応表

RC+バージョン	ロボットコントローラー	Wait	Till, Find, Sense, Trap	Printなど、左に属さないコマンド/分岐判定処理	GetRobotInsideBox関数の利用
RC+ 8.0	RC700シリーズ	不可	不可	可能	すべて可能
RC+ 8.0	RC90シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC700シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC90シリーズ	不可	不可	可能	すべて可能
RC+ 6.0	RC620	不可	不可	可能	すべて可能
RC+ 5.0	RC90シリーズ	可能	不可	可能	不可

- 不可: 利用不可能な組み合わせ
- 可能: 利用可能な組み合わせ
- すべて可能: Wait, Till, Find, Sense, Trap, Printなどの表示、分岐判定処理に利用可能

InsideBox関数使用例

下記のプログラム例は、エリア3番にロボット1が進入しているかを判断する例です。

```
Function PrintInsideBox
If InsideBox(3,1) = True Then
Print "Inside Box3"
```

```
Else  
Print "Outside Box3"  
EndIf  
Fend
```

3.13.13 InsidePlane関数

進入検出平面の検出状態を返します。

書式

InsidePlane(平面番号[, ロボット番号 | All])

パラメーター

平面番号

状態を返す進入検出平面の番号 (1～15の整数)を指定します。

ロボット番号

検出するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。Allを指定した場合は、1台のロボットでも進入している場合はTrueを返します。

戻り値

指定した進入検出平面にロボットの先端位置が進入している場合は"True"、進入していない場合は"False"を返します。

参照

[InsideBox関数](#), [GetRobotInsidePlane関数](#), [Plane](#), [PlaneClr](#), [PlaneDef関数](#)

注意

EPSON RC+ 5.0では、Wait命令と組みあわせてInsidePlane関数の結果を待つことが可能でしたが、EPSON RC+ 6.0, RC+ 7.0, Epson RC+ 8.0では、Wait命令と組みあわせることはできません。このような場合InsidePlane関数の代わりに、GetRobotInsidePlane関数を使用してください。

対応表

RC+バージョン	ロボットコントローラー	Wait	Till, Find, Sense, Trap	Printなど、左に属さないコマンド/分岐判定処理	GetRobotInsidePlane関数の利用
RC+ 8.0	RC700シリーズ	不可	不可	可能	すべて可能
RC+ 8.0	RC90シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC700シリーズ	不可	不可	可能	すべて可能
RC+ 7.0	RC90シリーズ	不可	不可	可能	すべて可能
RC+ 6.0	RC620	不可	不可	可能	すべて可能
RC+ 5.0	RC90シリーズ	可能	不可	可能	不可

- 不可: 利用不可能な組み合わせ
- 可能: 利用可能な組み合わせ
- すべて可能: Wait, Till, Find, Sense, Trap, Printなどの表示、分岐判定処理に利用可能

InsidePlane関数使用例

下記のプログラム例は、平面3番にロボット1が進入しているかを判断する例です。

```
Function PrintInsidePlane
If InsidePlane(3,1) = True Then
Print "Inside Plane3"
Else
Print "Outside Plane3"
EndIf
Fend
```

3.13.14 InStr関数

文字列からの文字列検索をし、見つかった場所を返します。

書式

InStr(文字列, 検索文字列)

パラメーター

文字列

検索される文字列を指定します。

検索文字列

検索する文字列を指定します。

戻り値

検索する文字列が見つかった場合はその場所を、見つからなかった場合は-1を返します。

参照

[Mid\\$関数](#)

InStr関数使用例

```
Integer pos
```

```
pos = InStr("abc", "b")
```

3.13.15 Int関数

実数値を整数値に変換します。実数値の小数点以下を切り捨てた値を返します。

書式

Int(数値)

パラメーター

数値

実数値を指定します。

戻り値

パラメーターで設定した実数値を整数値に変換して返します。

解説

Int(数値) は実数値の小数点以下を切り捨てた値を返します。

注意

- 0より小さい数値の場合（負の値の場合）

パラメーターが0より小さい値の場合は、負の方向に切り捨てた値を返します。（数値が-1.35の場合は、-2を返します。）

参照

[Abs関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Mod 演算子](#), [Not 演算子](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#)

Int関数使用例

次の例はコマンドウィンドウで実行します。

```
> Print Int(5.1)
5
> Print Int(0.2)
0
> Print Int(-5.1)
-6
>
```

3.13.16 Int32

Int32型の変数を宣言します。(4バイト整数型変数)

書式

Int32 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Int32は整数型変数の宣言をします。整数型変数の範囲は-2147483648～2147483647です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Int32使用例

次はInt32を使って整数型の変数を宣言するプログラム例です。

```
Function int32test
    Int32 A(10)           'Int32型の1次元配列
    Int32 B(10, 10)       'Int32型の2次元配列
    Int32 C(5, 5, 5)      'Int32型の3次元配列
    Int32 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

3.13.17 Int64

Int64型の変数を宣言します。(8バイト整数型変数)

書式

Int64 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Int64は整数型変数の宣言をします。整数型変数の範囲は-9223372036854775808～9223372036854775807です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Int64使用例

次はInt64を使って整数型の変数を宣言するプログラム例です。

```
Function int64test
    Int64 A(10)           'Int64型の1次元配列
    Int64 B(10, 10)       'Int64型の2次元配列
    Int64 C(5, 5, 5)      'Int64型の3次元配列
    Int64 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```


3.13.18 Integer

Integer型の変数を宣言します。(2バイト整数型変数)

書式

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

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Integerは整数型変数の宣言をします。整数型変数の範囲は-32768～32767です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Integer使用例

次はIntegerを使って整数型の変数を宣言するプログラム例です。

```
Function inttest
    Integer A(10)           'Integer型の1次元配列
    Integer B(10, 10)       'Integer型の2次元配列
    Integer C(5, 5, 5)      'Integer型の3次元配列
    Integer var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

3.13.19 InW関数

入力ポートの状態をワード単位で返します。ワードポートは16個の入力ビットで構成されます。

書式

InW(ワードポート番号)

パラメーター

ワードポート番号

I/Oのワードポートを指定します。

戻り値

入力ポートの状態 (0～65535のLong型整数)を返します。

注意

- リアルタイムI/Oの入力ビットを含むワードポートに関する規則

ワードポート=1は、入力ポートの状態を0～255の整数で返します。

リアルタイムI/Oの入力ビットは、反映されません。

参照

[In関数](#), [Out](#), [OutW](#)

InW関数使用例

```
Long word0  
  
word0 = InW(0)
```

3.13.20 IODef関数

指定した入出力のビット、バイト、ワードまたはI/Oラベルが定義されているかを返します。

書式

IODef (IO型, IO幅, ポート番号)

IODef (IOラベル)

パラメーター

IO型

I/Oの型を表す整数値

- 0 - 入力
- 1 - 出力
- 2 - メモリー

IO幅

ポートの幅を表す整数値: 1 (ビット), 8 (バイト), または16 (ワード)

ポート番号

ラベルを返すビット、バイト、またはワードを表す整数値

IOラベル

文字列で標準I/Oまたは、メモリーI/Oラベルを指定する文字列式です。

戻り値

指定した入出力のビット、バイト、ワード、またはI/Oラベルが定義されている場合は"True"、それ以外は "False"を返します。

参照

[IOLabel\\$関数](#), [IONumber関数](#)

IODef関数使用例

```
Integer i

For i = 0 To 15
  If IODef( 0, 1, i) = TRUE Then
    Print "Port " , i, " is defined"
  Else
    Print "Port " , i, " is undefined"
  EndIf
Next i
```

3.13.21 IOLabel\$関数

指定した入出力のビット、バイト、ワードのI/Oラベルを返します。

書式

IOLabel\$(IO型, IO幅, ポート番号)

パラメーター

IO型

I/Oの型を表す整数値

- 0 - 入力
- 1 - 出力
- 2 - メモリー

IO幅

ポートの幅を表す整数値: 1 (ビット), 8 (バイト), 16 (ワード)

ポート番号

ラベルを返すビット、バイト、またはワードを表す整数値

戻り値

ラベルを含む文字列を返します。

解説

I/Oラベルが未定義の場合、エラーが発生します。

参照

[PLabel\\$関数](#), [IONumber関数](#), [IODef関数](#)

IOLabel\$関数使用例

```
Integer i

For i = 0 To 15
  Print "Input ", i, ": ", IOLabel$(0, 1, i)
Next i
```

3.13.22 IONumber関数

指定したI/OラベルのI/Oポート番号を返します。

書式

IONumber(IO ラベル)

パラメーター

IOラベル

文字列で標準I/OまたはメモリーI/Oラベルを指定する文字列式です。

戻り値

指定したI/OラベルのI/Oポート番号 (ビット, バイト, ワード) を返します。指定したI/Oラベルが存在しない場合は、エラーが発生します。

参照

[IOLabel\\$関数](#), [IODef関数](#)

IONumber関数使用例

```
Integer IObit  
  
IObit = IONumber("myIO")  
  
IObit = IONumber ("Station" + Str$(station) + "InCycle")
```

3.14 J

3.14.1 J1Angle

ポイントのJ1Angle 属性を設定します。

書式

(1) J1Angle ポイント指定 [, 設定値]

(2) J1Angle

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

設定値を実数値で指定します。省略可能です。

結果

J1Angle属性は、RSシリーズとNシリーズのロボットでのみ使われます。

設定値パラメーターを省略すると、指定ポイントに対するJ1Angle値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ1Angle値が表示されます。

- RSシリーズ: ポイントのX座標値が"0"で、かつY座標値も"0"となる特異姿勢の第1関節角度値を指定します。特異姿勢とならないポイントでは、J1Angle属性の値は意味を持ちません。
- Nシリーズ: 第1関節, 第4関節, 第6関節の軸心が直線上にある場合、第1関節と第6関節の軸心が直線上にある場合、または第1関節と第4関節の軸心が直線上にある場合の、第1関節角度値を指定します。特異姿勢とならないポイントでは、J1Angle属性の値は意味を持ちません。

参照

[Hand](#), [J1Angle関数](#), [J1Flag](#), [J2Flag](#), [J4Angle](#), [J4Angle関数](#)

J1Angle使用例

```
J1Angle P0, 10.0  
J1Angle P(mypoint), 0.0
```

3.14.2 J1Angle関数

ポイントのJ1Angle属性を返します。

書式

J1Angle [(ポイント 指定)]

パラメーター

ポイント 指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ1Angle設定を返します。

戻り値

J1Angle属性は、RSシリーズとNシリーズのロボットでのみ使われます。

- RSシリーズ: ポイントのX座標値が"0"で、かつY座標値も"0"となる特異姿勢の第1関節角度値を実数値で返します。
- Nシリーズ: 第1関節, 第4関節, 第6関節の軸心が直線状にある場合、第1関節と第6関節の軸心が直線状にある場合、または第1関節と第4関節の軸心が直線状にある場合の第1関節角度値を実数値で返します。

参照

[Hand](#), [J1Angle](#), [J1Flag](#), [J2Flag](#), [J4Angle](#), [J4Angle関数](#)

J1Angle関数使用例

```
Print J1Angle(pick)
Print J1Angle(P1)
Print J1Angle
```

3.14.3 J1Flag

ポイントのJ1Flag 属性を設定します。

書式

(1) J1Flag ポイント指定 [, 設定値]

(2) J1Flag

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

下記のいずれかを、整数または式で指定します。省略可能です。

- RSシリーズの場合
 - 0 (/J1F0): J1 の範囲: -90 ~ +270 (単位: 度)
 - 1 (/J1F1): J1 の範囲: -270 ~ -90 または、+270 ~ +450 (単位: 度)
- C8, C12シリーズの場合
 - 0 (/J1F0): J1 の範囲: -180 ~ +180 (単位: 度)
 - 1 (/J1F1): J1 の範囲: -240 ~ -180 または、+180 ~ +240 (単位: 度)

結果

J1Flag属性は、1つのポイントに対する第1関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対するJ1Flag値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ1Flag値が表示されます。

参照

[Hand](#), [J1Flag関数](#), [J2Flag](#)

J1Flag使用例

```
J1Flag P0, 1
J1Flag P(my point), 0
```


3.14.4 J1Flag関数

ポイントのJ1Flag属性を返します。

書式

J1Flag [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ1Flag設定を返します。

戻り値

- 0: /J1F0
- 1: /J1F1

参照

[Hand](#), [J1Flag](#), [J2Flag](#)

J1Flag関数使用例

```
Print J1Flag(pick)
Print J1Flag(P1)
Print J1Flag
Print J1Flag(Pallet(1, 1))
```

3.14.5 J2Flag

ポイントのJ2Flag 属性を設定します。

書式

(1) J2Flag ポイント指定 [, 設定値]

(2) J2Flag

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

下記のいずれかを、整数または式で指定します。省略可能です。

- 0 (/J2F0): J2 の範囲: -180 ~ +180 (単位: 度)
- 1 (/J2F1): J2 の範囲: -360 ~ -180 または、+180 ~ +360 (単位: 度)

結果

J2Flag属性は、1つのポイントに対する第2関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対するJ2Flag値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ2Flag値が表示されます。

参照

[Hand](#), [J1Flag](#), [J2Flag関数](#)

J2Flag使用例

```
J2Flag P0, 1  
J2Flag P(mypoint), 0
```

3.14.6 J2Flag関数

ポイントのJ2Flag属性を返します。

書式

J2Flag [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ2Flag設定を返します。

戻り値

- 0: /J2F0
- 1: /J2F1

参照

[Hand](#), [J1Flag](#), [J2Flag](#)

J2Flag関数使用例

```
Print J2Flag(pick)
Print J2Flag(P1)
Print J2Flag
Print J2Flag(P1 + P2)
```

3.14.7 J4Angle

ポイントのJ4Angle 属性を設定します。

書式

(1) J4Angle ポイント指定 [, 設定値]

(2) J4Angle

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

設定値を実数値で指定します。省略可能です。

結果

J4Angle属性は、Nシリーズのロボットでのみ使われます。

第4関節と第6関節の軸心が直線状にある場合の、第4関節角度値を指定します。

特異姿勢にならないポイントでは、J4Angle属性の値は意味を持ちません。

設定値パラメーターを省略すると、指定ポイントに対するJ4Angle値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ4Angle値が表示されます。

参照

[Hand](#), [J1Angle](#), [J1Angle関数](#), [J4Angle関数](#)

注意

下記のようにJ4FlagとJ4Angleの両方を使用した場合、J4Angleが優先されます

```
J4Angle P0,0  
J4Flag P0,1
```

J4Angle使用例

```
J4Angle P0, 10.0  
J4Angle P(mypoint), 0.0
```

3.14.8 J4Angle関数

ポイントのJ4Angle属性を返します。

書式

J4Angle [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ4Angle設定を返します。

戻り値

第4関節と第6関節の軸心が直線上にある場合の、第4関節角度値を実数値で返します。

J4Angle属性は、Nシリーズのロボットでのみ使われます。

参照

[Hand](#), [J1Angle](#), [J1Angle関数](#), [J4Angle](#)

J4Angle関数使用例

```
Print J4Angle(pick)
Print J4Angle(P1)
Print J4Angle
```

3.14.9 J4Flag

ポイントのJ4Flag属性を設定します。

書式

(1) J4Flag ポイント指定 [, 設定値]

(2) J4Flag

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

下記のいずれかを、整数または式で指定します。省略可能です。

- 0 (/J4F0): J4 の範囲: -180 ~ +180 (単位: 度)
- 1 (/J4F1): J4 の範囲: -360 ~ -180 または、+180 ~ +360 (単位: 度)

結果

J4Flag属性は、1つのポイントに対する第4関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対するJ4Flag値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ4Flag値が表示されます。

参照

[Elbow](#), [Hand](#), [J4Flag関数](#), [J6Flag](#), [Wrist](#)

J4Flag使用例

```
J4Flag P0, 1
J4Flag P(mypoint), 0
```

3.14.10 J4Flag関数

ポイントのJ4Flag属性を返します。

書式

J4Flag [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ4Flag設定を返します。

戻り値

- 0: /J4F0
- 1: /J4F1

参照

[Elbow](#), [Hand](#), [Wrist](#), [J4Flag](#), [J6Flag](#)

J4Flag関数使用例

```
Print J4Flag(pick)
Print J4Flag(P1)
Print J4Flag
Print J4Flag(Pallet(1, 1))
```

3.14.11 J6Flag

ポイントのJ6Flag 属性を設定します。

書式

(1) J6Flag ポイント指定 [, 設定値]

(2) J6Flag

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

設定値

整数または式を指定します。

- 範囲: 0から127 (/J6F0 ~ /J6F127)
- 指定ポイントに対するJ6 の範囲は、次のとおりです。
 - $(-180 * (\text{設定値} + 1) < J6 \leq -180 * \text{設定値})$
 - $(180 * \text{設定値} < J6 \leq 180 * (\text{設定値} + 1))$

結果

J6Flag属性は、1つのポイントに対する第6関節の値の範囲を指定します。設定値パラメーターを省略すると、指定ポイントに対するJ6Flag値が表示されます。両方のパラメーターが省略されているときは、現在のロボット位置のJ6Flag値が表示されます。

参照

[Elbow](#), [Hand](#), [J4Flag](#), [J6Flag関数](#), [Wrist](#)

注意

J6Flagの範囲は機種によって異なります。

範囲については、各マニピュレーターマニュアルの動作エリアを参照してください。

J6Flag使用例

```
J6Flag P0, 1
J6Flag P(mypoint), 0
```


3.14.12 J6Flag関数

ポイントのJ6Flag属性を返します。

書式

J6Flag [(ポイント指定)]

パラメーター

ポイント指定

ポイント式で指定します。省略可能で、省略するとロボットの現在位置のJ6Flag設定を返します。

戻り値

0 ～ 127: /J6F0 ～ /J6F127

参照

[Elbow](#), [Hand](#), [Wrist](#), [J4Flag](#), [J6Flag](#)

J6Flag関数使用例

```
Print J6Flag(pick)
Print J6Flag(P1)
Print J6Flag
Print J6Flag(P1 + P2)
```

3.14.13 JA関数

関節の角度から計算したロボット座標を返します。

書式

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

パラメーター

第1関節位置～第9関節位置関節の角度 (単位: deg)を実数値で指定します。直動関節の場合は (単位: mm)で指定します。第5関節位置, 第6関節位置は、垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボット専用のパラメーターです。第7関節位置は、ジョイント型7軸ロボット専用のパラメーターです。第8関節位置, 第9関節位置は、付加軸S、およびT関節専用のパラメーターです。

補足

動作範囲外の角度を指定した場合、動作範囲外エラーが発生します。

結果

指定した関節位置で示すロボット座標を返します。

解説

関節角度でロボット座標を指定する場合に、JA関数を使用します。

JA関数により返されるポイントがロボットの特異姿勢となる場合、そのポイントに対する動作命令を実行しても、JA関数の引数として与えた関節角度に一致するとは限りません。JA関数で指定する関節角度に動作したい場合は、ロボットの特異姿勢を避ける必要があります。

[例]

```
> go ja(0,0,0,90,0,-90)
> where
WORLD: X: 0.000 mm Y: 655.000 mm Z: 675.000 mm U: 0.000 deg V: -90.000
deg W: -90.000 deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 deg 4: 0.000 deg 5: 0.000
deg 6: 0.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 0 pls 5: 0
pls 6: 0 pls
> go ja(0,0,0,90,0.001,-90)
> where
WORLD: X: -0.004 mm Y: 655.000 mm Z: 675.000 mm U: 0.000 deg V: -90.000
deg W: -89.999 deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 deg 4: 90.000 deg 5: 0.001
deg 6: -90.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 2621440 pls 5: 29
pls 6: -1638400 pls
```

参照

[AgIToPls関数, XY関数](#)

JA関数使用例

```
P10 = JA(60, 30, -50, 45)
GO JA(135, 90, -50, 90)
P3 = JA(0, 0, 0, 0, 0, 0)
```

3.14.14 JogPanel

SPEL+のプログラムからEpson RC+のジョグ&ティーチ画面を起動します。

書式

JogPanel(ジョグパネル表示, ティーチパネル表示, 動作命令パネル表示, ハンド操作パネル表示, SFreeパネル表示)

パラメーター

パラメーター	解説
ジョグパネル表示	ジョグパネルの表示有無を指定します。 ▪ True:表示 ▪ False:非表示
ティーチパネル表示	ティーチパネルの表示有無を指定します。 ▪ True:表示 ▪ False:非表示
動作命令パネル表示	動作命令パネルの表示有無を指定します。 ▪ True:表示 ▪ False:非表示
ハンド操作パネル表示	ハンド操作パネルの表示有無を指定します。 ▪ True:表示 ▪ False:非表示
SFreeパネル表示	SFreeパネルの表示有無を指定します。 ▪ True:表示 ▪ False:非表示

解説

SPEL+のタスクからEpson RC+のジョグ&ティーチ画面を起動/表示するときに、本コマンドを使います。

タスクは、画面が閉じられるまで一時停止します。



Epson RC+の画面からロボットコマンドを実行するときは、その画面表示中にロボットを制御する他のタスクがないことを確認してください。

他にロボットを制御するタスクが起動している場合は、エラーになります。

参照

[RunDialog](#), [ToolWizard関数](#), [TeachPoint](#)

JogPanel使用例

```
JogPanel(True, True, True, True, True)
```

3.14.15 Joint

現在のロボットの位置を、関節の座標系で表示します。

書式

Joint

参照

[Pulse, Where](#)

Joint使用例

```
>joint  
JOINT:  1: -6.905 deg 2: 23.437 deg 3: -1.999 mm  4: -16.529 deg  
>
```

3.14.16 JointAccuracy

関節精度補正の補正值の設定と表示をします。

書式

(1) JointAccuracy 関節番号, 第1設定値, 第2設定値, 第3設定値, 第4設定値

(2) JointAccuracy 関節番号

パラメーター

関節番号

関節番号を指定します。

第1設定値

第1設定値 (整数)を、数値で指定します。範囲は0から2000までです。

第2設定値

第2設定値 (整数)を、数値で指定します。範囲は0から999までです。

第3設定値

第3設定値 (整数)を、数値で指定します。範囲は0から2000までです。

第4設定値

第4設定値 (整数)を、数値で指定します。範囲は0から999までです。

マニピュレーターごとに関節精度補正が有効な関節は異なり、有効な関節以外ではエラーになります。有効な関節については、以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

戻り値

(2)の場合は、関節番号に対応する現在の関節精度補正の補正值を表示します。

解説

JointAccuracyは、指定した関節の補正值を設定します。適切に補正值を設定することにより、軌跡精度をあげることができます。

注意

- JointAccuracyは、必要な時以外、絶対に変更しないでください

JointAccuracyは、工場出荷時に精密に設定されています。不必要に この値を変更すると軌跡精度の悪化を引き起こします。JointAccuracyはキャリブレーションウィザードを実行した場合に自動的に設定されるため、必要な時以外はJointAccuracyを絶対に変更しないようにしてください。

- CalibとHofsの実行

JointAccuracyが設定された状態でCalib, Hofsコマンドを実行すると、変更があった関節について関節精度補正の補正值が "0" になります。JointAccuracyの補正值を変更せずに、Hofsの値を変更するには、HofsJointAccuracyを実行してください。

参照

[HofsJointAccuracy](#), [Calib](#), [Hofs](#)

JointAccuracy使用例

以下は、コマンドウィンドウでの簡単な使用例です。第1関節に対して、第1設定値 "1000"、第2設定値 "420"、第3

設定値 "100"、第4設定値 "240"として関節精度補正の補正値を設定しています。その後、現在の第1関節の関節精度補正の補正値を表示します。

```
> JointAccuracy 1, 1000, 420, 100, 240
```

```
> JointAccuracy 1  
1000, 420, 100, 240  
>
```

3.14.17 JointAccuracy関数

関節精度補正の補正值の表示をします。

書式

JointAccuracy(関節番号, 設定番号)

パラメーター

関節番号

関節番号を指定します。

設定番号

表示する補正值を以下の定数または整数値(1~4)で指定します。

定数	値	
JAC_PARAM1	1	第1設定値
JAC_PARAM2	2	第2設定値
JAC_PARAM3	3	第3設定値
JAC_PARAM4	4	第4設定値

戻り値

指定した関節の設定番号に対応する関節精度補正の補正值(整数値)を返します。

参照

[JointAccuracy](#)

JointAccuracy関数使用例

下記はJointAccuracy関数を使ったプログラムの例です。

```
Function main
    '第2関節の精度補正值を設定する
    JointAccuracy 2, 32, 873, 6, 437
    '指定された関節の精度補正值を表示する
    DisplayJointAccuracy(2)
Fend

Function DisplayJointAccuracy(numJoint As Integer)
    Integer i
    Print "Joint =", numJoint, ", JointAccuracy settings:"
    For i = 1 To 4
        Print "  Param ", i, " = ", JointAccuracy(numJoint, i)
    Next i
Fend

[出力結果]
Joint = 2, JointAccuracy settings:
Param 1 = 32
Param 2 = 873
Param 3 = 6
Param 4 = 437
```


3.14.18 JRange

指定関節の許容動作エリアをパルス値で設定します。

書式

JRange 関節番号, 下限パルス値, 上限パルス値

パラメーター

関節番号

JRangeが指定する関節番号を1～9の整数値で設定します。付加軸のS軸は8、T軸は9になります。

下限パルス値

指定関節の動作範囲の下限パルス値を、式または数値で指定します。

上限パルス値

指定関節の動作範囲の上限パルス値を、式または数値で指定します。

解説

指定関節の動作範囲を上限パルス値と下限パルス値で設定します。Rangeコマンドでは全関節の動作範囲の指定が必要ですが、JRangeコマンドでは、1関節ごとに動作エリアを設定するので、パラメーターの数が少なくてすみす。設定した動作エリアを確認するには、Range コマンドを使います。

注意

- 下限パルス値 $\leq$ 上限パルス値を守ってください

JRangeコマンドの下限パルス値は、上限パルス値を上回らないように設定してください。下限パルス値が上限パルス値を上回るとエラーになり、動作コマンドの実行ができなくなります。

- Jrangeの設定値の変更

JRangeで設定された値は、一度設定されると、Rangeまたは JRangeコマンドで変更しない限り保持されます。電源オフでは、JRangeで設定した制限値は変更されません。

- 動作エリアの上下限

各マニピュレーターの型によって動作エリアの上限パルス値は異なりますので、エリアの上限パルス値の設定については、以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

参照

[Range, JRange関数](#)

JRange使用例

下記は、コマンドウィンドウからの操作例です。

```
> JRange 2, -6000, 7000      '第2関節の動作範囲を定義
```

```
> JRange 1, 0, 7000         '第1関節の動作範囲を定義
```

3.14.19 JRange関数

指定した関節の許容動作エリアパルス値 (レンジ設定値) を返す関数です。

書式

JRange(関節番号, パラメーター番号)

パラメーター

関節番号

参照する関節番号 (1～9の整数) を、式または数値で指定します。付加軸のS軸は8、T軸は9になります。

パラメーター番号

下記の2つの値のいずれかを、整数値で指定します。

- 1: 下限パルス値を指定
- 2: 上限パルス値を指定

戻り値

指定関節のレンジ設定値 (整数値、単位: パルス) を返します。

参照

[Range](#), [JRange](#)

JRange関数使用例

```
Long i, oldRanges(3, 1)

For i = 0 To 3
    oldRanges(i, 0) = JRange(i + 1, 1)
    oldRanges(i, 1) = JRange(i + 1, 2)
Next i
```

3.14.20 JS関数

Jump Sense, Jump3 Sense, Jump3CP Sense, JumpTLZ Sense実行後、Sense入力が成立したかどうかを返す関数です。

書式

JS

戻り値

"True"または"False"を返します。

- True: Sense入力条件が成立してアームが目標座標上方で停止した場合、JSは"True"を返します。
- False: アームがJump命令で設定された目標位置に到達し正常な動作を完了すると、JSは、"False"を返します。

解説

JSは、JumpやSense命令とセットで用います。JS命令の目的は、Jump命令が指示した動作中に（Senseで設定した）入力条件が成立したかどうかを返します。入力条件が成立するとJSは"True"を返します。入力条件が成立せず、アームが目標位置に到達するとJSは、"False"を返します。

JSは、単に状態を確認する命令で、動作を発生させたり動作中に何かの入力をチェックしたりする機能はありません。動作を指示するにはJump命令を用い、場合によって、そのJump命令によって起こる動作中に特定の入力をチェックするために、Sense命令を用います。

注意

- JS は直前に使用したJump命令, Jump3命令, Jump3CP, JumpTLZ命令とのみ連携します

JSは、直前に使用したJump命令の入力チェック (Sense命令によって指示されます)のみを行います。次のJump命令が起動すると、JS命令は、その新しいJump命令についての状態を返します。最初のJump命令に対する状態データは、返すことはできません。状態チェックが必要なJump命令には、必ずその直後にJSチェックを行うようにします。

参照

[JT関数](#), [Jump](#), [Jump3](#), [Jump3CP](#), [JumpTLZ](#), [Sense](#)

JS関数使用例

```
Function SearchSensor As Boolean
    Sense Sw(5) = On

    Jump P0
    Jump P1 Sense
    If JS = True Then
        Print "Sensor was found"
        SearchSensor = True
    EndIf
Fend
```

3.14.21 JT関数

直前のJump, Jump3, Jump3CP, JumpTLZ動作の結果を返す関数です。

書式

JT

戻り値

下記の形式のLong型数値のビットをセット、またはクリアします。

- Bit 0: 上昇動作開始時、または上昇動作量0の場合は1
- Bit 1: 水平動作開始時、または水平移動量0の場合は1
- Bit 2: 下降動作開始時、または下降動作量0の場合は1
- Bit 16: 上昇動作完了時、または上昇動作量0の場合は1
- Bit 17: 水平動作完了時、または水平動作量0の場合は1
- Bit 18: 下降動作完了時、または下降動作量0の場合は1

解説

直前に実行されたJump命令が、Sense, Till, Abortなどにより指定された動作停止条件を成立したかどうかを調べることができます。

参照

[JS関数](#), [Jump](#), [Jump3](#), [Jump3CP](#), [JumpTLZ](#), [Sense](#), [Till](#)

JT関数使用例

```
Function SearchTill As Boolean

    Till Sw(5) = On

    Jump P0
    Jump P1 Till
    If (JT And &H4) = &H4 Then
        Print "Motion stopped during descent"
        SearchTill = TRUE
    EndIf
Fend
```

3.14.22 JTran

原点復帰を必要としない1関節のみのPTP動作を行います。

書式

JTran 関節番号, 移動量

パラメーター

関節番号

動作する関節番号を整数値で指定します。付加軸のS軸は8、T軸は9になります。

移動量

実数を指定します。回転関節では、(単位: deg)、直動関節では、(単位: mm)で指定します。

解説

JTranは、現在位置から、指定した関節のみを指定量動作させます。

参照

[Go](#), [Jump](#), [Move](#), [PTran](#)

JTran使用例

```
JTran 1, 20
```

3.14.23 Jump

アームを、現在位置から指定位置まで、ゲートモーション (まず垂直方向に上昇し、それから水平移動し、最後に垂直方向に下降する、門型動作) によりPTP動作します。

書式

Jump 目標座標 [Cアーチ番号] [LimZ [Z座標値]] [CP] [{Sense|Till|Find}] [!並列処理!] [SYNC]

パラメーター

目標座標

ポイントデータで目標位置を指定します。

アーチ番号

アーチ番号は、Jumpにおけるアーチモーションの形状に、どのArch テーブルの設定を使うか指定します。

アーチ番号は、頭にアルファベット文字"C"をつけてください。有効値は、C0～C7です。省略可能です。

Z座標値

Jump命令による動作中に第3関節が移動し得る最大値 (Z制限値)、あるいは、JumpによるZ成分の高さシーリング値と考えてください。有効な第3関節の座標値ならば、どの値でもかまいません。省略可能です。

CP

パスモーションを指定します。省略可能です。

Sense、TillまたはFind

Sense、TillまたはFind式を記述します。省略可能です。

```
Sense | Till | Find
Sense Sw(式) = {On | Off}
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

!並列処理!

Jump命令による動作中に、I/Oや、その他のコマンドを実行するための並列処理ステートメントを追加することができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

Jump命令は、いわゆる「アーチモーション (アーチ型動作)」によってアームを現在位置から目標座標まで移動させます。いわば、3つの動きを1回で行うステートメントと考えることができます。例えば、アーチ番号が定義されていれば、1回のJump命令で次の3つの動作を行います。

1. まず、第3関節だけが、Jumpコマンド中のアーチ番号が演算するZ軸の高さまで動作します。
2. 次に、アームは、LimZが指定するZ制限位置に達するまでZ軸方向に上昇しながら、目標座標に水平移動します。それから、第1関節、第2関節、第4関節のそれぞれの動作を行いながらZ軸方向に下降を始めます。アームは最終的なX, Y, U座標位置が得られるまで動作します。
3. アームは、ターゲットのZ座標位置が得られるまでZ軸方向のみの移動を行い、目標座標が得られたところでJump命令が終了します。

目標座標 (移動の目的位置) は、Jump命令の中で指定することができないので、Jump命令が実行される前にティーチングしておく必要があります。Jump移動の加速と減速は、Accelによって行います。また、移動の速度は、Speedによって行います。

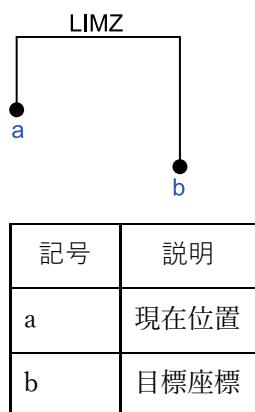
垂直6軸型ロボット (Nシリーズを含む) に対してJumpを実行することはできません。Jump3を使用してください。

■ CPについて

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

■ アーチ番号について

Jumpのアーチの型は、Jumpコマンド中で指定されるアーチ番号によって変更することができます。これによって、第1関節, 第2関節, 第4関節の各関節が動く前に、どれだけZ軸方向に動かしておきたいか決めることができます。Jump命令で有効に使えるアーチ番号値は、C0～C7までの値です。C0～C6までの値に対するArchテーブル値の設定は、ユーザーがArchコマンドによって定義することができるようになっています。しかし、C7は常に、「ゲートモーション」を定義しています。「ゲートモーション」とは、ロボットが、第1関節, 第2関節, 第4関節の各関節を動かす前に、第3関節だけを先に、LimZで定義される座標位置まで移動させる動きのことです。この「ゲートモーション」では、LimZが定義するZ制限値まで移動してから初めて、第1関節, 第2関節, 第4関節の各関節の動作が始まります。第1関節, 第2関節, 第4関節の各関節が、それぞれの最終目標座標位置まで移動すると、第3関節が、目標座標で定義される最終Z座標位置を目指して下降します。次の図は「ゲートモーション」の動きを表わします。



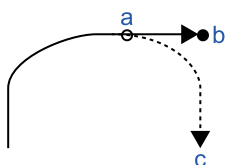
■ LimZについて

LimZ Z座標値は、現在設定されているローカル座標系の水平移動面におけるZ座標の最高値を指定します。指定されたアーチ設定によっては、LimZ値に達する前に、第1関節、第2関節、第4関節の各関節が動き始めることもあるかも知れませんが、LimZ値は、常に、その移動におけるZ座標方向の上限値を定義しています。もし、LimZパラメーターを省略すると、その直前(最後)にLimZによって指定された最高値が準用されます。

LimZによって指定される高さ方向の制限値は、ローカルロボット座標系でのZ座標値です。つまり、それは、ArmやTool座標のZ座標値ではありません。したがって、作業高の異なるツールやハンドを使用する場合は、十分注意し、必要な処置を行ってください。

■ Senseについて

省略できるパラメーターの1つです。Senseは、第3関節が最終の下降動作を行う前に、入力条件またはメモリI/O条件をチェックしたいときなどに使います。問題がなければ、ロボットのアームは、目標座標の上(最終的に第3関節の動作を残すだけになっている位置)に停止して、このコマンド(Jump)の実行が終了したことになります。ただし、Senseで指定した条件を検知しても、ロボットのアームは、すぐには停止しないので注意してください。



記号	説明
a	条件チェック
b	動作終了
c	目標座標

JSやStatなどのコマンドを組み合わせ、Sense条件が成立してロボットのアームが目標座標の前で停止したか、あるいは、Sense条件が成立せずロボットのアームはそのまま目標座標に停止したか、を確認することができます。

■ Tillについて

オプションのTillを使って、Jumpを実行する前に、ロボットを減速して停止させる条件を設定することができます。これは、I/O入力のうち1つ、あるいは、メモリーI/Oの1つをチェックする、という条件を含むことで設定できます。これには、SwまたはMemSw関数を使います。あらかじめ設定された条件にしたがって、入力がオンかオフかをチェックして、アームを減速し、停止させることができます。

Stat関数を使うことにより、Till条件が成立してこのコマンドが実行されたか、あるいは、Till条件が成立しないでロボットが目標座標に停止したかを確認することができます。

注意

- Jumpは垂直6軸型ロボット (Nシリーズを含む)には使用できません。

Jump3またはJump3CPを使用してください。

- アーチ番号 パラメーターを省略する場合

アーチ番号パラメーターの設定を省略すると、Jumpコマンド実行時のデフォルトArch値は、C7となります。上記でも説明されているとおり、Arch値がC7の場合、「ゲートモーション」移動(上記参照)になります。

- Jump とJump3, Jump3CP の違い

Jump3とJump3CPは、垂直6軸型ロボット (Nシリーズを含む)で使用することができますが、Jumpは使用することができません。水平多関節型ロボット (RSシリーズを含む)で、Z軸方向に上昇と下降を行う場合は、Jumpを使用すると動作時間が短縮できます。Jump3の接近動作や退避動作は、Z軸以外の方向に行うこともできます。

- Jump とGo の違い

JumpとGoの、最も重要な違いは、Goの場合、全関節の動きがシンクロしており、各関節は同時に動き始め、同時に停止します。Jumpでは、動作の始めと終わりに垂直方向だけの第3関節移動があります。デバイスを装着、配置というような作業にはこのコマンドをお勧めします。

- Jumpにおける減速停止

Jumpでは、アームは必ず減速しながら目標座標に停止します。

- Jumpにおける適正速度と加速指示

Jump動作におけるロボットの速度と加減速は、それぞれSpeedとAccelによって設定します。SpeedとAccelは、Jump, Goなど、あくまでもポイントからポイントへ動作する場合についてのみ設定できる命令であることに注意してください。例えば、MoveやArcのような直線および円弧補間動作を行う命令の場合は、SpeedS やAccelSの命令を使用してください。

また、Jumpの場合は、第3関節の上昇移動、第4関節の回転を含む水平動作、そして第3関節の下降について、速度と加減速を別々に設定することができます。

■ JumpのPass機能

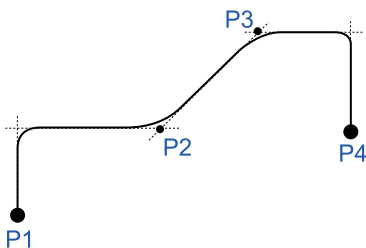
下降動作量が0のJumpにCPパラメーターをつけた場合、そのJumpの水平動作が減速停止することなく、直後に続くPTP動作と滑らかに接続できます。

また、直前のPTP動作命令にCPパラメーターをつけた場合、上昇動作量が0のJumpは、そのPTP動作が減速停止することなく、Jumpの水平動作と滑らかに接続できます。

これは、通常のJumpの水平動作 (1つのPTP動作) を、いくつかのPTP動作が滑らかに接続する動作に置きかえたい場合に便利です。

[例]

```
Go P1
Jump P2 :Z(-50) C0 LimZ -50 CP
Go P3 :Z(0) CP
Jump P4 C0 LimZ 0
```



アーチ使用時の重要事項

アーチモーションは、第3関節の上昇または下降動作と、横方向動作の合成を軌跡制御上で行うため、実際の軌跡を保証できるものではありません。動作速度やアームの動き方によってその軌跡は変化します。実際の軌跡は、作業で使用する実速度と姿勢で確認してください。

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

起こりやすいエラー

- LimZ値の設定が低すぎる場合

第3関節のアーム位置が、LimZの設定値よりも高い位置にある状態で、Jumpが実行されると、エラー4005になります。

参照

[Accel](#), [Arc](#), [Arc3](#), [Arch](#), [Go](#), [JS関数](#), [JT関数](#), [LimZ](#), [P# \(2. ポイント指定\)](#), [Pulse](#), [Sense](#), [Speed](#), [Stat関数](#), [Till](#)

Jump使用例

次の例は、ポイントP0からP1へ、単純なPTP動作をしたあと、P0へJumpで戻るものです。プログラム後半で、ア

ームはJumpを実行し、入力ビット4がオンにならないければ下降動作を行い、P1に移動します。入力ビット4がオンの場合、下降動作を行いません。

```
Function jumptest
  Home
  Go P0
  Go P1
  Sense Sw(4) = On
  Jump P0 LimZ -10
  Jump P1 LimZ -10 Sense '入力4をチェック
  If JS = True Then
    Print "Input #4 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P1."
  Else
    Print "The move to P1 completed successfully."
    Print "Input #4 never came on during the move."
  EndIf
Fend

> Jump P10+X50 C0 LimZ-20 Sense !D50;On 0;D80;On 1!
```

コマンドウィンドウからの操作例です。

```
> Jump P0 'P0にジャンプ動作します
> Jump P0 C0 'P0にアーチ番号C0で設定されたアーチモーションでジャンプ動作します
> Jump P0 LimZ -10 'P0に、Z制限値-10mmまででジャンプ動作します
> Jump P0 !D0; On 1; D50; Off 1! 'P0にジャンプ動作します。動作の移動量が50%になるまでは出力の1ビット目をOnし、50%以降は1ビット目をOffします。
```

入力の1ビット目がOnになったら、Jump命令を停止して次の処理に進みます。

```
Function main
  (中略)
  Till Sw(1) = On
  Jump P0 C0 CP Till
  (中略)
Fend
```

3.14.24 Jump3, Jump3CP

アームを3次元ゲート動作で移動します。

Jump3は、2つのCP動作と1つのPTP動作の組み合わせです。

Jump3CPは、3つのCP動作の組み合わせです。

書式

(1) Jump3 退避座標, 接近開始座標, 目標座標 [, Cアーチ番号] [, CP] [, LJM [, 姿勢フラグ選択]] [, Sense | Till | Find] [, !並列処理!] [, SYNC]

(2) Jump3CP 退避座標, 接近開始座標, 目標座標 [, ROT] [, Cアーチ番号] [, CP] [, LJM [, 姿勢フラグ選択]] [, Sense | Till | Find] [, !並列処理!] [, SYNC]

パラメーター

退避座標

現在位置より上の退避点をポイント指定します。

接近開始座標

目標座標より上の接近開始点をポイント指定します。

目標座標

動作の到達する目標座標をポイント指定します。

ROT

ツール姿勢変化を優先させて、動作の速度、加減速度を決めます。省略可能です。

アーチ番号

アーチ番号は、Jump3命令のアーチ型動作を決める、アーチテーブルを指定します。アーチ番号の頭には、必ず大文字「C」をつけてください。(有効値はC0～C7です。) アーチ番号は省略可能です。

CP

パスモーションを指定します。省略可能です。

LJM

待避座標, 接近座標, 目標座標をLJM関数で変換します。省略可能です。

姿勢フラグ選択

LJM関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。

Sense | Till | Find

Sense、TillまたはFind式を記述します。省略可能です。

```
Sense | Till | Find
Sense Sw(式) = {On | Off}
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

!並列処理!

Jump3、Jump3CP命令に並列処理ステートメントを追加し、動作中にI/Oやその他のコマンドを実行できます。省略可能です。

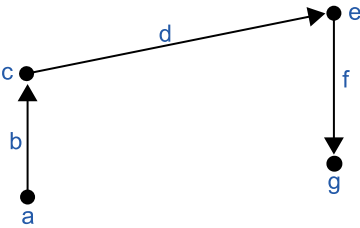
SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

アームを現在位置から目標座標まで3次元ゲート動作で移動します。3次元ゲート動作は、退避動作、スパン動作、そして接近動作から成り立っています。現在位置から退避座標までの退避動作はCP動作です。退避座標から接近

開始座標までのスパン動作は、Jump3の場合はPTP動作で、Jump3CPの場合はCP動作です。接近開始座標から目標座標までの接近動作は、CP動作になります。

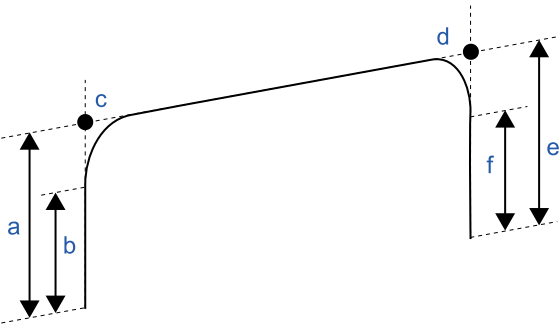


記号	説明
a	現在位置
b	退避動作 CP
c	退避座標
d	スパン動作 PTP/CP
e	接近開始座標
f	接近動作 CP
g	目標座標

アーチ動作は、アーチ番号の設定によって行います。

Jump3およびJump3CP のアーチ動作は下図に示すとおりです。

退避距離はArch上昇距離よりも長く、接近距離はArch下降距離よりも長くしてください。



記号	説明
a	退避距離
b	Arch上昇距離
c	退避座標
d	接近開始座標
e	接近距離
f	Arch下降距離

Jump3CPの速度と加減速度はそれぞれSpeedSとAccelSの設定値を使用します。速度と加減速度の関係については、"注意"の"Jump3, Jump3CPをCPとともに用いる"を参照してください。ただし、ROT修飾パラメーターを指定しているときの速度と加減速度はそれぞれSpeedRとAccelRの設定値を使用します。この場合、SpeedSとAccelSの設定値は無効となります。

ロボットのツール先端位置を、特定の座標に固定したままツール姿勢のみを変化させようとした場合や、ツール先端の移動距離に対してツール姿勢変化が大きい場合、ツール姿勢変化速度が著しく速くなる恐れがあります。これを防ぐため、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働きます。

CP動作時のツール姿勢変化速度の上限値を手動で設定したい場合は、SpeedRLimitationをオンにしてください。SpeedRLimitationをオンにした場合、CP動作時のツール姿勢変化速度が設定したSpeedRを超える場合は、ツール姿勢変化速度がSpeedRになるように動作速度を制限します。ツール姿勢変化速度が設定したSpeedRを超えない場合は、設定したSpeedSで動作します。ツール姿勢変化速度の上限値はSpeedRであらかじめ設定してください。

注意

- LimZは、Jump3 およびJump3CPには影響しません。

スパン動作は、必ずしも座標系のZ軸に垂直とは限らないため、LimZは、Jump3およびJump3CPには影響しません。

- Jump3のスパン動作はPTP動作です。

PTP動作はその軌跡が予測しにくいいため、ロボット本体や周辺装置との干渉に十分注意してください。

- Jump3, Jump3CPをCPとともに用いる

CPパラメーターを使うと、動作命令は減速開始と同時に制御を次のステートメントに移します。これは、ユーザーがいくつかの動作命令をつなげ、連続した動作を一定の速度で行わせたいときに便利です。CP指定なしのJump3命令, Jump3CP命令では、アームは必ず減速して、指定された目標座標に停止します。

- Jump3のPass機能

接近動作量が0のJump3にCPパラメーターをつけた場合、そのJump3のスパン動作が減速停止することなく、直後に続くPTP動作と滑らかに接続できます。

また、直前のPTP動作命令にCPパラメーターをつけた場合、退避動作量が0のJump3は、そのPTP動作が減速停止することなく、Jump3のスパン動作と滑らかに接続できます。

これは、通常のJump3のスパン動作 (1つのPTP動作) を、いくつかのPTP動作が滑らかに接続する動作に置きかえたい場合に便利です。

- Jump3CPのPass機能

接近動作量が0のJump3CPにCPパラメーターをつけた場合、そのJump3CPのスパン動作が減速停止することなく、直後に続くCP動作と滑らかに接続できます。

また、直前のCP動作命令にCPパラメーターをつけた場合、退避動作量が0のJump3CPは、そのCP動作が減速停止することなく、Jump3CPのスパン動作と滑らかに接続できます。

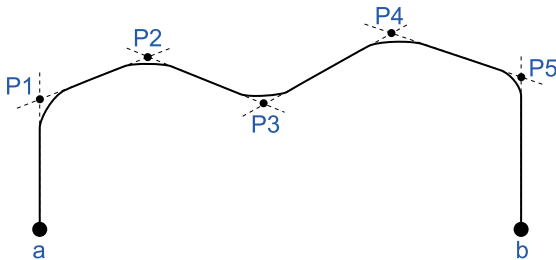
これは、通常のJump3CPのスパン動作 (1つのCP動作) を、いくつかのCP動作が滑らかに接続する動作に置きかえたい場合に便利です。

(例1)

```
Jump3 P1,P2,P2 CP
Go P3,P4 CP
Jump3 P4,P5,P5+tlz(50)
```

(例2)

```
Jump3CP P1,P2,P2 CP
Move P3,P4 CP
Jump3CP P4,P5,P5+tlz(50)
```



- Jump3, Jump3CPをLJMとともに用いる

LJMパラメーターを使うと、LJM関数を使用するプログラムを簡略化できます。

例えば、

```
P11 = LJM(P1, Here, 2)
P12 = LJM(P2, P11, 2)
P13 = LJM(P3, P12, 2)
Jump3 P11, P12, P13
```

という4行のプログラムを

```
Jump3 P1, P2, P3 LJM 2
```

という1行のプログラムに置き換えることができます。

LJMパラメーターは垂直6軸型ロボット (Nシリーズを含む)、およびRSシリーズロボットで有効です。

Jump3CPはスパン動作が直線 (CP)動作のため途中で手首姿勢を切り替えることができません。そのため手首姿勢の切り替えの可能性があるLJM関数の姿勢フラグ選択 (LJM 1)は使用しないでください。

アーチ使用時の重要事項

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

- 同じ位置で、同じ[Cアーチ番号]を持つJump3命令を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- 低速動作時より高速動作の方が、合成なしの退避移動量が増え、合成なしの接近移動量が減る傾向にあります。期待した移動距離が出ない場合は、速度や減速度を下げるか、接近距離を長めに設定してください。
- 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。

起こりやすいエラー

- 退避動作 (接近動作) とスパン動作で、主に動作する関節が同じ場合

Jump3, Jump3CPコマンドでアーチモーションを実行中に、異常加速度エラーが発生することがあります。これは、退避動作 (接近動作) とスパン動作で、主に動作する関節が同じ場合に特に顕著となります。このような場合は、Jump3の場合はAccel命令で、Jump3CPの場合はAccelS命令でスパン動作の加減速度を下げることで回避してください。また、動作姿勢によっては、AccelS命令で退避動作 (接近動作) の加減速度を下げることで有効な場合もあります。

参照

[Accel](#), [Arc](#), [Arc3](#), [Arch](#), [Go](#), [JS関数](#), [JT関数](#), [P# \(2. ポイント指定\)](#), [Pulse](#), [Sense](#), [Speed](#), [Stat関数](#), [Till](#)

Jump3使用例

'スカラロボットのJumpのように動作する垂直6軸型ロボット (Nシリーズを含む) の動作

```
Jump3 Here :Z(100), P3 :Z(100), P3
```

'Zツール座標を使用した退避動作と接近動作

```
Jump3 Here -TLZ(100), P3 -TLZ(100), P3
```

'Zベース座標を使用した退避動作とZツール座標を使用した接近動作

```
Jump3 Here +Z(100), P3 -TLZ(100), P3
```

Tool1で退避動作を行い、Tool2で接近動作を行う例

```
Arch 0,20,20
```

```
Tool 1
```

```
Go P1
```

```
P2 = P1 -TLZ(100)
```

```
Tool 2
```

```
Jump3 P2, P3-TLZ(100), P3 C0
```

3.14.25 JumpTLZ

アームを3次元ゲート動作で移動します。

Nシリーズ専用のコマンドです。

JumpTLZは、2つのCP動作と1つのPTP動作の組み合わせです。

書式

JumpTLZ

目標座標, TLZ方向移動量[, Cアーチ番号][, CP][, LJM[, 姿勢フラグ選択]][, Sense | Till | Find][, !並列処理!][, SYNC]

パラメーター

目標座標

動作の到達する目標座標をポイント指定します。

TLZ方向移動量

Tool座標のZ方向に移動する量を指定します。単位はmmです。現在設定されているTool番号のTool座標を使用します。

アーチ番号

アーチ番号は、JumpTLZ命令のアーチ型動作を決める、アーチテーブルを指定します。アーチ番号の頭には、必ず大文字「C」をつけてください。(有効値はC0からC7です。) アーチ番号は省略可能です。

CP

パスモーションを指定します。省略可能です。

LJM

目標座標をLJM関数で変換します。省略可能です。

姿勢フラグ選択

LJM関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。

Sense | Till | Find

Sense、TillまたはFind式を記述します。省略可能です。

```
Sense | Till | Find
Sense Sw(式) = {On | Off}
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

!並列処理!

Jump3, Jump3CP命令に並列処理ステートメントを追加し、動作中にI/Oやその他のコマンドを実行できます。省略可能です。

SYNC

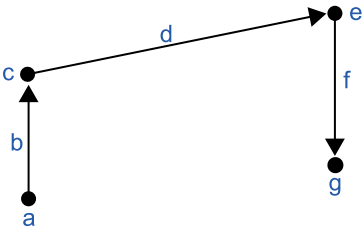
動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

アームを現在位置から目標座標まで3次元ゲート動作で移動します。3次元ゲート動作は、退避動作、スパン動作、そして接近動作から成り立っています。現在位置から退避座標までの退避動作はCP動作です。退避座標から接近開始座標までのスパン動作はPTP動作になります。

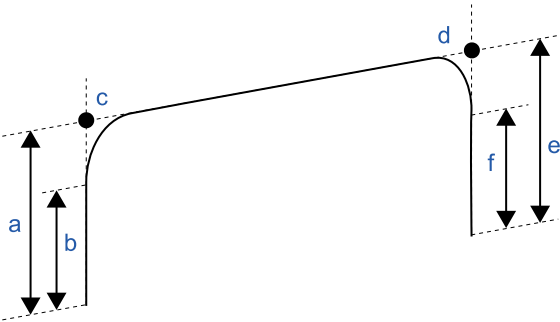
退避座標は、現在位置からTool座標のZ方向にTLZ方向移動量だけ移動した位置になります。退避座標のロボットの姿勢は、現在位置と同じになります。(特異点や特異点近傍を通過した場合のみ、ロボットの姿勢が同じにならないことがあります。)

接近開始座標は、退避座標からTool座標のX, Y方向に目標座標までの移動量だけ移動した位置になります。退避座標のU, V, Wの座標とロボットの姿勢は、目標位置と同じになります。(特異点や特異点近傍を通過した場合のみ、ロボットの姿勢が同じにならないことがあります。)



記号	説明
a	現在位置
b	退避動作 CP
c	退避座標
d	スパン動作 PTP
e	接近開始座標
f	接近動作 CP
g	目標座標

アーチ動作は、アーチ番号の設定によって行います。退避距離はArch上昇距離よりも長く、接近距離はArch下降距離よりも長くしてください。



記号	説明
a	退避距離
b	Arch上昇距離
c	退避座標
d	接近開始座標
e	接近距離
f	Arch下降距離

注意

- LimZ は、JumpTLZには影響しません。

スパン動作は、必ずしも座標系のZ軸に垂直とは限らないため、LimZ は、JumpTLZには影響しません。

- JumpTLZのスパン動作はPTP動作です。

PTP動作はその軌跡が予測しにくいいため、ロボット本体や周辺装置との干渉に十分注意してください。

- JumpTLZとJump3の違い

JumpTLZとJump3は下記の点で異なります。

- JumpTLZ:
 - 退避座標は、現在位置からTool座標のZ方向に移動した位置以外にすることはできません。
 - 接近座標は、目標座標からTool座標のZ方向以外に移動させることはできません。
 - また、接近距離を指定することはできません。
 - 退避座標, 接近座標, 目標位置で異なるTool座標を選択することはできません。(Tool1で退避動作を行い、Tool2で接近動作を行うとことはできません)
- Jump3:
 - 退避座標の位置を任意に指定できます
 - 接近座標の位置を任意に指定できます
 - 退避座標, 接近座標, 目標位置で異なるTool座標を選択することができます。(Tool1で退避動作を行い、Tool2で接近動作を行うとことが可能です)

- JumpTLZが使用できる機種

Nシリーズのみ使用可能です。

アーチ使用時の重要事項

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

- 同じ位置で、同じ[Cアーチ番号]を持つJumpTLZ命令を実行しても、高速動作時と比較して低速時の軌跡は低くなります。したがって、高速で障害物に衝突しないことを確認しても、低速で動作すると衝突する可能性がありますので注意してください。
- 低速動作時より高速動作の方が、合成なしの退避移動量が増え、合成なしの接近移動量が減る傾向にあります。期待した移動距離が出ない場合は、速度や減速度を下げるか、接近距離を長めに設定してください。
- 同じ距離の動作であっても、アームの動き方によって軌跡は変化します。

起こりやすいエラー

- 退避動作 (接近動作) とスパン動作で、主に動作する関節が同じ場合

JumpTLZコマンドでアーチモーションを実行中に、異常加速度エラーが発生することがあります。これは、退避動作 (接近動作) とスパン動作で、主に動作する関節が同じ場合に特に顕著となります。このような場合は、Accel命令でスパン動作の加減速度を下げることで回避してください。また、動作姿勢によっては、AccelS命令で退避動作 (接近動作) の加減速度を下げることで有効な場合もあります。

参照

[Accel](#), [Arc](#), [Arc3](#), [Arch](#), [Go](#), [JS関数](#), [JT関数](#), [P# \(2. ポイント指定\)](#), [Pulse](#), [Sense](#), [Speed](#), [Stat関数](#), [Till](#)

JumpTLZ使用例

現在位置から Tool座標のZ方向に100 mm上昇し、目的地 (P0)に移動させる場合:

```
JumpTLZ  P0, -100
```

3.15 L

3.15.1 LatchEnable

R-I/O入力によるロボット位置のラッチ機能を有効、または無効にします。

書式

LatchEnable { On | Off }

パラメーター

On | Off

On: ロボット位置のラッチ機能を有効にします。Off: ロボット位置のラッチ機能を無効にします。

結果

パラメーターが省略されているときは、現在のラッチ機能の有効、無効を表示します。

解説

R-I/Oに接続したトリガー入力信号によるロボット位置のラッチ機能を有効、または無効にします。ラッチ機能を有効にしてから、最初のトリガー入力でロボットの位置がラッチされます。繰り返しロボット位置をラッチする場合は、一度LatchEnable Offでラッチ機能を解除し、再びLatchEnable Onを実行します。繰り返し使用する場合、各コマンドの処理時間を考慮すると、約60 ms以上の間隔が必要となります。LatchEnable自体の実行時間は無視できます。

注意

ラッチ機能を有効にする前に、SetLatchでトリガー入力ポートとトリガー信号論理の設定をしてください。

参照

[LatchPos関数](#), [LatchState関数](#), [SetLatch](#)

LatchEnable使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True    'トリガー待ち
  Print LatchPos      'ラッチ位置表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

3.15.2 LatchState関数

R-I/Oによるロボット位置のラッチ状態を返す関数です。

書式

LatchState

戻り値

ロボット位置のラッチが完了している場合は"True"、完了していない場合は"False"を返します。

ラッチ完了を確認したあと、LatchPos関数でラッチ位置情報を取得します。

SetLatchで連続ラッチ回数を指定した場合、指定した回数のラッチが全て完了すると"True"を返します。

参照

[LatchEnable](#), [LatchPos関数](#), [SetLatch](#), [Wait](#)

LatchState関数使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True    'トリガー待ち
  Print LatchPos      'ラッチ位置表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

3.15.3 LatchPos関数

R-I/O入力信号でラッチしたロボットの位置を返す関数です。

書式

LatchPos ([WithToolArm | WithoutToolArm], ラッチ番号)

パラメーター

WithToolArm | WithoutToolArm

関数呼び出し時に、Tool, Arm設定に基づいた位置を返すか、Tool 0, Arm 0の位置を返すかを指定します。パラメーターは省略可能ですが、ラッチ番号を指定する場合は省略しないでください。省略すると、WithToolArmが設定されます。

- WithToolArm: 0
- WithoutToolArm: 1

WithToolArm

値0の定数です。関数呼び出し時のTool, Arm設定に基づいた位置を返します。

WithoutToolArm

値1の定数です。Tool, Arm設定に関わらず、Tool 0, Arm 0の位置を返します。

ラッチ番号

LatchEnable On以降、何回目のR-I/O入力信号でラッチしたポイントデータを返すかを指定します。1, 2, 3, 4が指定可能です。SetLatchで連続ラッチ回数を指定することで、LatchEnable On以降、最大4回までのR-I/O入力信号によるポイントデータのラッチが可能です。パラメーターは省略可能です。省略すると1回目のR-I/O入力信号でラッチしたポイントデータを返します。

戻り値

R-I/O入力信号でラッチしたロボットの位置をポイントデータで返します。

LatchPos関数の実行には約15 msecの処理時間がかかります。

パラメーターがWithToolArmの場合、関数呼び出し時のTool, Arm設定に基づいた位置を返します。

パラメーターがWithoutToolArmの場合、Tool, Arm設定に関わらず、Tool 0, Arm 0の位置を返します。

参照

[LatchEnable](#), [LatchState関数](#), [SetLatch](#)

LatchPos関数使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE, 4
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True      'トリガー待ち
  Print LatchPos(WithoutToolArm, 1)      'ラッチ位置1表示
  Print LatchPos(WithoutToolArm, 2)      'ラッチ位置2表示
  Print LatchPos(WithoutToolArm, 3)      'ラッチ位置3表示
  Print LatchPos(WithoutToolArm, 4)      'ラッチ位置4表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

パラメーターを省略する場合の例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True      'トリガー待ち
  Print LatchPos      'ラッチ位置1表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

LatchPosの戻り値をポイントデータに代入する例

```
P2 = LatchPos      'ラッチ位置1を代入。パラメーター省略

P2 = LatchPos(WithoutToolArm, 1)      'ラッチ位置1を代入
P3 = LatchPos(WithoutToolArm, 2)      'ラッチ位置2を代入
P4 = LatchPos(WithoutToolArm, 3)      'ラッチ位置3を代入
P5 = LatchPos(WithoutToolArm, 4)      'ラッチ位置4を代入
```

3.15.4 LCase\$関数

文字列を小文字にして返します。

書式

LCase\$(文字列)

パラメーター

文字列

小文字にする文字列を指定します。

戻り値

小文字に変換した文字列を返します。

参照

[LTrim\\$関数](#), [Trim\\$関数](#), [RTrim\\$関数](#), [UCase\\$関数](#)

LCase\$関数使用例

```
str$ = "Data"  
str$ = LCase$(str$)   ' str$ = "data"
```


3.15.5 Left\$関数

文字列の左から、指定の文字列を取り出す関数です。

書式

Left\$(文字列, 文字数)

パラメーター

文字列

指定の文字列が左からコピーされる元の文字列です。

文字数

文字列の左からコピーする文字数を直接数値で指定します。

戻り値

指定した文字列の左から、指定の文字数をとって返します。

解説

Left\$は、指定した文字列の左から文字数の数だけ文字を取り出して返します。Left\$は、指定文字列内の文字数だけ文字を返すことができます。

参照

[Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Len関数](#), [Mid\\$関数](#), [Right\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Left\$関数使用例

下記は、文字列を解析するプログラムの例です。

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String, ByRef PartName$ As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(DataIn$, Len(DataIn$) - pos))

End
```

コマンドウィンドウでLeft\$命令を使う例です。

```
> Print Left$("ABCDEFGH", 2)
AB

> Print Left$("ABC", 3)
ABC
```

3.15.6 Len関数

文字列の文字数を返します。

書式

Len(文字列)

パラメーター

文字列

文字列式を指定します。

戻り値

Len命令の引数として与えられた文字列の文字数を、整数値で返します。

解説

Lenは、指定した文字列の文字数を整数値 (0～255)で返します。(文字列の字数制限範囲は0～255です。)

参照

[Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Left\\$関数](#), [Mid\\$関数](#), [Right\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Len関数使用例

下記は、文字列を解析するプログラムの例です。

```
Function ParsePartData(DataIn$ As String, ByRef PartNum$ As String, ByRef PartName$
As String, ByRef PartCount As Integer)

    Integer pos
    String temp$

    pos = Instr(DataIn$, ",")
    PartNum$ = Left$(DataIn$, pos - 1)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Left$(DataIn$, pos - 1)

    PartCount = Val(Right$(DataIn$, Len(DataIn$) - pos))

End
```

他にコマンドウィンドウでLen命令を使う例です。

```
> ? len("ABCDEFGH")
7

> ? len("ABC")
3

> ? len("")
0
>
```

3.15.7 LimitTorque

ハイパワーモード時のトルク上限値の設定と表示をします。

書式

(1) LimitTorque 全関節ハイパワートルク上限値

(2) LimitTorque 第1関節ハイパワートルク上限値, 第2関節ハイパワートルク上限値, 第3関節ハイパワートルク上限値, 第4関節ハイパワートルク上限値

(3) LimitTorque 第1関節ハイパワートルク上限値, 第2関節ハイパワートルク上限値, 第3関節ハイパワートルク上限値, 第4関節ハイパワートルク上限値, 第5関節ハイパワートルク上限値, 第6関節ハイパワートルク上限値

(4) LimitTorque

パラメーター

全関節ハイパワートルク上限値

すべての関節のハイパワートルク上限値を、各関節の瞬間最大トルクに対する割合を表す整数値で指定します。

第n関節ハイパワートルク上限値

第n関節のハイパワートルク上限値を、第n関節の瞬間最大トルクに対する割合を表す整数値で指定します。

結果

パラメーターが省略されているときは、現在のLimitTorque値を表示します。

解説

ハイパーモード時のトルク上限値を制限します。通常は最大トルクが設定されており、本設定値を変更する必要はありませんが、例えば、周辺機器との干渉によるロボットや装置のダメージを少なくするため、特定の動作に必要なトルク以上のトルクを発生しないように制限したい場合に有効です。上限値は、特定の動作をした時のピークトルクをPTRQで測定し、その値に変動分 (10%程度) を考慮した余裕を加えた値とします。

本コマンドでローパワーモード時のトルク上限値を下回るハイパワートルク上限値を設定することはできません。各機種、関節に対して設定できる最小値はそれぞれ異なります。設定後に設定値を表示し、実際に設定された上限値を確認してください。

下記の場合、LimitTorqueはデフォルト値に戻ります。

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

注意

- 低すぎるLimitTorque設定

LimitTorqueは、特定の動作に対して、設定された加減速度で動作するために必要なトルクの大きさにかかわらず、設定されたトルク制限値を上限値とし、トルクを制限します。その結果、特定の動作が設定した上限値より大きなトルクを必要とする場合、ロボットは適切に動作をすることができず、振動的な動作をしたり、異音が発生したり、位置偏差エラーおよびオーバーランが発生したりすることが考えられます。必ずPTRQを計測してから、トルク制限機能を使用してください。万一、上記の状態が発生した場合は、トルクが不足しているため、トルク上限値を大きくし、正常に動作するように調整してください。

参照

[LimitTorque関数](#), [Power](#), [PTRQ](#), [RealTorque関数](#)

LimitTorque使用例

第1関節の最大トルクを80%に制限して動作する例です。

```
Function main
  Motor On
  Power high
  Speed 100; Accel 100,100
  LimitTorque 80,100,100,100  '第1関節の最大トルクを80%に制限する
  Jump P1                    'Jump動作を実行する
Fend
```

3.15.8 LimitTorque関数

LimitTorque命令の設定値を返します。

書式

LimitTorque(関節番号)

パラメーター

関節番号

1～9の整数で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

LimitTorque命令の設定値を整数値で返します。

参照

[LimitTorque](#)

LimitTorque関数使用例

```
Print LimitTorque(1) '第1関節のLimitTorque値を表示
```

3.15.9 LimitTorqueLP

ローパワーモード時のトルク上限値の設定と表示をします。

書式

(1) LimitTorqueLP 全関節ローパワートルク上限値

(2) LimitTorqueLP 第1関節ローパワートルク上限値, 第2関節ローパワートルク上限値, 第3関節ローパワートルク上限値, 第4関節ローパワートルク上限値

(3) LimitTorqueLP 第1関節ローパワートルク上限値, 第2関節ローパワートルク上限値, 第3関節ローパワートルク上限値, 第4関節ローパワートルク上限値, 第5関節ローパワートルク上限値, 第6関節ローパワートルク上限値

(4) LimitTorqueLP

パラメーター

全関節ローパワートルク上限値

すべての関節のローパワートルク上限値を、各関節の瞬間最大トルクに対する割合を表す整数値で指定します。

第n関節ローパワートルク上限値

第n関節のローパワートルク上限値を、第n関節の瞬間最大トルクに対する割合を表す整数値で指定します。

結果

パラメーターが省略されているときは、現在のLimitTorqueLP 値を表示します。

本コマンドにより値が変更されていない場合は、デフォルト値が表示されます。

解説

ローパワーモード時のトルク上限値を制限します。デフォルトでは、ローパワー動作に対して必要なトルクが上限値として設定されており (値は機種や軸によって異なる。15～60%程度)、本設定値を変更する必要はありませんが、例えば、周辺機器との衝突によるロボットや装置のダメージを少なくするため、衝突のない正常動作に必要なトルクを大きく超えるトルクが発生しないように制限したい場合に有効です。上限値は、正常動作をした場合のピークトルクをPTRQで測定し、その値に変動分 (4割を推奨) を考慮したマージンを加えた値とします。同じ値を異なるロボットに適用する場合は、さらに1～2割のマージンを加えてください。

PTRQの値は、ローパワー時のデフォルトの最大トルクを1.0として表示します。例えば、変更前のデフォルト値が27%で、PTRQでの測定値が0.43となった場合、 $27\% \times 0.43 \times 1.4 = 16.25$ となるので、整数で繰り上げて17を設定します。

本コマンドに、5%を下回る値や、デフォルト値を上回る値を設定することはできません。その場合、5以下の設定値は5に、デフォルト値を超えた設定値はデフォルト値に丸められます。例えば、"LimitTorqueLP 100"とすると、デフォルト値は必ず100以下なので、全軸デフォルト値の設定に戻ります。設定後に設定値を表示し、実際に設定された上限値を確認してください。

コントローラーを再起動するまで、LimitTorqueLPの設定値は有効となります。

注意

■ 低すぎるLimitTorqueLP設定

LimitTorqueLPは、特定の動作に対して、設定された加減速度で動作するために必要なトルクの大きさにかかわらず、設定されたトルク制限値を上限値とし、トルクを制限します。その結果、特定の動作が設定した上限値より大きなトルクを必要とする場合、ロボットは適切に動作をすることができず、位置偏差エラーが発生することが考えられます。必ずローパワー状態でのPTRQを計測してから、トルク制限機能を使用してください。

上記の状態が発生した場合は、トルクが不足しているため、トルク上限値を大きくし、正常に動作するように調整してください。

参照

[LimitTorqueLP関数, PTRQ](#)

LimitTorqueLP使用例

第1関節の最大トルクを10%に制限して動作する例です。

```
Function main
Motor On
Power low
LimitTorqueLP 10,27,31,42      '第1関節の最大トルクを10%に制限する
                                'その他の軸はデフォルトの値を設定
Go P1      'Go動作を実行する
Fend
```

3.15.10 LimitTorqueLP関数

LimitTorqueLP命令の設定値を返します。

書式

LimitTorqueLP(関節番号)

パラメーター

関節番号

1～9の整数で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

LimitTorqueLP命令の設定値を整数値で返します。

参照

[LimitTorqueLP](#)

LimitTorqueLP関数使用例

```
Print LimitTorqueLP(1) '第1関節のLimitTorqueLP値を表示
```


3.15.11 LimitTorqueStop

ハイパワーモード時にトルク上限に達した場合にロボットを停止する機能をオン、またはオフします。

書式

(1) LimitTorqueStop 状態

(2) LimitTorqueStop 状態, 関節番号

(3) LimitTorqueStop

パラメーター

状態

- On: トルク上限での停止機能を有効にします。
- Off: トルク上限での停止機能を無効にします。

関節番号

1から6までの関節番号を指定します。(スカラロボットの場合、関節番号は1から4まで)

結果

パラメーターが省略されているときは、現在のLimitTorqueStop状態を表示します。

解説

ハイパワー動作でのトルク上限での停止機能を有効にします。トルク上限 (デフォルトは100%) で、ロボットを即時停止させます。LimitTorqueによるトルク制限機能と併用することにより、ハイパワーモード時の衝突や接触による、ロボットや装置のダメージを低減する効果が得られます。

全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オフです。

コントローラー起動時にデフォルト値に戻りますが、それ以外の場合、本コマンドで明示的に設定しない限り、状態は変化しません。

トルク上限に達した場合、エラー 5040 "ハイパワー状態でモータートルク出力が異常" のメッセージを出力し、ロボットを停止します。

参照

[LimitTorque](#), [LimitTorque関数](#)

LimitTorqueStop使用例

第1関節の最大トルクを30%に制限し、即時停止させる例です。

```
Function main
Motor On
Power high
Speed 20
Accel 20,20
LimitTorque 30,100,100,100      '第1関節の最大トルクを30%に制限する
LimitTorqueStop On, 1          '第1関節は最大トルクで即停止する
Go P1                          'Go動作を実行する
Fend
```

3.15.12 LimitTorqueStop関数

LimitTorqueStop命令の設定値を返します。

書式

LimitTorqueStop(関節番号)

パラメーター

関節番号

1～6の整数で指定します。

戻り値

LimitTorqueStop命令の設定値を整数値で返します。

- 0 = オフ
- 1 = オン

参照

[LimitTorqueStop](#)

LimitTorqueStop関数使用例

```
Print LimitTorqueStop(1) '第1関節のLimitTorqueStop値を表示
```

3.15.13 LimitTorqueStopLP

ローパワーモード時にトルク上限に達した場合にロボットを停止する機能をオン、またはオフします。

書式

(1) LimitTorqueStopLP 状態

(2) LimitTorqueStopLP 状態, 関節番号

(3) LimitTorqueStopLP

パラメーター

状態

- On: トルク上限での停止機能を有効にします。
- Off: トルク上限での停止機能を無効にします。

関節番号

1から6までの関節番号を指定します。(スカラロボットの場合、関節番号は1から4まで)

結果

パラメーターが省略されているときは、現在のLimitTorqueStopLP状態を表示します。

解説

ローパワー動作でのトルク上限での停止機能を有効にします。トルク上限で、ロボットを即時停止させます。LimitTorqueLPによるトルク制限機能と併用することにより、ローパワーモード時の衝突や接触による、ロボットや装置のダメージを低減する効果が得られます。

全軸のオンオフと、各軸のオンオフの設定ができます。デフォルト値は、全軸オフです。

コントローラー起動時にデフォルト値に戻りますが、それ以外の場合、本コマンドで明示的に設定しない限り、状態は変化しません。

トルク上限に達した場合、エラー 5041 "ローパワー状態でモータートルク出力が異常"のメッセージを出力し、ロボットを停止します。

参照

[LimitTorqueLP](#), [LimitTorqueLP関数](#)

LimitTorqueStopLP使用例

第3関節の最大トルクを15%に制限し、即時停止させる例です。

```
Function main
Motor On
Power low
LimitTorqueLP 20,27,15,42      '第3関節の最大トルクを15%に制限する 'その他の軸はデフォルトの値
                                'を設定
LimitTorqueStopLP On, 3        '第3関節は最大トルクで即停止する
Go P1                          'Go動作を実行する
Fend
```

3.15.14 LimitTorqueStopLP関数

LimitTorqueStopLP命令の設定値を返します。

書式

LimitTorqueStopLP(関節番号)

パラメーター

関節番号

1～6の整数で指定します。

戻り値

LimitTorqueStopLP命令の設定値を整数値で返します。

- 0 = オフ
- 1 = オン

参照

[LimitTorqueStopLP](#)

LimitTorqueStopLP関数使用例

```
Print LimitTorqueStopLP(3) '第3関節のLimitTorqueStopLP値を表示
```

3.15.15 LimZ

Jumpコマンドのときの第3関節の高さ (Z座標値)の初期値を設定します。

書式

(1) LimZ Z座標値

(2) LimZ

パラメーター

Z座標値

第3関節の動作範囲内にある座標の値を指定します。

結果

パラメーターが省略されているときは、現在のLimZ値を表示します。

解説

Jump命令実行時には、ロボットアームが第3関節 (Z軸)方向で上昇し、次にX-Y 平面で移動し、最後に第3関節 (Z軸)方向に下降しますが、LimZは、そのときアームが第3関節 (Z軸)方向に動作する高さの上限を設定します。LimZは、Jump命令時の第3関節の動作範囲の最高座標のデフォルト値を設定します。Jump命令時に特定のLimZ値が設定されていなければ、最後に設定されたLimZ値が使われます。

注意

- LimZ値を0にリセットする

コントローラーの再起動、SFree, SLock, Motor Onなどの命令で、いずれもLimZ値は、0に初期化されます。

- LimZ値はArm、Tool、またはLocal 座標には使えません

LimZの高さ制限値は、ロボット座標のZ座標値です。Arm、Tool、またはLocal 座標のZ座標値ではありません。したがって、高さの異なるハンドまたは手先ツールを使うときには、注意してください。

- LimZは、Jump3およびJump3CPには影響しません。

スパン動作は、必ずしも座標系のZ軸に垂直とは限らないため、LimZ は、Jump3 およびJump3CPには影響しません。

参照

[Jump](#)

LimZ使用例

Jump操作時にLimZを使う例です。

```
Function main
  LimZ -10          'LimZのデフォルト値を設定
  Jump P1           'Jump実行時-10で水平移動
  Jump P2 LimZ -20  'Jump実行時-20で水平移動
  Jump P3           'Jump実行時-10で水平移動
Fend
```

3.15.16 LimZ関数

LimZ命令の設定値を返します。

書式

LimZ

戻り値

LimZ命令の設定値を実数値で返します。

参照

[LimZ](#)

LimZ関数使用例

```
Real savLimz  
  
savLimz = LimZ  
LimZ -25  
Go pick  
LimZ savLimz
```

3.15.17 LimZMargin

LimZ設定値より高い位置で動作開始したときのエラー検出のマージンを設定、および設定値を返す。

書式

(1) LimZMargin LimZマージン

(2) LimZMargin

パラメーター

LimZマージン

LimZエラー検出のマージン値を指定します。

結果

パラメーターが省略されているときは、現在のLimZMargin値を表示します。

解説

Jump命令実行時には、LimZで設定した高さに第3関節(Z軸)が上昇しますが、Jump動作開始時にLimZ位置より高い位置に第3関節 (Z軸)ある場合、エラーが発生します。LimZMarginはこのエラー検出に対してマージン値を設定します。デフォルトでは0.02 mmが設定されています。

注意

- LimZMargin値をデフォルト値にリセットする

コントローラーの再起動、SFree, SLock, Motor Onなどの命令で、いずれもLimZMargin値は、デフォルト値に初期化されます。

参照

[LimZMargin関数](#), [LimZ](#)

LimZMargin使用例

Jump操作時にLimZMarginを使う例です。

```
Function main
    LimZ -10                'LimZのデフォルト値を設定
    LimZMargin 0.03        'LimZのエラー検出マージンに0.03 mmを設定
    Jump P1                'Jump実行時 (10で水平移動)
    Jump P2 LimZ -20        'Jump実行時 (20で水平移動)
    Jump P3                'Jump実行時 (10で水平移動)
Fend
```

3.15.18 LimZMargin関数

LimZMargin命令の設定値を返します。

書式

LimZMargin

戻り値

LimZMargin命令の設定値を実数値で返します。

参照

[LimZMargin](#), [LimZ](#)

LimZMargin関数使用例

```
Real savLimzMargin  
  
savLimzZMargin = LimZMargin  
LimZMargin 0.03  
Jump pick  
LimZ savLimZMargin
```


3.15.19 Line Input

入力データを1行読んで、そのデータを文字列変数に代入します。

書式

Line Input 文字列変数名\$

パラメーター

文字列変数名\$

文字列変数名を指定します。(文字列変数は、最後に\$をつけてください。)

解説

Line Inputはディスプレイデバイスからの1行入力データを読んで、Line Input命令の文字列変数に代入します。Line Input命令がユーザーからデータを受け取れる状態になると、ディスプレイデバイスにプロンプト"?"が表示されます。このプロンプトの後に入力されたデータ行が、文字列の値として代入されます。入力データを入力した後は、[ENTER]キーを押してください。

参照

[Input](#), [Input #](#), [Line Input #](#), [ParseStr / ParseStr関数](#)

Line Input使用例

Line Inputの使用例です。

```
Function Main
  String A$
  Line Input A$ ' 入力データを1行読んでA$に代入
  Print A$
Fend
```

上記プログラムを実行すると、下記のセッションが行われます。

```
?A, B, C
A, B, C
```

3.15.20 Line Input

ファイル, 通信ポート, データベース, デバイスから、1行データを読み込みます。

書式

Line Input #ポート番号, 文字列変数名\$

パラメーター

ポート番号

ファイル, 通信ポート, データベース, デバイスを示すID番号です。ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。データベース番号は、OpenDBステートメントで指定した番号です。

デバイスIDは以下の数値です。

- 21 RC+
- 20 TP4

文字列変数名\$

文字列変数名を指定します。(文字列変数は、最後に\$をつけてください。)

解説

Line Input # は、ポート番号で指定したデバイスから1行データを読み、文字列変数名\$で指定された変数に代入します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

参照

[Input](#), [Input #](#), [Line Input](#)

Line Input #使用例

下記例では、通信ポート1から文字列データを受け取り、文字列変数A\$に代入します。

```
Function lintest
  String a$
  Print #1, "Please input string to be sent to robot"
  Line Input #1, a$
  Print "Value entered = ", a$
Fend
```

3.15.21 LJM関数

指定ポイントに対して参照ポイントから見て最も関節移動量が少なくなるように姿勢フラグを変換したポイントデータを返します。

書式

LJM (ポイント指定 [, 参照ポイント指定 [, 姿勢フラグ選択]])

パラメーター

ポイント指定

対象とするポイントデータを指定します。

参照ポイント指定

基準にするポイントデータを指定します。参照ポイント指定を省略した場合は現在位置 (Here)を参照ポイントとします。

姿勢フラグ選択

垂直6軸型

- 1: J4軸が最短移動量となるように、手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flag、J1Flagによって変換します。姿勢フラグ選択を省略したときのデフォルトです。
- 2: J4FlagおよびJ6Flagによって変換します。
- 3: J5軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flag、J1Flagによって変換します。
- 4: J6軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flag、J1Flagによって変換します。

"姿勢フラグ選択"	腕姿勢	肘姿勢	手首姿勢	J1Flag	J4Flag	J6Flag	移動量が最短となる軸の優先順
1	-	-	○	○	○	○	J4
2	-	-	-	○	○	○	-
3	-	-	○	○	○	○	J5
4	-	-	○	○	○	○	J6

Note: "-" は、"参照ポイント指定"で指定した姿勢と同じ姿勢です。

RSシリーズ

- 1: 腕姿勢 (Handフラグ)、J1FlagおよびJ2Flagによって変換します。"姿勢フラグ選択"を省略したときのデフォルトです。
- 2: 腕姿勢 (Handフラグ)、J1FlagおよびJ2Flagによって変換します。"姿勢フラグ選択"が変換されるときにU軸が動作範囲外となるエラーを防ぎます。

N2シリーズ

- 1: J1, J5軸の優先順で、関節移動量が少なくなる姿勢に変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ)、肘姿勢 (Elbowフラグ)、手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagです。肘姿勢 (Elbowフラグ)は、必ず上肘姿勢になります。"姿勢フラグ選択"を省略したときのデフォルトです。
- 2: J1, J4軸の優先順で、関節移動量が少なくなる姿勢に変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ)、肘姿勢 (Elbowフラグ)、手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagです。肘姿勢 (Elbowフラグ)は、必ず上肘姿勢になります。

- 3: J4軸が最短移動量となるように、手首姿勢 (Wristフラグ), J4FlagおよびJ6Flagによって変換します。
- 4: J4FlagおよびJ6Flagによって変換します。
- 5: "参照ポイント指定"で指定した姿勢と異なる腕姿勢 (Handフラグ)に変更し、J5軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ), 肘姿勢 (Elbowフラグ), 手首姿勢 (Wristフラグ), J4FlagおよびJ6Flagです。また、肘姿勢 (Elbowフラグ)は、必ず上肘姿勢になります。
- 6: "参照ポイント指定"で指定した姿勢と異なる腕姿勢 (Handフラグ)に変更し、J4軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ), 肘姿勢 (Elbowフラグ), 手首姿勢 (Wristフラグ), J4FlagおよびJ6Flagです。また、肘姿勢 (Elbowフラグ)は、必ず上肘姿勢になります。
- 7: 肘姿勢 (Elbowフラグ)を下肘姿勢にし、J1, J5軸の優先順で最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ), 肘姿勢 (Elbowフラグ), 手首姿勢 (Wristフラグ), J4FlagおよびJ6Flagです。
- 8: 肘姿勢 (Elbowフラグ)を下肘姿勢にし、J1, J4軸の優先順で最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。変換対象となる姿勢は、腕姿勢 (Handフラグ), 肘姿勢 (Elbowフラグ), 手首姿勢 (Wristフラグ), J4FlagおよびJ6Flagです。

"姿勢フラグ選択"	腕姿勢	肘姿勢	手首姿勢	J4Flag	J6Flag	移動量が最短となる軸の優先順
1	○	*1	○	○	○	J1>J5
2	○	*1	○	○	○	J1>J4
3	-	-	○	○	○	J4
4	-	-	-	○	○	-
5	*2	*1	○	○	○	J5
6	*2	*1	○	○	○	J4
7	○	*3	○	○	○	J1>J5
8	○	*3	○	○	○	J1>J4

Note: "-" は、"参照ポイント指定"で指定した姿勢と同じ姿勢です。

- *1: 上肘姿勢
- *2: "参照ポイント指定"で指定した姿勢と腕姿勢が異なります。
- *3: 下肘姿勢

N6シリーズ

- 1: J4軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。"姿勢フラグ選択"を省略したときのデフォルトです。
- 2: J4FlagおよびJ6Flagによって変換します。
- 3: J5軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。
- 4: J6軸が最短移動量となるように手首姿勢 (Wristフラグ)、J4FlagおよびJ6Flagによって変換します。

"姿勢フラグ選択"	腕姿勢	肘姿勢	手首姿勢	J1Flag	J4Flag	J6Flag	移動量が最短となる軸の優先順
1	-	-	○	○	○	○	J4
2	-	-	-	○	○	○	-
3	-	-	○	○	○	○	J5
4	-	-	○	○	○	○	J6

Note: "-" は、"ポイント指定"で指定した姿勢と同じ姿勢です。

解説

垂直6軸型ロボットやNシリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは上記ポイント演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。

LJM関数はこのような手首部の意図しない回転を防止するためにポイントデータの姿勢フラグを適切に変換します。

また、Nシリーズでは、腕姿勢フラグや肘姿勢フラグを変更することで、タクトタイムの短縮や垂直6軸ロボットで必要な回避ポイントのティーチングを省くことも可能です。

RSシリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、第1アームが意図しない方向へ回転することがあります。LJM関数はこのような第1アームの意図しない回転を防止するために、ポイントデータの姿勢フラグを適切に変換します。

また、RSシリーズでは、姿勢フラグが変換されたときにU軸が動作範囲外に動作しようとしてエラーが発生することがあります。LJM関数はこのようなU軸の動作範囲外エラーを防止するために、U軸の目標角度を動作範囲内の目標角度に補正します。これは、姿勢フラグ選択で2を指定することによって利用可能となります。

垂直6軸型, Nシリーズ, RSシリーズ以外のロボットでは、指定ポイントをそのまま返します。

注意

■ 参照ポイントの省略と並列処理

参照ポイントの省略と並列処理を1つの動作命令内で混在させることはできません。

```
Go LJM(P10)    ! D10; MemOn 1 !
```

上記のような使い方はできません。

```
P999 = Here
Go LJM(P10,P999) ! D10; MemOn 1 !
```

上記のようなプログラムに変更してください。

■ N2シリーズの姿勢フラグ選択について

- 姿勢フラグ選択1, 2:

ロボットのタクトタイムを短縮させたい場合、姿勢フラグ1、または2を選択してください。

第1関節の移動量が最少となる姿勢をとるため、ほとんどの動作でタクトタイムが最短となる動作を行うことができます。第5関節の移動量を少なくする動作を行う場合は姿勢選択フラグ1を、第4関節の移動量を少なくする動作を行う場合は姿勢選択フラグ2を選択してください。

- 姿勢フラグ選択3, 4:

垂直6軸型と同じように使用したい場合に選択してください。

姿勢フラグ3は垂直6軸型の姿勢フラグ1と同じです。

姿勢フラグ4は垂直6軸型の姿勢フラグ2と同じです。

- 姿勢フラグ選択5, 6:

ロボット動作中にハンドがロボット周辺の壁などに接触する場合、姿勢フラグ5、または6を選択してください。ハンドがロボットの原点近傍を通過するため、ロボットが周囲の障害物に当たりにくい動作を行うことができます。第5関節の移動量を少なくする動作を行う場合は姿勢選択フラグ5を、第4関節の移動量を少なくする動作を行う場合は姿勢選択フラグ6を選択してください。

- 姿勢フラグ選択7, 8:

下肘姿勢を取りたい場合、姿勢フラグ7または8を選択してください。動作によっては、ロボットは姿勢選択フラグ5, 6のように原点近傍を通過する動作を行うため、ロボットの周囲に障害物がある場合に、障害物に当たりにくい動作を行うことができます。第5関節の移動量を少なくする動作を行う場合は姿勢選択フラグ7を、第4関節の移動量を少なくする動作を行う場合は姿勢選択フラグ8を選択してください。

■ ローカル番号

LJM関数で返されるポイントのローカル番号は、"ポイント指定"と同じローカル番号になります。

参照

[Pallet](#)

LJM関数使用例

```
Function main
  Integer i, j

  P0 = XY(300, 300, 300, 90, 0, 180)
  P1 = XY(200, 280, 150, 90, 0, 180)
  P2 = XY(200, 330, 150, 90, 0, 180)
  P3 = XY(-200, 280, 150, 90, 0, 180)

  Pallet 1, P1, P2, P3, 10, 10

  Motor On
  Power High
  Speed 50; Accel 50, 50
  SpeedS 1000; AccelS 5000

  Go P0
  P11 = P0 -TLZ(50)

  For i = 1 To 10
    For j = 1 To 10
      'ポイントの指定
      P10 = P11      '退避ポイント
      P12 = Pallet(1, i, j)      '目標ポイント
      P11 = P12 -TLZ(50)      '接近開始ポイント
```

```
'各ポイントのLJM変換
P10 = LJM(P10)
P11 = LJM(P11, P10)
P12 = LJM(P12, P11)
'動作実行
    Jump3 P10, P11, P12 C0
Next
Next
Fend

Function main2
    P0 = XY(300, 300, 300, 90, 0, 180)
    P1 = XY(400, 0, 150, 90, 0, 180)
    P2 = XY(400, 500, 150, 90, 0, 180)
    P3 = XY(-400, 0, 150, 90, 0, 180)
    Pallet 1, P1, P2, P3, 10, 10

    Motor On
    Power High
    Speed 50; Accel 50, 50
    SpeedS 1000; AccelS 5000

    Go P0

    Do
        'ポイントの指定
        P10 = Here -TLZ(50)          '退避ポイント
        P12 = Pallet(1, Int(Rnd(9)) + 1, Int(Rnd(9)) + 1)    '目標ポイント
        P11 = P12 -TLZ(50)          '接近開始ポイント

        If TargetOK(P11) And TargetOK(P12) Then    'ポイントのチェック
            '各ポイントのLJM変換
            P10 = LJM(P10)
            P11 = LJM(P11, P10)
            P12 = LJM(P12, P11)
            '動作実行
            Jump3 P10, P11, P12 C0
        EndIf
    Loop
Fend
```

3.15.22 LoadPoints

ポイントファイルを、ロボットのポイントメモリーエリアにロードします。

書式

LoadPoints ファイル名 [, Merge]

パラメーター

ファイル名

ロボットのポイントメモリーエリアにロードするファイルを指定する文字列。拡張子は".pts"固定です。拡張子を省略した場合、".pts"が追加されます。指定するファイルはプロジェクト内のファイルに限ります。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

Merge

新しいポイントをロードする前に現在のポイントをクリアしたくないときに設定しておきます。これを設定すると、設定されているポイントに新しいポイントが追加されます。追加するポイントがすでにファイルの中にある場合は、上書きします。省略可能です。

解説

LoadPointsは、コントローラーのメインメモリーにポイントファイルをロードします。

ポイントファイルを統合するときは、Mergeを設定します。例えば、共通ポイントを0～100の範囲で含む1つのメインポイントファイルがあるとしします。それらのポイントをクリアせずに現在動作中の各パーツに対する新たなポイントファイルをロードするときは、Mergeを設定します。その場合、範囲は101～999になります。

起こりやすいエラー

- パスは指定できません

ファイル名がパスを含む場合、エラーが発生します。

- 指定ファイルが見つからない場合（ファイルが存在しない）

ファイルが見つからない場合はエラーとなります。

- 別のロボットのポイントファイル

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。この場合、プロジェクトエディターでポイントファイルを追加するか、SavePointsまたはImportPointsを実行します。

参照

[ImportPoints](#), [Robot](#), [SavePoints](#)

LoadPoints使用例

```
Function main
```

```
  '現在のロボットに共通のポイントをロード  
  LoadPoints "R1Common.pts"
```

```
  'パーツモデル1のポイントを統合  
  LoadPoints "R1Model1.pts", Merge
```

```
  ロボット 2
```



```
' ロボット2のポイントファイルをロード  
LoadPoints "R2Model1.pts"
```

```
End
```

3.15.23 Local

ローカル座標系の定義と表示をします。

書式

(1) Local ローカル座標系番号, (ポイント番号1: ポイント番号2), (ポイント番号3: ポイント番号4) [, { L | R }] [, BaseU]

(2) Local ローカル座標系番号, 座標系データ

(3) Local ローカル座標系番号, 原点, [X軸指定], [Y軸指定], [{ X | Y }]

(4) Local ローカル座標系番号

パラメーター

ローカル座標系番号

ローカル座標系の番号を指定します。ローカル座標系は1～15の整数で、合計15まで定義することができます。

ポイント番号1, ポイント番号3

ローカル座標系のポイントデータをポイント変数で指定します。

ポイント番号2, ポイント番号4

ベース座標系のポイントデータをポイント変数で指定します。

L | R

ローカル原点を、左 (1番) または右 (2番) にあわせます。省略可能です。

BaseU

指定するとU軸座標はベース座標系を使います。省略可能で、省略すると、U軸座標はローカル座標系を使います。

座標系データ

ローカル座標系の原点と向きをポイントデータで直接指定します。スカラロボット (RSシリーズ含む) の場合、V座標とW座標を"0"を指定してください。

原点

ローカル座標系原点を定義するロボット座標系上の位置を、P#(整数) またはP(式) で指定します。

X軸指定

ローカル座標系のX軸上の点を定義するロボット座標系上の位置を、P#(整数) またはP(式) で指定します。省略可能です。

Y軸指定

ローカル座標系のY軸上の点を定義するロボット座標系上の位置を、P#(整数) またはP(式) で指定します。省略可能です。

X

原点とX軸指定を結んだ直線をローカル座標系のX軸と定義します。X軸と3つのポイントで作られた平面からローカル座標系が算出されるため、Y軸指定は、必ずしもY軸上のポイントになりません。省略可能です。(デフォルト)

Y

原点とY軸指定を結んだ直線をローカル座標系のY軸と定義します。Y軸と3つのポイントで作られた平面からローカル座標系が算出されるため、X軸指定は、必ずしもX軸上のポイントになりません。省略可能です。

解説

(1) Local は、ベース座標系の2点の位置データ、ポイント番号2とポイント番号4に合致するローカル座標系の2点、ポイント番号1およびポイント番号3を指定して、ローカル座標系を定義します。

例:

```
Local 1, (P1:P11), (P2:P12)
```

P1 と P2 はローカル座標系の点です。P11 と P12 はベース座標系の点です。

もし、ローカル座標系で指定した2点間の距離が、ベース座標系の2点間の距離と異なる場合は、ローカル座標系の2点の midpoint とベース座標系の2点の midpoint が一致する位置にローカル座標系を定義します。

同様に、2つの座標系の midpoint を使って、ローカル座標系のZ軸を定義します。

(2) 原点とベース座標系に対する角度でローカル座標系を定義します。

例:

```
Local 1, XY(x, y, z, u)
Local 1, XY(x, y, z, u, v, w)
Local 1, P1
```

(3) 原点, X軸上ポイント, Y軸上ポイントを指定して、3次元ローカル座標系を定義します。各ポイントのX, Y, Z座標のみを使用し、U, V, W座標は無視します。パラメーターのXを指定すると、X軸指定はローカル座標系のX軸上になり、Y軸指定のZ座標のみ使用します。パラメーターのYを指定すると、Y軸指定はローカル座標系のY軸上になり、X軸指定のZ座標のみ使用します。

例:

```
Local 1, P1, P2, P3
Local 1, P1, P2, P3, X
Local 1, P1, P2, P3, Y
```

(4) 指定されたローカルの設定を表示します。

- L/R パラメーターの使用: ローカル座標系は、前述したように基本的に midpoint によって定義しますが、オプションのLまたはRを使って座標系の左右を指定することができます。
 - 左ローカル: 左ローカルは、ローカル座標系の点、ポイント番号1が、ベース座標系の点、ポイント番号2に一致する位置にローカル座標を定義します。(Z軸方向も含まれます。)
 - 右ローカル: 右ローカルは、ローカル座標系の点ポイント番号3が、ベース座標系の点ポイント番号4に一致する位置に、ローカル座標を定義します。(Z軸方向も含まれます。)
- BaseU パラメーターの使用: BaseUパラメーターを省略すると、設定された4点のXY座標値に適合するようローカル座標系のU軸が自動的に補正されます。したがって、ベース座標系の2点は、最初からU軸座標値を含んでいることもあります。

ローカル座標系のU軸は、自動的に補正するよりも、ティーチングによる回転軸の補正など、ベース座標系の2点のU軸値に基づいて補正を行うことをお勧めします。そのときには、BaseUパラメーターの設定が必要です。

注意

- 水平多関節ロボットの場合、V,Wに対しての設定はしないでください。

水平多関節ロボットを使用する場合、ベース座標系にてV,W座標に数値を設定しないか、"0"を入力してください。設定した場合は、J4での範囲外にエラーなどが発生する場合があります。

参照

[ArmSet](#), [Base](#), [ECPSet](#), [LocalClr](#), [TLSet](#), [Where](#)

Local使用例

コマンドウィンドウからの操作例です。

左ローカルでローカル座標系の原点を定義する例:

```
> p1 = 0, 0, 0, 0/1  
> p2 = 100, 0, 0, 0/1  
> p11 = 150, 150, 0, 0  
> p12 = 300, 150, 0, 0  
> local 1, (P1:P11), (P2:P12), L  
  
> p21 = 50, 0, 0, 0/1  
> go p21
```

原点をローカル座標系の原点と定義する例:

```
> local 1, 100, 200, -20, 0
```

X軸を45度回転させた原点をローカル座標系の原点と定義する例:

```
> local 2, 50, 200, 0, 0, 45, 0
```

P2をローカル座標系のX軸に合わせた位置を3次元ローカル座系の原点と定義する例:

```
> local 3, p1, p2, p3, x
```

P3をローカル座標系のY軸に合わせた位置を3次元ローカル座標系の原点と定義する例:

```
> local 4, p1, p2, p3, y
```

3.15.24 Local関数

設定されたローカル座標系データを返す関数です。

書式

Local(ローカル座標系番号)

パラメーター

ローカル座標系番号

ローカル座標系番号 (1～15の整数)を、式または数値で指定します。

戻り値

設定されたローカル座標系データをポイントデータとして返します。

参照

[Local](#)

Local関数使用例

```
P1 = Local(1)
```

3.15.25 LocalClr

ローカル座標系をクリア (未定義) にします。

書式

LocalClr ローカル座標系番号

パラメーター

ローカル座標系番号

設定をクリア (未定義) にするローカル座標系番号 (1～15の整数) を、式または数値で指定します。

参照

[Arm](#), [ArmSet](#), [ECPSet](#), [Local](#), [Tool](#), [TLClr](#), [TLSet](#)

LocalClr使用例

```
LocalClr 1
```

3.15.26 LocalDef関数

ローカルの設定状態を返します。

書式

LocalDef(ローカル座標系番号)

パラメーター

ローカル座標系番号

状態を返すローカルの座標系番号 (1～15の整数)を指定します。

戻り値

指定したローカルが設定されている場合"True"、設定されていない場合"False"を返します。

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLClr](#), [TLSet](#)

LocalDef関数使用例

```
Function DisplayLocalDef(localNum As Integer)

    If LocalDef(localNum) = False Then
        Print "Local ", localNum, "is not defined"
    Else
        Print "Local 1: ",
        Print Local(localNum)
    EndIf
End
```

3.15.27 Lof関数

指定したRS-232Cポート、またはTCP/IPポートのバッファの受信データの行数を返します。

書式

Lof(通信ポート番号)

パラメーター

通信ポート番号

OpenCom (RS-232C)、またはOpenNet (TCP/IP)ステートメントで指定した番号を指定します。

戻り値

受信バッファのデータ行数を返します。受信データが存在しない場合は"0"を返します。

解説

Lof関数は、指定した通信ポートの受信データの有無を確認します。受信データはInput#命令により引き出されます。Wait命令によりLof関数の戻り値を待機することができます。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

PC COMポート (1001～1008)を使用している場合、Wait命令でLof関数を使用することはできません。

参照

[ChkCom関数](#), [ChkNet関数](#), [Input #](#), [Wait](#)

Lof関数使用例

次の例はコマンドウィンドウで実行します。通信ポート1の受信データの行数を表示します。

```
>print lof(1)
5
>
```


3.15.28 LogIn関数

他のユーザーとして、Epson RC+にログインします。

書式

LogIn (ログID, パスワード)

パラメーター

ログID

ユーザーログインIDの文字列式

パスワード

ユーザーパスワードの文字列式

解説

アプリケーションでEpson RC+のセキュリティーを操作することができます。例えば、他のユーザーがシステムにログインするためのメニューを表示することができます。ユーザーは独自のセキュリティーの権限を所有しています。セキュリティーについての詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド"

開発環境でプログラムを実行した場合、プログラムが停止したあと、プログラムがスタートする前のユーザーに戻ります。

Autoモードでオペレーターウィンドウを実行しているときは、Auto LogInが有効になっている限り、アプリケーションがゲストユーザーとしてログインします。この場合、Epson RC+ システムで設定されていれば、アプリケーションがカレントウィンドウズユーザーとしてログインします。

注意

このコマンドは、セキュリティーオプションが有効になっているときのみ使用できます。

LogIn関数使用例

```
Integer errCode  
errCode = LogIn("operator", "oprpass")
```

3.15.29 Long

Long型の変数を宣言します。(4バイト整数型変数)

書式

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

パラメーター

変数名

Long型に宣言する変数名を指定します。配列変数の最大要素番号配列変数の最大要素番号で、最高3次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Longは整数型変数の宣言をします。Long型変数の範囲は-2147483648~2147483647です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Long使用例

次の例はLongを使ってLong型の変数を宣言するプログラムです。

```
Function longtest

    Long A(10)           'Long型の1次元配列
    Long B(10, 10)       'Long型の2次元配列
    Long C(5, 5, 5)      'Long型の3次元配列
    Long var1, arrayVar(10)
    Long i
    Print "Please enter a Long Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter a Long Number"
        Input arrayVar(i)
        Print "Value Entered was ", arrayVar(i)
    Next i
Fend
```

3.15.30 LSet\$関数

指定された文字列の最後に半角スペースを加えて、指定の長さにした文字列を返します。

書式

LSet\$(文字列,文字列の長さ)

パラメーター

文字列

文字列式を指定します。

文字列の長さ

返される文字列の長さを示す、整数または式を指定します。

戻り値

指定された文字列の最後に、半角スペースを加えた文字列を返します。

参照

[RSet\\$関数](#), [Space\\$関数](#)

LSet\$関数使用例

```
temp$ = "123"  
temp$ = LSet$(temp$, 10)   ' temp$ = "123      "
```

3.15.31 LShift関数

数値データを指定したビット数だけ左シフトします。

書式

LShift(数値, シフトビット数)

パラメーター

数値

シフトする整数値を指定します。

シフトビット数

左シフトするビット数 (0～31の整数値)を指定します。

戻り値

指定した数値を、指定したビット数だけ左シフトした結果を返します。

解説

指定数値を左 (桁の大きい方)に、指定ビット数シフトします。シフトした分の下位のビットは、常に0がセットされます。

Lshiftは数値 * 2シフトビット数 (数値を2シフトビット数の回数だけ掛け算する)のと同じです。

注意

- 数値データ型

数値型は数種類あります。LshiftはByte型, Double型, Int32型, Integer型, Long型, Real型, Short型, UByte型, UInt32型, UShort型の数値に使うことができます。

参照

[And 演算子](#), [LShift64関数](#), [Not 演算子](#), [Or 演算子](#), [RShift関数](#), [RShift64関数](#), [Xor 演算子](#)

LShift関数使用例

```
Function lshiftst
  Integer i
  Integer num, snum
  num = 1
  For i = 1 to 10
    Print "i =", i
    snum = LShift(num, i)
    Print "The shifted num is ", snum
  Next i
Fend
```

コマンドウィンドウからLshift関数の結果を返す、その他の例です。

```
> Print LShift(2,2)
8
> Print LShift(5,1)
10
> Print LShift(3,2)
12
>
```

3.15.32 LShift64関数

数値データを指定したビット数だけ左シフトします。

書式

LShift64(数値, シフトビット数)

パラメーター

数値

シフトする整数値を指定します。

シフトビット数

左シフトするビット数 (0～63の整数値)を指定します。

戻り値

指定した数値を、指定したビット数だけ左シフトした結果を返します。

解説

指定数値を左 (桁の大きい方)に、指定ビット数シフトします。シフトした分の下位のビットは、常に0がセットされます。

Lshift64は数値 * 2シフトビット数 (数値を2シフトビット数の回数だけ掛け算する)のと同じです。

注意

- 数値データ型

数値型は数種類あります。Lshift64はInt64型, UInt64型の数値に使うことができます。

参照

[And 演算子](#), [LShift関数](#), [Not 演算子](#), [Or 演算子](#), [RShift関数](#), [RShift64関数](#), [Xor 演算子](#)

LShift64関数使用例

```
Function lshiftst
    Int64 i
    Int64 num, snum
    num = 1
    For i = 1 to 10
        Print "i =", i
        snum = LShift64(num, i)
        Print "The shifted num is ", snum
    Next i
Fend
```

コマンドウィンドウからLshift64関数の結果を返す、その他の例です。

```
> Print LShift64(2,2)
8
> Print LShift64(5,1)
10
> Print LShift64(3,2)
12
>
```

3.15.33 LTrim\$関数

文字列の左側半角スペースを削除して返します。

書式

LTrim\$ (文字列)

パラメーター

文字列

文字列式を指定します。

戻り値

左側の半角スペースを削除した文字列を返します。

参照

[RTrim\\$関数](#), [Trim\\$関数](#)

LTrim\$関数使用例

```
str$ = "  data  "  
str$ = LTrim$(str$) ' str$ = "data  "
```

3.16 M

3.16.1 Mask 演算子

Wait命令の条件を示す値をビット単位でマスクします。

書式

Wait 値1 Mask 値2

パラメーター

値1

Waitの入力条件を示す値を指定します。

値2

resultで返す数値を指定します。

解説

Mask演算子は、Wait入力条件を示す値をビットAnd演算します。

参照

[Wait](#)

Mask 演算子使用例

'入力ポート0の下位3ビットが1になるまで待機

```
Wait In(0) Mask 7 = 1
```

3.16.2 MCal

インクリメンタルエンコーダーのロボットの原点復帰(メカ原点の検出)を行います。

書式

MCal

解説

インクリメンタル型エンコーダーのロボットでは原点復帰(メカ原点の検出)をしなければなりません。この原点復帰は、電源投入後に実行してください。原点復帰を行う前に、動作コマンドや、その他の現在位置データを必要とするコマンドを実行するとエラーになります。

原点復帰は、MCordrコマンドで指定された関節順序で実行されます。工場出荷時のMCordr初期値は、マニピュレーターの機種によって異なりますので、詳細は以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

起こりやすいエラー

- MCal実行前に動作命令を実行しようとしたとき

原点復帰を行う前に、動作コマンドや、その他の現在位置データを必要とするコマンド(例えばPlist*命令など)を実行するとエラーになります。

- アブソリュートエンコーダー搭載のロボット

アブソリュートエンコーダー搭載のロボットでは、MCalは必要ありません。

ロボット設置時の注意

- 第3関節の原点復帰のためのスペース

第3関節が原点復帰するときには、まず上昇移動してから下降して原点に停止します。そのため、第3関節が原点復帰できるスペースを確保しておくことが非常に重要です。推奨値としては、Z上限値の上部に6mm以上のスペースを確保しておくことをお勧めします。(ロボット上部6mm間のスペースにツールや治具をつけないでください。)

参照

[Hofs](#), [Home](#), [Hordr](#), [MCordr](#)

MCal使用例

モニターウィンドウからの操作例です。

```
> Motor On
> Mcal
>
```


3.16.3 MCalComplete関数

MCalの状態を返します。

書式

MCalComplete

戻り値

MCalが正常に動作した場合はTrue、それ以外はFalseを返します。

参照

[MCal](#)

MCalComplete関数使用例

```
If Not MCalComplete Then  
    MCal  
EndIf
```

3.16.4 MCordr

MCalによる原点復帰時の、関節動作順序の指定と表示を行います。インクリメンタルエンコーダー搭載のロボットのみ、必要です。

書式

(1) MCordr 設定値1, 設定値2, 設定値3, 設定値4[, 設定値5][, 設定値6][, 設定値7][, 設定値8][, 設定値9]

(2) MCordr

パラメーター

設定値1

MCalプロセスの第1ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値2

MCalプロセスの第2ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値3

MCalプロセスの第3ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値4

MCalプロセスの第4ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値5

MCalプロセスの第5ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値6

MCalプロセスの第6ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値7

MCalプロセスの第7ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値8

MCalプロセスの第8ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

設定値9

MCalプロセスの第9ステップで原点復帰する関節 (0～9)を、ビットパターン (2進数の値)で指定します。
(下記ビットパターン表を参照してください。)

戻り値

パラメーターを省略すると現在のメカ原点復帰順序を表示します。

解説

電源オン時、アームを動作させる前には、必ずMCal命令を実行してください。MCal命令が実行されると各関節は、それぞれの原点復帰位置に移動します。

MCalコマンド実行時の関節の動作順序を指定します。設定値1で指定された関節が動作し、原点復帰を終了したら、次に設定値2で指定された関節が動作、というように順次、設定値3の関節、設定値4の関節と原点復帰します。

MCordr命令の意義は、原点復帰時の各関節の復帰順序を、ユーザーが変更できるようにすることです。復帰順序は、9つのステップに分かれて設定されています。ユーザーはMCordrにより各ステップで復帰する関節を指定することができます。各ステップで復帰する関節は、複数を指定することも可能です。しかし、一般的には、第3関節を最初にステップ1で移動させ、他の関節をそれ以降のステップで復帰させることをお勧めします。(注意を参照してください。)

MCordr命令では、9つのステップに対しビットパターンが指定されなければなりません。各関節に対し、ビットパターンが定められています。あるステップでビットが"1"であれば、対応する関節が原点復帰します。ビットが"0"であれば、対応する関節は、そのステップにおいては原点復帰しません。各関節に対するビットパターンは次のように割り当てられています。

ビットパターン表

関節名	ビット番号	2進数表記
第1関節	bit 0	&B000001
第2関節	bit 1	&B000010
第3関節	bit 2	&B000100
第4関節	bit 3	&B001000
第5関節	bit 4	&B010000
第6関節	bit 5	&B100000
第7関節	bit 6	&B1000000
第8関節	bit 7	&B10000000
第9関節	bit 8	&B100000000

注意

■ MCordrとHordrの違い

HordrとMCordrコマンドは重要な点が異なります。MCordrは、MCalとともに用いられ、ロボットの原点復帰時の関節復帰順序を指定しますが、Hordrは、Homeとともに用いられ、ホーム位置への関節復帰順序を指定します。

■ 原点復帰位置への復帰順序のデフォルト設定

工場出荷時の設定は、次のようになっています。

- 第1ステップで第3関節 (Z)が復帰します。
- 第2ステップで、第1関節 (X), 第2関節 (Y), 第4関節 (U)が同時に原点復帰位置に復帰します。
- 第3および第4ステップは使われていません。デフォルト値は次のとおりです。

```
MCordr &B0100, &B1011, 0, 0
```

■ 通常第3関節 (Z)を先に原点復帰させます

最初に単独で第3関節 (Z)を復帰させる理由は、水平移動が行われる前にツールをワーク表面から移動させることにあります。これは、原点復帰の際に、ツールが動作エリア内のものに干渉することを防ぐためです。

- MCordr値は保持されます

MCordr値は、ユーザーが変更を行うか、ロボットが再設定されない限り、保持されます。

参照

[MCal](#)

MCordr使用例

下記はモニターウィンドウからの4軸ロボットの操作例です。

この例では、原点復帰順序を2進数で次のように設定します

第1ステップで第3関節、第2ステップで第1関節、第3ステップで第2関節、第4ステップで第4関節が原点復帰します。

```
> MCordr  &B0100, &B0001, &B0010, &B1000
```

この例では、原点復帰順序を10進数で次のように設定します

第1ステップで第3関節、第2ステップで第1関節, 第2関節, 第4関節が同時に原点復帰します。

```
> MCordr  4, 11, 0, 0
```

次の例では、現在の原点復帰順序を10進数で表示しています。

```
>mcordr  
4, 11, 0, 0  
>
```

3.16.5 MCordr関数

MCordrパラメーター設定を返します。

書式

MCordr(設定値番号)

パラメーター

設定値番号

参照する設定値番号 (1～9の整数)を、式または数値で指定します。

戻り値

原点復帰する指定した関節の設定値を、2進数の値 (整数)で返します。

解説

MCalにより原点復帰するときの関節の動作順序を返します。

参照

[MCal](#)

MCordr関数使用例

MCordr関数を使ったプログラム例です。

```
Integer a  
a = MCordr(1)
```

3.16.6 MemIn関数

指定したメモリーI/Oポートの状態を返します。各ポートには8個のメモリービットがあります。

書式

MemIn(ポート番号)

パラメーター

ポート番号

メモリーI/Oのバイトを指定します。

戻り値

0～255までの整数を返します。戻り値は、8ビットで、各ビットがメモリーI/Oビット1個に対応します。

解説

MemInは、1度にメモリーI/Oビット8個の値を見ることができます。MemIn命令は、メモリーI/Oビット8個の状態を1つの変数に保存したり、Waitとともに用いることで、2個以上のI/Oビットに関係する特定の条件に一致するまで、プログラムを待機したりできます。

1度に8ビットの値を得ることができるので、戻り値の範囲は、0～255になります。下記の表で、それぞれの戻り値と個々のメモリーI/Oビットの状態の対応を、参照してください。

メモリーI/Oビット表 (ポート0使用時)

戻り値	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

メモリーI/Oビット表 (ポート31使用時)

戻り値	255	254	253	252	251	250	249	248
3	Off	Off	Off	Off	Off	Off	On	On
7	Off	Off	Off	Off	Off	On	On	On
32	Off	Off	On	Off	Off	Off	Off	Off
255	On	On	On	On	On	On	On	On

注意

■ MemInとMemSwの違い

MemSw命令により、メモリーI/Oビット1の値が読み取れます。MemSwの戻り値は0または1で、メモリーI/OビットがOnであるかOffであるかを表わします。MemSwはメモリーI/Oビットを個別にチェックします。MemIn命令は、メモリーI/Oビットの状態をチェックするという点で、MemSw命令に似ています。ですが、MemSw命令はメモリーI/Oを、1ビットずつ状態をチェックするのに対し、MemIn命令は8ビット同時にチェ

ックします。MemInの戻り値は、0～255までの値を返します。その値により、8ビットのうちどれがOnで、どれがOffであるかがわかります。

参照

[In関数](#), [InBCD関数](#), [Off](#), [MemOff](#), [On](#), [MemOn](#), [OpBCD](#), [Oport関数](#), [Out](#), [MemOut](#), [Sw関数](#), [MemSw関数](#), [Wait](#)

MemIn関数使用例

下記は、最初の8個のメモリーI/Oの現在の値を求めたあと、8個すべての値が"0"であることを確認して次に進むプログラムの例です。戻り値が"0"でない場合は、エラーメッセージが表示され、タスクは停止します。

```
Function main
  Integer var1

  var1 = MemIn(0)  '最初の8個のメモリーI/Oビットの値を取得
  If var1 = 0 Then
    Go P1
    Go P2
  Else
    Print "Error in initialization!"
    Print "First 8 memory I/O bits were not all set to 0"
  EndIf
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memout 0, 1
> print MemIn(0)
1
> memon 1
> print MemIn(0)
3
> memout 31, 3
> print MemIn(31)
3
> memoff 249
> print MemIn(31)
1
>
```

3.16.7 MemInW関数

メモリーI/Oポートの状態をワード単位で返します。ワードポートは16個のメモリーI/Oビットで構成されます。

書式

MemInW(ワードポート番号)

パラメーター

ワードポート番号

I/Oのワードポートを指定します。

戻り値

メモリーI/Oポートの状態 (0～65535のLong型整数)を返します。

参照

[MemIn関数](#), [MemOut](#), [MemOutW](#)

MemInW関数使用例

```
Long word0  
word0 = MemInW(0)
```


3.16.8 MemOff

メモリーI/Oの指定ビットをオフします。

書式

MemOff { ビット番号 | メモリーI/Oラベル }

パラメーター

ビット番号

メモリーI/Oのビットを整数で指定します。

メモリーI/Oラベル

メモリーI/Oラベルを指定します。

解説

MemOffは、指定したメモリーI/Oのビットをオフ (0に) します。MemSw命令で指定したメモリービットの状態チェックを行います。Wait命令もメモリービットに使われ、指定したメモリーI/O状態になるまでシステムを待たせます。

注意

- メモリーI/Oのオフ

メモリーI/Oは、コントローラー再起動時にオフされます。非常停止, 安全扉開, プログラム終了, Reset, Epson RC+ 再起動では、オフされません。

参照

[In関数](#), [MemIn関数](#), [InBCD関数](#), [Off](#), [On](#), [MemOn](#), [OpBCD](#), [Oport関数](#), [Out](#), [MemOut](#), [Sw関数](#), [MemSw関数](#), [Wait](#)

MemOff使用例

下記は、それぞれが動作命令を起動する2つのタスクを使った例です。これら2つのタスクには、お互いに相手がロボット動作命令を終了したところで順番に制御をゲインするよう、インターロック機能が働いています。そのため、2つのタスクが順番にそれぞれ指示どおりの動作ステートメントを実行することができます。MemSwはWait命令とともに用いて、安全のために、再度動作を始める前に、メモリーI/Oビット1が適切な値になるのを待ちます。MemOnとMemOffは、正確に同期するように、メモリーI/Oをオン、またはオフします。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For I = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
  Integer I
  For I = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> MemOn 1          'メモリーI/Oビット1をオン  
> Print MemSw(1)  
1  
> MemOff 1         'メモリーI/Oビット1をオフ  
> Print MemSw(1)  
0
```

3.16.9 MemOn

メモリーI/Oの指定ビットをオンします。

書式

MemOn {ビット番号 | メモリーIOラベル}

パラメーター

ビット番号

メモリーI/Oのビットを整数で指定します。

メモリーIOラベル

メモリーI/Oのラベルを指定します。

解説

MemOnは、指定したビットをオン (1) にします。MemSw命令で、指定したメモリービットの状態チェックを行います。Wait命令もメモリービットに使われ、指定したS/W状態になるまでシステムを待たせます。

注意

- メモリーI/Oのオフ

メモリーI/Oは、コントローラー再起動時にオフされます。非常停止, 安全扉開, プログラム終了, Reset, Epson RC+ 再起動では、オフされません。

参照

[In関数](#), [MemIn関数](#), [InBCD関数](#), [Off](#), [MemOff](#), [On](#), [OpBCD](#), [Oport関数](#), [Out](#), [MemOut](#), [Sw関数](#), [MemSw関数](#), [Wait](#)

MemOn使用例

下記は、それぞれが動作命令を起動する2つのタスクを使った例です。これら2つのタスクには、お互いに相手がロボット動作命令を使い終わったところで順番に制御をゲインするよう、インターロック機能が働いています。そのため、2つのタスクが順番にそれぞれ指示どおりの動作ステートメントを実行することができます。MemSwはWait命令とともに用いて、安全のため、再度動作を始める前にメモリーI/Oビット1が適切な値になるのを待ちます。MemOnとMemOffは、正確に同期するように、メモリーI/Oをオン、またはオフします。

タスク同期には、SignalとWaitsig命令を使うこともできます。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For I = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
  Integer I
  For I = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memon 1  
> print memsw(1)  
1  
> memoff 1  
> print memsw(1)  
0
```

3.16.10 MemOut

8個のメモリーI/Oビットを同時に設定します。

書式

MemOut ポート番号, 出力データ

パラメーター

ポート番号

メモリーI/Oのバイトポート番号を指定します。ポート番号は、下記のようにビットに対応します。

ポート番号	出力データ
0	0-7
1	8-15
...	...

出力データ

ポート番号で指定した出力グループの出力パターンを、0～255の整数値で指定します。16進数で表示された場合は、&H0～&HFFが範囲になります。下位桁は、下位ビット（または最初の4出力ビット）を、上位桁は、上位ビット（または次の4出力ビット）を表わします。

解説

MemOutは、設定する出力ビットを指示するポート番号と出力データの組み合わせで、8個のメモリーI/Oビットを同時に設定します。ポート番号パラメーターで、どのグループ（どの8個の出力ビット）を使うか指定します。例えば、ポート番号 = 0 のとき、出力ビット0～7を指定します。ポート番号 = 1 のとき、出力ビット8～15を指定します。

まずポート番号で8個の出力ビットを指定したのち、出力データパラメーターで特定の出力パターンを定義します。出力データパラメーターがとり得る値は0～255で、16進数あるいは10進数の整数で指定します。（&H0～&HFFまたは0～255）

下記表は、I/Oの組み合わせ例の一部を、ポート番号が"0"および"1"のときに分けて、それに対応する出力データ値とともに示したものです。

ポート番号=0の時の出力設定（出力ビット番号）

出力データ値	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	Off	On	Off	On	Off
11	Off	Off	Off	Off	On	Off	On	On

出力データ値	7	6	5	4	3	2	1	0
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

ポート番号=1の時の出力設定 (出力ビット番号)

出力データ値	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	Off	On	Off	On	Off
11	Off	Off	Off	Off	On	Off	On	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

参照

[In関数](#), [MemIn関数](#), [InBCD関数](#), [Off](#), [MemOff](#), [On](#), [MemOn](#), [OpBCD](#), [Oport関数](#), [Out](#), [Sw関数](#), [MemSw関数](#), [Wait](#)

MemOut使用例

下記は、メインタスクが"iotask"という名前のタスクを起動するプログラムです。"iotask"は、S/W メモリーI/Oビットの0~3をオンオフする簡単なタスクです。MemOut命令を使うと、S/WメモリーI/Oビットを個々にオンオフせず、1つのコマンドでそれを行うことができます。

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
Fend

Function iotask

  Do
    MemOut 0, &H

    Wait 1
    MemOut 0, &H0
    Wait 1
  Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> MemOut 1,6      'メモリーI/Oビット9と10をオン
> MemOut 2,1      'メモリーI/Oビット8をオン
> MemOut 3,91     'メモリーI/Oビット24, 25, 27, 28, 30をオン
```

3.16.11 MemOutW

メモリーI/Oポートの状態を16ビット同時にワード単位で設定します。

書式

MemOutW ワードポート番号, 出力データ

パラメーター

ワードポート番号

メモリーI/Oワードを指定します。

出力データ

メモリーI/Oデータ (0～65535の整数)を、式または数値で指定します。

解説

パラメーターのワードポート番号で指定したメモリーI/Oポートグループの状態を、指定した出力データに変更します。

参照

[MemIn関数](#), [MemInW関数](#), [MemOut](#)

MemOutW使用例

```
MemOutW 0, 25
```


3.16.12 MemSw関数

指定されたメモリーI/Oビットの状態を返します。

書式

MemSw(ビット番号)

パラメーター

ビット番号

メモリーI/Oビット番号を表わす数値で指定します。

戻り値

指定されたビットがオンのとき"1"を、オフのとき"0"を返します。

解説

MemSwは、1メモリーI/Oビットの状態を返します。MemSwには、通常MemOnとMemOff命令がセットで使われます。MemOnは指定されたビットをオンにし、MemOffは指定されたビットをオフします。

参照

[In関数](#), [MemIn関数](#), [InBCD関数](#), [Off](#), [MemOff](#), [On](#), [MemOn](#), [OpBCD](#), [Oport関数](#), [Out](#), [MemOut](#), [Sw関数](#), [Wait](#)

MemSw関数使用例

下記例では、2つのタスクが、それぞれが動作命令を起動する機能を持ちながら、一方がロボット制御をしていないときに、一方がロボット動作を制御するようインターロックしています。それにより、各々のタスクが与えられた動作ステートメントを、きちんと想定通りの順序にしたがって実行することができます。

MemSwは、Wait命令とともに使われることで、メモリーI/O ビット1が、次の動作を安全に行える適正な値になるまで待って、動作を再開します。

```
Function main
  Integer I
  MemOff 1
  Xqt 2, task2
  For I = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend

Function task2
  Integer I
  For I = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next I
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> memon 1
> print memsw(1)
1
> memoff 1
> print memsw(1)
0
```

3.16.13 MHour関数

ロボットのモーターの積算励磁時間を返します。

書式

MHour ([ロボット番号])

パラメーター

ロボット番号

励磁時間を確認するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

戻り値

モーターの積算励磁時間を実数値で返します。

参照

[Time関数](#), [Hour](#)

MHour関数使用例

```
Robot 2
Print MHour
Print MHour(1)
```

3.16.14 Mid\$関数

文字列より指定した開始位置から、指定した文字数の文字を取り出します。

書式

Mid\$(文字列, 開始位置 [, 文字数])

パラメーター

文字列

元になる文字列を指定します。

開始位置

文字列を取り出す開始位置を指定します。

文字数

文字列から取り出す文字数を指定します。省略可能です。省略すると開始位置から文字列の最後まで文字を返します。

戻り値

元となる文字列から取り出した文字列を返します。

解説

Mid\$ は元になる文字列より指定した開始位置から、文字数分の文字列を取り出します。

参照

[Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Left\\$関数](#), [Len関数](#), [Right\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Mid\$関数使用例

下記の例は、文字列"ABCDEFGHJIJ"の真中から2文字と、開始位置5の残りの文字列を取り出します。

```
Function midtest
    String basestr$, m1$, m2$
    basestr$ = "ABCDEFGHJIJ"
    m1$ = Mid$(basestr$, (Len(basestr$) / 2), 2)
    Print "The middle 2 characters are: ", m1$
    m2$ = Mid$(basestr$, 5)
    Print "The string starting at 5 is: ", m2$
Fend
```

3.16.15 Mkdir

サブディレクトリーを作成します。

書式

Mkdir ディレクトリー名

パラメーター

ディレクトリー名

作成するサブディレクトリーのパスと名称の文字列を指定します。パスの詳細は、ChDiskを参照してください。

解説

指定したパス上に、新しくディレクトリーを作成します。パスを省略した場合は、カレントディレクトリー内にサブディレクトリーを作成します。

注意

ディスクがPCの場合に実行可能です。

参照

[ChDir](#), [ChDrive](#), [RenDir](#), [Rmdir](#)

Mkdir使用例

コマンドウィンドウでの操作例

```
> Mkdir \Data  
> Mkdir \Data\PTS  
> Mkdir \TEST1\TEST2
```

3.16.16 Mod 演算子

数値式を他の数値式で割った余り値を返します。

書式

被除数 Mod 除数

パラメーター

被除数

割られる数を指定します。

除数

割る数を指定します。

結果

被除数を除数で割った余り値を返します。

解説

Modは、2つの数値を割った余り値 (整数)を得るときに使います。Mod命令の便利な使い方としては、偶数か奇数かを調べるときなどです。

Mod命令の手順は、次のとおりです

:被除数を除数で割り算します。この割り算の残りが、Mod命令の戻り値として返されます。

参照

[Abs関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Not 演算子](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#)

Mod 演算子使用例

下記の例では、ある数値 (var1)が偶数か奇数か調べます。その数値が偶数ならば、Mod命令は、結果"0"を返します。奇数ならば、結果"1"を返します。

```
Function modtest
Integer var1, result

Print "Enter an integer number:"
Input var1
result = var1 Mod 2
Print "Result = ", result
If result = 0 Then
Print "The number is EVEN"
Else
Print "The number is ODD"
EndIf
Fend
```

その他の使用例として、コマンドウィンドウからの操作例をあげます。

```
> Print 36 Mod 6
> 0

> Print 25 Mod 10
> 5
>
```

3.16.17 Motor

ロボットの全軸のモーターをオン、またはオフします。

書式

Motor {On | Off}

パラメーター

On | Off

モーターパワーをオン状態に設定する場合にはOnを、オフ状態にする場合にはOffを指定します。

解説

Motor On命令はモーターパワーをオンに設定し、全軸のブレーキを解除します。Motor Off命令はモーターパワーをオフに設定し、全軸のブレーキを設定します。

ロボットを動かすためには、モーターパワーがオンでなければなりません。

非常停止後、またはResetコマンドを必要とするエラーが発生した後は、Resetコマンドを実行してからMotor Onします。

Motor On命令を実行すると、ロボット制御パラメーターを下記の設定にします。

ロボット制御パラメーター

- SpeedとSpeedR, SpeedS の設定値: (初期値に初期化されます。)
- AccelとAccelR, AccelSの設定値: (初期値に初期化されます。)
- QPDecelR, QPDecelSの設定値: (初期値に初期化されます。)
- LimZパラメーターの設定値: (0に初期化されます。)
- CPパラメーターの設定値: (Offに初期化されます。)
- SoftCPパラメーターの設定値: (Offに初期化されます。)
- Fineの設定: (初期値に初期化されます。)
- Power Low設定: (ローパワーモードになります。)
- PTPBoostの設定値: (初期値に初期化されます。)
- TCLim, TCSpeedの設定値: (初期値に初期化されます。)
- PgLSpeed の設定値: (初期値に初期化されます。)
- PerformModeの設定値: (標準モードに初期化されます。)

参照

[Brake](#), [Power](#), [Reset](#), [SFree](#), [SLock](#)

Motor使用例

次の例はコマンドウィンドウから実行します。

```
> Motor On  
  
> Motor Off
```

3.16.18 Motor関数

指定したロボットのモーターパワーの状態を返します。

書式

Motor [(ロボット番号)]

パラメーター

ロボット番号

状態を確認するロボット番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

戻り値

- 0 = モーターオフ
- 1 = モーターオン

参照

[Motor](#)

Motor関数使用例

```
If Motor = Off Then
    Motor On
EndIf
```

3.16.19 Move

アームを現在位置から指定位置まで直線補間動作で動かします。

書式

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

パラメーター

目標座標

ポイントデータで目標位置を指定します。

ROT

ツール姿勢変化を優先して、動作の速度、加減速度を決めます。省略可能です。

ECP

外部制御点動作を指定します。省略可能です。(ECPオプション使用時のみ有効)

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

! 並列処理!

動作中にI/Oなどのコマンドを実行できるように、並列処理ステートメントを追加できます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

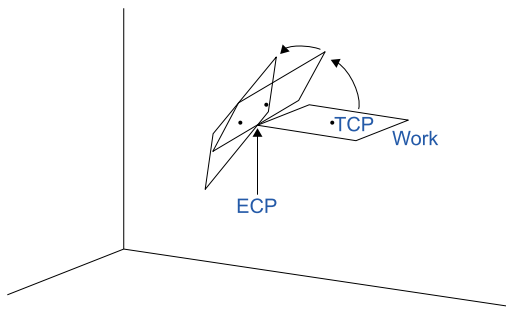
Moveはアームを現在位置から目標座標まで直線移動させます。Moveでは全関節が同時に動き始め、同時に停止します。目標座標は、Move命令の実行前にあらかじめティーチングしておきます。Moveの加減速は、AccelS命令で制御します。Moveの速度は、SpeedS命令で制御します。SpeedSで設定した値が、それぞれの関節の許容速度を1つでも超えた場合、モーターの励磁を切ってロボットは停止します。

Moveの速度と加減速度はそれぞれSpeedSとAccelSの設定値を使用します。速度と加減速度の関係については、"注意"の"MoveをCPとともに用いる"を参照してください。ただし、ROT修飾パラメーターを指定しているときの速度と加減速度は、それぞれSpeedRとAccelRの設定値を使用します。その場合、SpeedSとAccelSの設定値は無効となります。ROT修飾パラメーターを付けたとき、姿勢変化がなく、移動距離が"0"ではない場合には、エラーとなります。

ロボットのツール先端位置を、特定の座標に固定したままツール姿勢のみを変化させようとした場合や、ツール先端の移動距離に対してツール姿勢変化が大きい場合、ツール姿勢変化速度が著しく速くなる恐れがあります。これを防ぐため、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働きます。

CP動作時のツール姿勢変化速度の上限値を手動で設定したい場合は、SpeedRLimitationをオンにしてください。SpeedRLimitationをオンにした場合、CP動作時のツール姿勢変化速度が設定したSpeedRを超える場合は、ツール姿勢変化速度がSpeedRになるように動作速度を制限します。ツール姿勢変化速度が設定したSpeedRを超えない場合は、設定したSpeedSで動作します。ツール姿勢変化速度の上限値はSpeedRであらかじめ設定してください。

ECPを使っている場合、指定したECP番号に対応する外部制御点で、ワークが直線に動きます。このとき先端関節の中心は、直線をたどりません。



Move動作完了以前にロボットを減速停止させたい場合は、ユーザーはその条件を、Till修飾子を使って指定できます。ここで指定した条件は、入力ビットのうち1つをチェックすることになり、Sw命令を使って行います。ユーザーは、入力がオンかオフかをチェックし、指定した条件に基づいてアームを停止させます。この機能は、入力条件が成立するとMoveが中止される割込みに似ています。Move動作中一度も入力条件が成立しなければ、目標座標で指定された位置までアームは到達します。

Till修飾子は省略可能です。Till修飾子の詳細については、Tillコマンドを参照してください。

注意

■ Moveでできないこと

動作を行う前に動作範囲確認をすることはできません。したがって目標座標位置が許容動作範囲内にあっても、そこに至る軌道が許容動作範囲外を通過すると、アームが突然停止して、サーボに衝撃を与え、障害を生じる危険があります。これを防ぐために、Moveを高速度で実行する前に、低速度で動作範囲確認を行ってください。つまり、目標座標がアームの動作範囲内にあっても、Move動作でそこに至る軌道が、物理的にアームの許容動作範囲外にあると、アームは動かなくなります。

■ MoveをCPとともに用いる

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

■ Moveに適切な速度, 加減速度指示を与える

SpeedSとAccelS命令は、Move動作中のマニピュレーターの速度と加減速度を指定しますが、SpeedSとAccelSは、直線および円弧補間動作に対する命令であることに注意してください。PTP動作に対しては、SpeedとAccel命令が適用になります。

起こりやすいエラー

■ 直線移動距離が0となる動作を実行しようとした場合

Moveは、4自由度ロボット(水平多関節型 (RSシリーズを含む)など)のU座標値や6自由度ロボット(垂直6軸型 (Nシリーズを含む))のU, V, W座標値のみを変化させるような動作を行おうとするとエラーが発生します。この場合はROTパラメーターを使用してください。

■ 関節の制限速度を超えるエラー

指示された動作中、1関節でも許容速度を超えると、オーバースピードエラーが発生します。モーターオーバースピードエラーの場合、アームは停止し、モーターの励磁は切れます。

■ RSシリーズで原点近傍を通過する動作を実行した場合

RSシリーズでは、Moveで原点近傍を通過するような動作を行うと、オーバースピードエラーが発生することがあります。原点近傍を通過するような動作については、以下のような対策を行ってください。

- SpeedSの設定速度を下げてください。
- 原点近傍を通過しない経路に変更してください。
- Moveの代わりにGoなどのPTP動作を使用してください。

参照

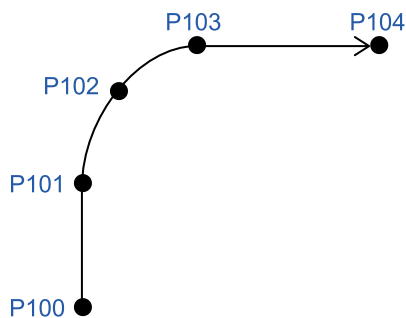
[AccelS](#), [Arc](#), [Arc3](#), [CP](#), [Go](#), [Jump](#), [Jump3](#), [Jump3CP](#), [SpeedS](#), [Sw関数](#), [Till](#), [SpeedR](#), [SpeedRLimitation](#)

Move使用例

下記の例は、P0～P1までアームをPTP動作し、その後、直線でP0まで戻す簡単な例です。プログラム後半で、アームは、入力ビット2がオンになるまでP2に向かって直線移動します。動作中に入力ビット2がオンになると、アームはP2に到達前でも減速停止し、次のプログラム命令を実行します。

```
Function movetest
  Home
  Go P0
  Go P1
  Move P0
  Move P2 Till Sw(2) = On
  If Sw(2) = On Then
    Print "Input #2 came on during the move and"
    Print "the robot stopped prior to arriving on"
    Print "point P2."
  Else
    Print "The move to P2 completed successfully."
    Print "Input #2 never came on during the move."
  EndIf
Fend
```

次にMoveをCPとともに用いる例です。下記図は、P100から始まる円弧軌道を示します。P100からP101まで直線移動し、P101から円弧が始まりP102を通過してP103に円弧を描き続けます。P103からP104までは直線動作になり、この間に減速停止します。P104で停止するまで、途中のポイントでは減速しないことに注目してください。下記のファンクションでは、このような動作をプログラムしています。



```
Function CornerArc
  Go P100
  Move P101 CP      'P101で停止しない
  Arc P102, P103 CP 'P103で停止しない
  Move P104        'P104に減速停止
Fend
```

3.16.20 MsgBox

ダイアログボックスのメッセージを表示し、ユーザーがボタンを選択するのを待ちます。

書式

MsgBox メッセージ\$ [, ボタンの種類] [, タイトル\$] [, 結果を受け取る整数]

パラメーター

メッセージ\$

表示されるメッセージ

ボタンの種類

表示するボタンの数と型, アイコンのスタイル, ボタンの見出しなどを指定する値の合計を表わす数値、または式を指定します。

最初の値のグループ(0 - 5) はボタンの数と種類を示します。

2番目のグループ(16,32,48,64)はアイコンのスタイルを示します。

3番目のグループ(0,256,512)はデフォルトのボタンを決定します。それぞれのグループから値を1つだけ選択して合計したものがボタンの種類引数の値になります。

Epson RC+には、このパラメーター用にあらかじめ決められた定数があり、下記表の値を用います。省略可能です。

■ ボタンの数と種類

定数	値	意味
MB_OK	0	[OK]ボタンだけを表示
MB_OKCANCEL	1	[OK]と[キャンセルボタン]を表示
MB_ABORTRETRYIGNORE	2	[中止][再試行][無視]ボタンを表示
MB_YESNOCANCEL	3	[はい][いいえ][キャンセルボタン]を表示
MB_YESNO	4	[はい][いいえ]ボタンを表示
MB_RETRYCANCEL	5	[再試行]と[キャンセルボタン]を表示

■ アイコンのスタイル

定数	値	意味
MB_ICONSTOP	16	Stop サイン
MB_ICONQUESTION	32	"?"マーク
MB_ICONEXCLAMATION	48	"!"マーク
MB_ICONINFORMATION	64	"i"マーク

■ デフォルトのボタン

定数	値	意味
MB_DEFBUTTON1	0	1番目のボタンはデフォルト

定数	値	意味
MB_DEFBUTTON2	256	2番目のボタンはデフォルト
MB_DEFBUTTON3	512	3番目のボタンはデフォルト

タイトル\$

ダイアログボックスのタイトルバーに表示される文を指定します。省略可能です。

結果を受け取る整数

ユーザーの選択を表わす値 (整数)を受け取る変数を指定します。Epson RC+には、このパラメーター用にあらかじめ決められた定数があります。このパラメーターで返される値は、下記表のとおりです。省略可能です。

定数	値	意味
IDOK	1	[OK]ボタンが選択された。
IDCANCEL	2	[キャンセル]ボタンが選択された。
IDABORT	3	[中止]ボタンが選択された。
IDRETRY	4	[再試行]ボタンが選択された。
IDIGNORE	5	[無視]ボタンが選択された。
IDYES	6	[はい]ボタンが選択された。
IDNO	7	[いいえ]ボタンが選択された。

解説

MsgBoxは、自動的にメッセージをフォーマットします。白紙状態にしたい場合は、メッセージにCRLFを用います。下記例を参照してください。

参照

[InputBox](#)

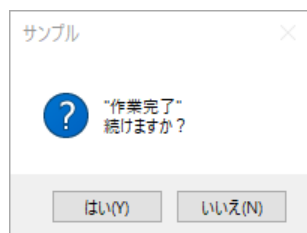
MsgBox使用例

下記の例では、ユーザーに、作業を継続するか、しないかを確認するメッセージボックスを表示します。メッセージボックスは、[はい]と[いいえ]の2つのボタンを表示します。"?マークアイコンも表示します。ユーザーがボタンを選択して、MsgBoxが返ってくると、その答えを検討します。[いいえ]が答えだと、Quitコマンドですべてのタスクを終了します。

```
Function msgtest
    String msg$, title$
    Integer mFlags, answer

    msg$ = Chr$(34) + "作業完了" + Chr$(34) + CRLF
    msg$ = msg$ + "続けますか?"
    title$ = "サンプル"
    mFlags = MB_YESNO + MB_ICONQUESTION
    MsgBox msg$, mFlags, title$, answer
    If answer = IDNO then
        Quit All
    EndIf
Fend
```

上記のコードで作られるメッセージボックスは、下記画面のようになります。



制限事項

パラメーターのメッセージ\$、およびタイトル\$ に半角カンマ", "が含まれる場合、文字列が正しく表示できません。半角カンマを含まない文字列をご使用ください。

3.16.21 MyTask関数

現在のプログラムのタスク番号を返します。

書式

MyTask

戻り値

現在のタスクのタスク番号を返します。タスク番号の範囲は以下の整数値です。

- 通常タスク: 1から32
- バックグラウンドタスク: 65から80
- トラップタスク: 257から267

解説

MyTaskは現在のプログラムのタスク番号を数値で返します。MyTask関数をプログラムに記述し実行した場合、プログラムを実行しているタスク番号を返します。

参照

[Xqt](#)

MyTask関数使用例

次の例は1から8のI/Oポートをオン、またはオフします。

```
Function main
  Xqt 2, task      'タスク2を実行
  Xqt 3, task      'タスク3を実行
  Xqt 4, task      'タスク4を実行
  Xqt 5, task      'タスク5を実行
  Xqt 6, task      'タスク6を実行
  Xqt 7, task      'タスク7を実行
  Xqt 8, task      'タスク8を実行
  Call task
Fend

Function task
  Do
    On MyTask      '現在のタスク番号と同じ番号のI/Oポートをオン
    Off MyTask     '現在のタスク番号と同じ番号のI/Oポートをオフ
  Loop
Fend
```

3.17 N

3.17.1 Next

ForとNext命令は、組み合わせて用いてループを作成します。ForとNextの間にある一連の命令は、ユーザーが指定する回数だけ繰り返します。

書式

For 変数名1 = 初期値To終了値 [Step増分値] ステートメント Next変数名1

パラメーター

変数名1

For/Nextループで、繰り返しデータを代入するカウンター変数名を指定します。通常は整数変数として定義されますが、実数を指定することもできます。

初期値

指定した変数に、ループの最初に代入する数値を、式または直接数値で指定します。

終了値

ループの終了を表わす数値を、式または直接数値で指定します。この値が成立するとFor/Nextループは終了し、Next命令の次のステートメントの実行に移ります。

増分値

For/Nextループ1回ごとに変数を増分する値を、式または直接数値で指定します。この値は、正負いずれの値でも指定することができます。しかし負の値を指定するときは、初期値>終了値 となるように設定してください。このパラメーターが省略されていると、自動的に"1"を増分値として増減します。省略可能です。

ステートメント

For/Nextループに挿入するSPEL+のステートメントを記述します。

解説

For/Nextは、ループ内にある一連のステートメントを、指定された回数だけ繰り返します。ループは、Forステートメントで始まり、Nextステートメントで終わります。変数でループ内のステートメントが繰り返された回数を数えます。

初期値は、カウンターの最初の数値です。終了値と増分値の設定が正しければ、正負いずれの値でもかまいません。

終了値は、カウンターの最後の数値です。この値に達すると、For/Nextループは終了し、プログラムの制御は、Next命令の次の命令に移ります。

Next命令に達するまで、プログラムは、Forステートメントの次のステートメントを実行します。カウンター変数は、増分値で設定した値だけ、増減していきます。増分値の設定がないと、"1"で増減します。

次にカウンター変数の値を終了値と比較します。もしカウンター変数の値が終了値より少ないか、あるいは同じ場合、Forステートメント後の命令の実行に戻ります。カウンター変数の値が終了値より大きければ、For/Nextループの次の実行に移り、Next命令直後の命令に継続します。

注意

■ 負の値の増分値

増分値が負の値の場合、カウンター変数の値はループが行われるごとに減っていきます。この場合、初期値は終了値よりも必ず大きくしてください。

参照[For...Next](#)**For/Next使用例**

```
Function fornext
  Integer ctr
  For ctr = 1 to 10
    Go Pctr
  Next ctr
  '
  For ctr = 10 to 1 Step -1
    Go Pctr
  Next ctr
Fend
```


3.17.2 Not 演算子

数値をビット単位で補数を取ります。

書式

Not 演算数

パラメーター

演算数

整数値を指定します。

結果

演算数の1の補数を返します。

解説

Not関数は演算数をビット単位で補数を取ります。結果のビットは、対応する演算数のビットを反転します。0のビットは1に、1ビットは0に変換します。

参照

[Abs関数](#), [And 演算子](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [LShift関数](#), [Mod 演算子](#), [Or 演算子](#), [RShift関数](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#), [Xor 演算子](#)

Not使用例

コマンドウィンドウでNotを使った例です。

```
>print not(1)
-2
>
```

3.18 O

3.18.1 Off

ビット番号で指定された出力ビットをオフ、または指定時間オフした後オンします。

書式

Off { 出力ビット番号 | 出力ラベル } [, 時間] [, 非同期指定] [, Forced]

パラメーター

出力ビット番号

オフするI/O出力ビットを整数値で指定します。

出力ラベル

出力ラベルを指定します。

時間

出力ビットがオフしている時間を、秒で指定します。指定された時間経過後、出力ビットは再びオンします。オフ時間は、0.01秒以上10秒以下で指定してください。省略可能です。

非同期指定

タイマーが設定されると、非同期指定を用いて次のコマンドを実行するタイミングを指定します。省略可能です。

- 0 - 出力ビットがオフすると同時に、次の命令を実行します。
- 1 - デフォルト設定。指定時間オフした後オンして、次の命令を実行します。

Forced

省略可能です。通常は省略します。

解説

Offは、指定した出力ビットをオフ (または0に) 戻します。

時間パラメーターが指定されると、出力ビット番号で指定された出力ビットは、指定時間オフした後、再びオンします。もしオフ実行前に、すでに出力ビットがオフになっていれば、指定時間経過後にオンします。

非同期指定設定は、指定時間の設定が次のようになっているときに有効です。

- 1: 出力ビットをオフし、指定時間後再度オンして次のコマンドを実行します。(非同期指定の設定は、これがデフォルト設定となっています。このパラメーターを省略すると"1"を設定したことになります。)
- 0: 出力ビットをオフし、同時に次のコマンドを実行します。

注意

- リモート制御用に割り当てられた出力ビット

システム出力用に設定された出力ビットを指定するとエラーになります。リモート出力ビットは、システムの状態に応じて自動的にオンオフします。

- 非常停止が発生したとき

Epson RC+ では、非常停止が発生すると、すべての出力ビットがオフします。非常停止の場合でも設定を保持したい場合には、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートをOFF]チェックボックスで設定し直すことができます。

- Forcedフラグ

NoPauseタスク, NoEmgAbort タスク(Xqt 時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)から、非常停止中や安全扉開時にI/O出力のオフを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InBCD関数](#), [MemOn](#), [MemOff](#), [MemOut](#), [MemSw関数](#), [OpBCD](#), [Oport関数](#), [Out](#), [Wait](#)

Off使用例

下記例では、メインタスクが、"iotask"という名前のタスクを起動させます。"iotask"は、出力ビット1と2を別々にオン、またはオフし、10秒後に再度実行する簡単なタスクです。

```
Function main
  Xqt 2, iotask
  Go P1
  .
  .
  .
Fend

Function iotask
  Do
    On 1
    On 2
    Off 1
    Off 2
    Wait 10
  Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> on 1
> off 1, 10      '出力1をオフし、10秒後再度オン
> on 2
> off 2
```

3.18.2 OLAccel

過負荷率に応じた加減速度の自動調整を設定します。

書式

OLAccel {On | Off}

パラメーター

On | Off

- On: 過負荷率に応じた加減速度の自動調整を有効にします。
- Off: 過負荷率に応じた加減速度の自動調整を解除します。

解説

OLAccel は、ロボットの過負荷率 (OLRate)に応じた加減速度の自動調整機能を有効にするかどうかの設定をします。OLAccelをOnすると、PTP動作命令の実行時に、その時のロボット過負荷率に応じて加減速度が自動調整されます。つまり、PTP動作実行時にロボットの過負荷率が一定値を越えているとき、動作時の加減速度を自動的に落としていき、過負荷エラーの発生を未然に防ぐ働きをします。従来、過負荷エラーが発生するような高デューティな動作に対しては、お客さまがプログラミングによってロボットを停止させたり、速度や加速度を調節したりする必要がありましたが、OLAccelを使用することによって、その必要性が少なくなります。ただし、あらゆるサイクルで過負荷エラーが発生しなくなるわけではなく、非常に高デューティで負荷の大きいサイクルでは、過負荷エラーの発生を必ずしもなくすることはできません。その場合には、お客さまによるロボットの停止や、速度、加減速度の調節が必要となります。また、使用環境によっては、過負荷エラーが発生せずにロボットが動作しつづけることによって、モーターが温度上昇しオーバーヒートエラーが発生することがあります。

適正な負荷の状態では、このコマンドを使用することはありません。

テストサイクル時に、OLRate を使って、ロボットが過負荷エラーの起きやすい状態にあるかどうか、チェックできます。

下記のいずれかの場合、OLAccel値は、初期化されます。

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

注意

加減速度の自動調整機能をサポートしないロボットに対してOLAccel Onを実行すると、エラーが発生します。

参照

[OLAccel関数](#), [OLRate](#)

OLAccel使用例

```
>olaccel on
>olaccel
OLACCEL is ON

Function main
Motor On
Power High
Speed 100
```

```
Accel 100, 100
OLAccel On
Xgt 2, MonitorOLRate
Do
Jump P0
Jump P1
Loop
Fend

Function MonitorOLRate
Do
'OLRateを表示
OLRate
Wait 1
Loop
Fend
```

3.18.3 OLAcel関数

過負荷率に応じた加減速度の自動調整の状態を返します。

書式

OLAcel

戻り値

- Off = 過負荷率に応じた加減速度の自動調整が解除
- On = 過負荷率に応じた加減速度の自動調整が有効

参照

[OLAcel](#), [OLRate](#)

OLAcel関数使用例

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

3.18.4 OLRate

指定関節の過負荷率を表示します。

書式

OLRate [関節番号]

パラメーター

関節番号

1～9の整数で指定します。付加軸のS軸は8、T軸は9になります。

解説

OLRateは、指定された関節の過負荷率を表示する関数です。OLRateは、サイクルが関節に過剰な負担をかけていないかチェックします。負荷の多いサイクルで使用すると温度や電流がサーボエラーの原因となることがあります。OLRateで、ロボットがサーボエラーの起きやすい状態にあるかどうか、チェックします。

サイクル運転中、OLRateをモニターする別のタスクを実行します。もし、OLRateが1.0を超える関節があれば、サーボエラーが起きます。

サーボエラーは、負荷が過剰になったときに最も発生しやすくなります。テストサイクル時にOLRateを使って、速度や加減速度設定を確認し、実使用時にサーボエラーが発生しないよう予防的措置をとることができます。

有効値を得るために、OLRateはロボットの動作中に実行してください。

適正な負荷の状態では、このコマンドを使用することはありません。

参照

[OLRate関数](#)

OLRate使用例

```
>olrate
0.10000  0.20000
0.30000  0.40000
0.50000  0.60000

Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Do
    OLRate      'OLRateを表示
    Wait 1
  Loop
Fend
```

3.18.5 OLRate関数

指定関節の過負荷率を返します。

書式

OLRate(関節番号)

パラメーター

関節番号

1～9の整数で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

指定された関節の過負荷率を返します。戻り値は、0.0～2.0です。

解説

OLRateは、サイクルが関節に過剰な負担をかけていないかチェックします。負荷の多いサイクルで使用すると温度や電流がサーボエラーの原因となることがあります。OLRate で、ロボットがサーボエラーの起きやすい状態にあるかどうか、チェックします。

サイクル運転中、OLRateをモニターする別のタスクを実行します。もし、OLRateが1.0を超える関節があれば、サーボエラーが起きます。

サーボエラーは、負荷が過剰になったときに最も発生しやすくなります。テストサイクル時にOLRateを使って、速度や加減速度設定を確認し、実使用時にサーボエラーが発生しないよう予防的措置をとることができます。

有効値を得るために、OLRateはロボットの動作中に実行してください。

適正な負荷の状態では、このコマンドを使用することはありません。

参照

[OLRate](#)

OLRate関数使用例

```
Function main
  Power High
  Speed 50
  Accel 50, 50
  Xqt 2, MonitorOLRate
  Do
    Jump P0
    Jump P1
  Loop
Fend

Function MonitorOLRate
  Integer i
  Real olRates(4)
  Do
    For i = 1 to 4
      olRates(i) = OLRate(i)
      If olRate(i) > .5 Then
        Print "Warning: OLRate(", i, ") is over .5"
      EndIf
    Next i
  Loop
Fend
```


3.18.6 On

指定された出力ビットをオンし、指定時間後、オフします。

書式

On { 出力ビット番号 | 出力ラベル } [,時間] [,非同期指定] [,Forced]

パラメーター

出力ビット番号

オンするI/O出力ビットを整数で指定します。

出力ラベル

出力ラベルを指定します。

時間

指定した出力ビットがオンしている時間を秒で指定します。この時間が経過すると、指定出力ビットはオフします。(経過時間は、0.01秒以上を指定してください。)省略可能です。

非同期指定

時間設定を行ったとき、非同期指定で次のコマンドの実行タイミングを指定することができます。省略可能です。

- 0 -出力ビットがオンすると同時に、次の命令を実行します。この場合時間設定の最大は10秒です。
- 1 -デフォルト設定。指定時間オンした後オフして、次の命令を実行します。

Forced

省略可能です。通常は省略します。

解説

Onは、指定した出力ビットをオン (1に設定) します。時間パラメーターが設定されていると指定された出力ビットがオンし、指定時間経過後オフします。

非同期指定パラメーターの設定は、時間を設定したときに次のように用います。

- 1: 出力ビットをオンし、指定時間経過後オフして次のコマンドを実行します。(非同期指定の設定は、これがデフォルト設定となっています。このパラメーターを省略すると"1"を設定したことになります。)
- 0: 出力ビットをオンし、同時に次のコマンドを実行します。

注意

- リモートに設定された出力ビット

リモートに設定された出力ビットを指定するとエラーになります。リモート出力ビットはシステムの状態に応じて自動的にオンしたりオフしたりするようになっています。リモートに関する詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド"

リモートコネクタの個々のビットを、出力やI/Oに設定するのは、Epson RC+ の[セットアップ]メニューから[システム設定]-[コントローラー]-[リモート]パネルから行います。

- 非常停止が発生したとき

ロボットコントローラーでは、非常停止が発生すると、すべての出力ビットがオフします。非常停止の場合でも設定を保持したい場合には、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートをOFF]チェックボックスで設定し直すことができます。

- Forcedフラグ

NoPauseタスク, NoEmgAbortタスク (Xqt 時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時にI/O出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InBCD関数](#), [MemOff](#), [MemOn](#), [Off](#), [OpBCD](#), [Oport関数](#), [Out](#), [Wait](#)

On使用例

下記例では、メインタスクが、"iotask"という名前のタスクを起動させます。"iotask"は、出力ビット1と2を別々にオン、またはオフし、10秒後に再度実行する簡単なタスクです。

```
Function main
    Xqt iotask
    Go P1
    .
    .
    .
Fend

Function iotask
    Do
        On 1
        On 2
        Off 1
        Off 2
        Wait 10
    Loop
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> on 1
> off 1, 10      '出力1をオフし、10秒後再度オン
> on 2
> off 2
```

3.18.7 OnErr

エラー発生時に、エラー処理サブルーチンに制御を枝分かれさせる割込みを設定します。ユーザーがエラー処理できるようにします。

書式

OnErr GoTo {ラベル | 0}

パラメーター

ラベル

エラー発生時に移行する先のステートメントラベルを指定します。

0

OnErr設定をクリアするパラメーターを指定します。

解説

OnErrによりユーザーはエラー処理を行うことができます。OnErrが使われていないと、エラー発生時にタスクは中止され、エラーが表示されます。しかし、OnErrを使うと、自動的にエラーから回復できるように、制御をエラー処理サブルーチンに移すことができます。エラーが回復すると、制御はEResume命令で指定する戻り先に移ります。これにより、エラーが発生してもタスクの実行を中断することなく、自動的にエラー処理を行うことができます。また、複雑になりやすい問題も、常に同じやり方で自動的に処理されるので、ワークセルを格段にスムーズに働かせることができます。

OnErrコマンドがパラメーター"0"を指定して設定された場合は、現在のOnErr設定はクリアされます。(OnErr 0 実行後、もしエラーが発生すると、プログラムの実行は停止されます。)

参照

[Err関数](#), [EResume](#)

OnErr使用例

下記は、簡単なユーティリティプログラムの例で、ポイントデータP0-P399が存在するか、存在しないかをチェックします。ポイントデータが存在しないと、画面に"このポイントは存在しません"というメッセージが表示されます。このプログラムでは、CX 命令を使って、ポイントが定義されているか、定義されていないかをテストします。もし定義されていなければ、制御はエラー処理サブルーチンに移され、画面に定義されていないポイントが表示されます。

```
Function errDemo
  Integer i, errNum

  OnErr GoTo errorHandler

  For i = 0 To 399
    temp = CX(P(i))
  Next i
  Exit Function
'
'
'*****
'* Error Handler                                     *
'*****
errHandler:
  errNum = Err
  '未定義ポイントを使用していないか確認
  If errNum = 7007 Then
    Print "Point number P", i, " is undefined!"
  Else
    Print "ERROR: Error number ", errNum, " occurred while"
```

```
        Print "        trying to process point P", i, " !"
    EndIf
    EResume Next
Fend
```

3.18.8 OpBCD

2桁の10進数を2進化10進数 (BCD)に変換し、指定のバイトポートに出力します。

書式

OpBCD ポート番号, 出力データ [, Forced]

パラメーター

ポート番号

I/Oの出力バイトを指定します。指定数値は、それぞれ下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
2	16-23
3	24-31
...	...

出力データ

ポート番号で指定した出力グループの出力パターンを、0から99の整数値で指定します。2番目の桁 (1の位) は、選択されたグループのうち下位4出力ビットを、1番目の桁 (10の位)は、上位4出力ビットを表わします。

Forced

省略可能です。通常は省略します。

解説

OpBCDは、8個の出力ビットをBCDで同時に設定します。標準および拡張出力ビットは、8個ずつ1組にグループ化されています。OpBCD命令のポート番号パラメーターで、どのグループ (どの8個の出力ビット)を使うか指定します。例えば、ポート番号 = 0 のとき出力ビット0から7、ポート番号 = 1のとき 出力ビット8から15を指定します。

ポート番号 (8個の出力ビット)を選択したのち、出力データパラメーターで特定の出力パターンを定義します。出力データパラメーターは、1または2桁の値で、有効範囲は0から99です。1番目の桁 (10の位)は、ポート番号で選択した8出力ビットのうち出力の上位4ビットを、2番目の桁 (1の位)は、ポート番号で選択した8出力ビットのうち出力の下位4ビットを表わします。

BCDでは各桁の有効数値が0から9までなので、すべてのI/Oの組み合わせを満たすことができません。下記表は、ポート番号=0のときのI/Oの組み合わせ例と、それに対応する出力番号値を示します。

出力設定 (出力番号)

出力番号	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On

出力番号	7	6	5	4	3	2	1	0
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	On	Off	Off	Off	Off
11	Off	Off	Off	On	Off	Off	Off	On
99	On	Off	Off	On	On	Off	Off	On

BCDでは、10進数値でしか指定することができません。そのため、BCDを用いるOpBCD命令ではすべての出力ビットをオンすることはできないことになります。出力番号値の最大値は、各桁ともに"9"なので、OpBCDで使える最大の値は、"99"になります。上記の表を見ると、"99"ではすべての出力をオンすることができないことがわかります。出力番号値が"99"のときの出力状態は、0, 3, 4, 7がオンで、他はすべてオフになっています。

注意

OpBCD とOutの違い

SPEL+のOutとOpBCDは、下記の点で大きく異なります。

- OpBCD命令は、2桁のBCDフォーマットを用いて、指定された8ビットポートの上位4ビット、下位4ビットそれぞれの出力ビットをオン、またはオフで設定します。よって、4ビットの状態が9 (&B1001) を超える &B1010～&B1111 は設定できません。

- Out命令は、出力データの値をそのまま8ビットポートに出力します。

例： `Out 8, &hFF` 'ポート番号8 (64～71) のビットを全てOnにします。

- リモートに設定された出力ビット

リモートに設定されている出力ビットに対し、OpBCDがオンするよう指定するとエラーになります。リモートの出力ビットは、システムの状態に応じて自動的にオン、またはオフしています。リモートについての詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド"

[セットアップ]メニューから[システム設定]-[コントローラー]-[リモート]パネルからリモートコネクターの個々のビットの設定(リモートまたはI/Oに設定するなど)を行うことができます。

- 非常停止時の出力

ロボットコントローラーには、非常停止時にはすべての出力がオフする機能があります。この機能を有効、あるいはオフするには、[セットアップ]メニューから[システム設定]-[コントローラー]-[環境設定]パネルの、[非常停止で出力ポートをOFF]チェックボックスの設定を行います。

- Forcedフラグ

NoPauseタスク, NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時にI/O出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InBCD関数](#), [MemOff](#), [MemOn](#), [MemSw関数](#), [Off](#), [On](#), [Oport関数](#), [Out](#), [Sw関数](#), [Wait](#)

OpBCD使用例

下記は、メインタスクが"iotask"という名前のタスクを起動するプログラムです。"iotask"は、出力ビット1 と 2をオンしてから出力ビット0 と 3をオンする簡単なタスクです。出力ビット1と2がオンすると、出力ビット0 と 3がオフし、また前者がオフすると後者はオンします。

```
Function main
    Xqt 2, iotask
    Go P1
    .
    .
    .
Fend

Function iotask
    Do
        OpBCD 0, 6
        OpBCD 0, 9
        Wait 10
    Loop
Fend
```

下記はコマンドウィンドウからの簡単な操作例です。

```
> OpBCD 1, 6      '出力1と2をオン
> OpBCD 2, 1      '出力8をオン
> OpBCD 3, 91     '出力24, 28, 31をオン
```

3.18.9 OpenDB

データベース、Excelブックをオープンします。

書式

OpenDB #データベース番号, データベース種別 [, SQLサーバー名], データベース名

パラメーター

データベース番号

501から508の整数を指定します。

データベース種別

オープンするデータベースの種別を[SQL], [Access], [Excel]の中から選択します。

SQLサーバー名

データベース種別に[SQL]を指定した場合、SQLサーバー名を指定します。省略した場合 (LOCAL)サーバーを指定したことになります。ネットワーク上のSQLサーバーは指定できません。

データベース種別に[Access]または[Excel]を指定した場合、SQLサーバー名は指定しません。

データベース名

- データベース種別に[SQL]を指定した場合、SQLサーバー上のデータベース名を指定します。
- [Access]を指定した場合、Accessファイル名を指定します。Accessファイル名のパスを省略した場合、カレントフォルダーを検索します。詳細は、ChDiskを参照してください。
- [Excel]を指定した場合、Excelファイル名を指定します。Excelファイルに指定できる形式は、Excel 2007ブック、Excel 97-2003ブックファイルです。Excelファイル名のパスを省略した場合、カレントフォルダーを検索します。詳細は、ChDiskを参照してください。

解説

指定したデータベースを指定したファイル番号でオープンします。

指定したデータベースは、RC+がインストールされているPCのディスク上に存在しなければなりません。存在しない場合、エラーとなります。指定したファイル番号はデータベースをオープンしている間、そのデータベースを識別するために使われ、CloseDBコマンドでデータベースをクローズするまで別のデータベースを参照するために使うことはできません。ファイル番号は、データベース操作命令 (SelectDB, Print#, Input#, CloseDB)で使用します。

Microsoft office 2010 64 bit版のAccessとExcelファイルは使用できません。

注意

RC+ がインストールされているPCの接続が必要です

参照

[SelectDB関数](#), [CloseDB](#), [UpdateDB](#), [DeleteDB](#), Input #, Print #

OpenDB使用例

SQLデータベースを使用した例

下記に、SQL サーバー2000のサンプルデータベースNorthwindを使用した、テーブルからデータを読み込む簡単な例を示します。

```
Integer count, i, eid  
String Lastname$, Firstname$, Title$
```



```
OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees")
For i = 0 To count - 1
    Input #501, eid, Lastname$, Firstname$, Title$
    Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
Next
CloseDB #501
```

Accessデータベースを使用した例 下記に、Microsoft Access 2007のサンプルデータベース生徒名簿を使用した、テーブルからデータを読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, dummy$

OpenDB #502, Access, "c:\MyDataBase\生徒名簿.accdb"
count = SelectDB(#502, "生徒")
For i = 0 To count - 1
    Input #502, eid, dummy$, dummy$, Lastname$, dummy$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #502
```

Excelブックを使用した例 下記に、Microsoft Excelブック生徒名簿を使用した、シートからデータを読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$

OpenDB #503, Excel, "c:\MyDataBase\生徒名簿.xls"
count = SelectDB(#503, "[生徒$]")
For i = 0 To count - 1
    Input #503, eid, Lastname$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #503
```

3.18.10 OpenCom

RS-232Cポートをオープンします。

書式

OpenCom #ポート番号

パラメーター

ポート番号

オープンするRS-232Cのポート番号を整数値で指定します。

SPEL+制御部

1から4

PC部

1001から1008

解説

RS-232Cポートをタスクから使用可能にします。

SPEL+制御部のポートを使用するには、コントローラーにオプションのI/Oボードを装着する必要があります。

PC部のポートを使用するにはRC+の設定が必要です。詳しくは以下のマニュアル内のRC-232Cに関する記述を参照してください。

"Epson RC+ ユーザーズガイド - [セットアップ]メニュー"

参照

[ChkCom関数](#), [CloseCom](#), [SetCom](#)

OpenCom使用例

```
Integer PortNo  
  
PortNo = 1001  
OpenCom #PortNo  
Print #PortNo, "Data from COM1"  
CloseCom #PortNo
```

3.18.11 OpenCom関数

OpenComを実施しているタスク番号を取得します。

書式

OpenCom(ポート番号)

パラメーター

ポート番号

RS-232Cのポート番号を整数値で指定します。

SPEL+ 制御部

1から4

PC部

1001から1008

解説

OpenComを実施しているタスク番号を取得します。

参照

[ChkCom関数](#), [CloseCom](#), [OpenCom](#), [SetCom](#)

OpenCom関数使用例

```
Print OpenCom(PortNo)
```

3.18.12 OpenNet

TCP/IPネットワークポートをオープンします。

書式

OpenNet #ポート番号 As { Client | Server }

パラメーター

ポート番号

オープンするTCP/IPのポート番号の整数値を指定します。ポート番号の範囲は201～216です。

解説

OpenNetはネットワーク上の他のコンピューターと通信するために、TCP/IPポートをオープンします。

1つのシステムはサーバーとして、他のシステムはクライアントとしてオープンします。どちらが先に起動してもかまいません。

参照

[ChkNet関数](#), [CloseNet](#), [SetNet](#)

OpenNet使用例

2つのコントローラーのTCP/IP設定が、下記のようにになっている例です。

Controller #1:

Port: #201

Host Name: 192.168.0.2

TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
Fend
```

Controller #2:

Port: #201

Host Name: 192.168.0.1

TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
Fend
```

3.18.13 OpenNet関数

OpenNetを実施しているタスク番号を取得します。

書式

OpenNet(ポート番号)

パラメーター

ポート番号

TCP/IPのポート番号の整数値を指定します。ポート番号の範囲は201～216です。

解説

OpenNetを実施しているタスク番号を取得します。

参照

[ChkNet関数](#), [CloseNet](#), [OpenNet](#), [SetNet](#)

OpenNet関数使用例

```
Print OpenNet (PortNo)
```

3.18.14 Oport関数

指定された出力ビットの状態を返す関数です。

書式

Oport(出力ビット番号)

パラメーター

出力ビット番号

I/Oの出力ビットを指定します。

戻り値

I/Oの指定出力ビットの状態を0または1の整数で返します。

- 0: オフ
- 1: オン

解説

Oportは、出力ビットの状態チェックを行います。入力ビットに対するSw命令の機能とよく似た働きをします。Oportが最もよく使われるのは、フィーダーやコンベヤーやグリッパーなどです。またI/Oを通じて機能するその他のデバイスのホストなどに接続されている出力ビットの状態チェックにも使われます。Oport関数でチェックされる状態は、1または0の値で返されます。これらの値は、指定された出力ビットがオンかオフかを示します。

注意

- Oport と Swの違い

OportとSw命令には、重要な違いがあります。両者ともI/O状態のチェックに使われますが、チェックするI/Oのタイプが異なります。Sw命令では入力ビットのチェックを行います。Oport命令は、標準および拡張ハードウェアの出力ビットのチェックを行います。これらのハードウェアポートは、コントローラー外部のデバイスと交信する独立した出力ビットです。

参照

[In関数](#), [InBCD関数](#), [MemIn関数](#), [MemOff](#), [MemOn](#), [MemOut](#), [MemSw関数](#), [Off](#), [On](#), [OpBCD](#), [Out](#), [Sw関数](#), [Wait](#)

Oport関数使用例

下記は、出力ビット5をオンし、それがオンしているかチェックしてからプログラムを継続する例です。

```
Function main
  TMOut 10
  OnErr errchk
  Integer errnum
  On 5      '出力5をオン
  Wait Oport(5)
  Call mkpart1
  Exit Function

errchk:
  errnum = Err(0)
  If errnum = 94 Then
    Print "TIME Out Error Occurred during period"
    Print "waiting for Oport to come on. Check"
    Print "Output #5 for proper operation. Then"
    Print "restart this program."
  Else
```

```
        Print "ERROR number ", errnum, "Occurred"  
        Print "Program stopped due to errors!"  
    EndIf  
    Exit Function  
Fend
```

コマンドウィンドウからの簡単な操作例です。

```
> On 1  
> Print Oport(1)  
1  
> Off 1  
> Print Oport(1)  
0  
>
```

3.18.15 Or 演算子

ビット単位または論理で、2つの値をOr演算します。

書式

値1 Or 値2

パラメーター

値1

整数値またはBoolean値を指定します。

値2

整数値またはBoolean値を指定します。

結果

整数値で指定した場合は、値をビット単位でOr演算した結果を返します。Boolean値で指定した場合は、論理Or演算の結果を返します。

解説

整数値の場合、Or演算子は、演算数をビット単位でOr演算します。演算結果は、2つの演算数の各ビットをOrした値です。

Boolean値の場合、値のどちらか一方が真 (True)であれば、演算結果は真 (True)になります。

参照

[And 演算子](#), [LShift関数](#), [Mod 演算子](#), [Not 演算子](#), [RShift関数](#), [Xor 演算子](#)

Or 演算子使用例

次の例はビット単位Or演算子の場合です。

```
>print 1 Or 2
3
>
```

次の例は論理Or演算子の場合です。

```
If a = 1 Or b = 2 Then
c = 3
EndIf
```


3.18.16 Out

8個の出力ビットを同時に設定 (出力) します。

書式

Out ポート番号, 出力データ [, Forced]

パラメーター

ポート番号

I/O出力バイトを指定します。指定数値は、下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
...	...

出力データ

ポート番号で指定した出力グループの出力パターンを、0～255の整数値で指定します。16進数で表示された場合は、&H0から&HFFが範囲になります。下位桁は、下位ビット (または最初の4出力ビット) を、上位桁は、上位ビット (または次の4出力ビット) を表わします。

Forced

省略可能です。通常は省略します。

解説

Outは、ポート番号と出力データの組み合わせで8個の出力ビットを同時に設定します。ポート番号パラメーターで、どのグループ (どの8個の出力ビット) を使うか指定します。例えば、ポート番号=0 のとき出力ビット0～7、ポート番号=1のとき出力ビット8～15を指定します。

まずポート番号で8個の出力ビットを指定したのち、出力データパラメーターで特定の出力パターンを指定します。出力データパラメーターがとる値は0～255で、16進数あるいは10進数の整数で指定します。(&H0～&HFF、または0～255)

下記表は、I/Oの組み合わせ例の一部を、それに対応する出力データ値とともに、ポート番号が"0"および"1"のときに分けて示したものです。

ポート番号=0 の時の出力設定 (出力ビット番号)

出力データ値	7	6	5	4	3	2	1	0
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	Off	On	Off	On	Off

出力データ値	7	6	5	4	3	2	1	0
11	Off	Off	Off	Off	On	Off	On	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

ポート番号=1の時の出力設定 (出力ビット番号)

出力データ値	15	14	13	12	11	10	9	8
01	Off	Off	Off	Off	Off	Off	Off	On
02	Off	Off	Off	Off	Off	Off	On	Off
03	Off	Off	Off	Off	Off	Off	On	On
08	Off	Off	Off	Off	On	Off	Off	Off
09	Off	Off	Off	Off	On	Off	Off	On
10	Off	Off	Off	Off	On	Off	On	Off
11	Off	Off	Off	Off	On	Off	On	On
99	Off	On	On	Off	Off	Off	On	On
255	On	On	On	On	On	On	On	On

注意

■ OpBCDとOutの違い

SPEL+のOutとOpBCDは、下記の2つの点で大きく異なります。

- OpBCD命令は、オン、またはオフする8個の出力ビットを指定するのに2進10進 (BCD)フォーマットを使います。2進10進フォーマットでは、&HA, &HB, &HC, &HD, &HE そして&HF などの値は使えないので、8個の出力ビットの組み合わせすべてを満たすことができません
- Out命令は、出力をオン、またはオフする8ビットの値に0～255の値を使います (OpBCDでは0～99です)。これにより、8ビット出力グループの可能な組み合わせすべてを満たせることになり、ユーザーの仕様に合わせて指定することができます。

■ Forcedフラグ

NoPauseタスク, NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時にI/O出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InBCD関数](#), [MemOff](#), [MemOn](#), [MemOut](#), [MemSw関数](#), [Off](#), [On](#), [Oport関数](#), [Sw関数](#), [Wait](#)

Out使用例

下記は、メインタスクが"iotask"という名前のタスクを起動するプログラムです。"iotask"は、出力ビット0～3をオ

ンオフする簡単なタスクです。Out命令を使うと、出力ビットを個々にオンオフせず、1つのコマンドでそれを行うことができます。

```
Function main
    Xqt iotask
    Do
        Go P1
        Go P2
    Loop
Fend

Function iotask
    Do
        Out 0, &H0F
        Out 0, &H00
        Wait 10
    Loop
Fend
```

コマンドウィンドウからの簡単な操作例です。

```
> Out 1,6      '出力9と10をオン
> Out 2,1      '出力8をオン
> Out 3,91     '出力24, 25, 27, 28, 30をオン
```

3.18.17 Out関数

出力ポートの状態をバイト単位で返します。

書式

Out(ポート番号)

パラメーター

ポート番号

I/O出力バイトを指定します。指定数値は、下記のように出力ビットに対応します。

ポート番号	出力ビット
0	0-7
1	8-15
...	...

戻り値

指定した出力ポートの状態をバイト単位で返します。

参照

[Out](#)

Out関数使用例

```
Print Out(0)
```

3.18.18 OutReal

実数値の出力データを32ビット浮動小数点データ (IEEE754準拠)として出力ポート2ワード (32ビット)の状態を設定します。

書式

OutReal ワードポート番号, 出力データ [,Forced]

パラメーター

ワードポート番号

I/O出力ワードを指定します。

出力データ

出力データ (Real型実数)を、式または数値で指定します。

Forced

省略可能です。通常は省略します。

解説

ワードポート番号で指定した出力ワードポートと次の出力ワードポートへ指定IEEE754のReal値を出力します。ワードポート番号パラメーターには、出力ワードラベルを使用できます。

注意

■ Forcedフラグ

NoPauseタスク、NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)から、非常停止中や安全扉開時にI/O出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InW関数](#), [InBCD関数](#), [InReal関数](#), [Out](#), [OutW](#), [OpBCD](#), [OutReal関数](#)

OutReal使用例

```
OutReal 32, 2.543
```

3.18.19 OutReal関数

出力ポートの状態を32ビット浮動小数点データ (IEEE754準拠)として取得します。

書式

OutReal(ワードポート番号)

パラメーター

ワードポート番号

I/O出力ワードを指定します。

戻り値

指定した出力ポートの状態を32ビット浮動小数点データ (IEEE754準拠)で返します。

参照

[In関数](#), [InW関数](#), [InBCD関数](#), [InReal関数](#), [Out](#), [OutW](#), [OpBCD](#), [OutReal](#)

OutReal関数使用例

```
Real rdata01  
  
rdata01 = OutReal(0)
```

3.18.20 OutW

出力ポートの状態を16ビット同時にワード単位で設定します。

書式

OutW ワードポート番号, 出力データ [, Forced]

パラメーター

ワードポート番号

I/O出力ワードを指定します。

出力データ

出力データ (0から65535の整数)を、式または数値で指定します。

Forced

省略可能です。通常は省略します。

解説

パラメーターのワードポート番号で指定したユーザーI/Oの出力ポートグループの状態を、指定した出力データに変更します。

注意

- Forcedフラグ

NoPauseタスク, NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)、およびバックグラウンドタスクから、非常停止中や安全扉開時にI/O出力のオンを行う場合にこのフラグを指定します。

非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

参照

[In関数](#), [InW関数](#), [Out](#)

OutW使用例

```
OutW 0, 25
```

3.18.21 OutW関数

出力ポートの状態をワード単位 (2バイト) で返します。

書式

OutW(ワードポート番号)

パラメーター

ワードポート番号

I/O出力ワードを指定します。

戻り値

指定した出力ポートの状態を16ビットで返します。

参照

[OutW](#)

OutW関数使用例

```
OutW 0, &H1010
```


3.19 P

3.19.1 P# (1. ポイント定義)

ポイントデータを定義します。

書式

Pポイント番号 = ポイント指定

ポイントラベル = ポイント指定

パラメーター

ポイント番号

数値または括弧付きの式で指定します。

- P番号
- P(式)

ポイントラベル

ポイントラベルを指定します。

ポイント指定

下記のポイントデータで指定します。

Pポイント番号, ポイントラベル, Here, Pallet, ポイントデータ関数 (Here関数, XY関数, JA関数, Pulse関数など)

詳細は、「P# (2. ポイント指定)」を参照してください。

解説

他のポイントまたはポイント式を代入することによってポイントデータを定義します。

参照

[Local](#), [Pallet](#), [PDef関数](#), [PDel](#), [PList](#)

ポイントの代入例

下記はコマンドウィンドウからの操作例です。

P1に座標データを設定します。

```
> P1 = XY(300,200,-50,100)
```

左アームの姿勢を指定します:

```
> P2 = XY(-400,200,-80,100)/L
```

P2のX座標データに20を加算し、その値をP3として定義します。

```
> P3 = P2 +X(20)
> plist 3
P3 = XY(-380,200,-80,100)/L
```

P2のY座標データから50を減算し、Z座標データを-30とし、右アームの姿勢として、その値をP4として定義します。

```
>P4=P2 -Y(50) :Z(-30) /R
> plist 4
P4 = XY(-450,200,-30,100)/R
```

Pallet(3, 5)のU coord (U座標値)に90を加算し、P6として定義します。

```
> P5 = Here  
> P6 = pallet(3, 5) +U(90)
```

3.19.2 P# (2. ポイント指定)

ポイント指定と動作コマンドのためのポイントデータを指定します。

書式

point [{ + | - } ポイント指定] [ローカル番号] [ハンド(腕系)姿勢] [肘姿勢] [手首姿勢] [j4flag値] [j6flag値] [j1flag値] [j2flag値] [相対オフセット値] [絶対座標]

パラメーター

ポイント指定

ポイントデータを下記のいずれかで指定します。

- P番号
- P(式)
- ポイントラベル
- Pallet (パレット番号, 分割番号)
- 現在位置指定
- Here
- ポイントデータ関数
- XY関数, JA関数, Pulse関数など

ローカル番号

ローカル番号 (1～15) を指定します。番号の前に必ずスラッシュ"/"か、"@ "マークをつけてください。"/"は座標がローカルになるのを意味しています。"@ "マークは座標がローカル座標に変換されることを意味します。省略可能です。

ハンド(腕系)姿勢

水平多関節型 (RSシリーズを含む) と垂直6軸型 (Nシリーズを含む) のロボットのハンド(腕系)姿勢です。/L (左腕姿勢)、または/R (右腕姿勢) を指定します。省略可能です。

肘姿勢

垂直6軸型ロボット (Nシリーズを含む) の場合に必要なパラメーターです。/A (ABOVE) または/B (BELOW) を指定します。

手首姿勢

垂直6軸型ロボット (Nシリーズを含む) の場合に必要なパラメーターです。/F (FLIP) または/NF (NOFLIP) を指定します。

j4flag値

垂直6軸型ロボット (Nシリーズを含む) の場合に必要なパラメーターです。/J4F0 または/J4F1 を指定します。

j6flag値

垂直6軸型ロボット (Nシリーズを含む) の場合に必要なパラメーターです。/J6F0 から/J6F127 の値を指定します。

j1flag値

RSシリーズと垂直6軸型ロボット (Nシリーズを含まない) の場合に必要なパラメーターです。/J1F0 または/J1F1 を指定します。

j2flag値

RSシリーズの場合に必要なパラメーターです。/J2F0 または/J2F1 を指定します。

j1angle値

RSシリーズとNシリーズの場合に必要なパラメーターです。/J1A (実数値) を指定します。

j4angle値

Nシリーズの場合に必要なパラメーターです。/J4A (実数値)を指定します。

相対オフセット値

1つまたはそれ以上の相対座標調整を行います。省略可能です。

{+|-}{X|Y|Z|U|V|W|RZ|RY|RX|R|S|T|ST} (式)

TL オフセットは、現在のツール座標系の相対オフセットです。

{+|-}{TLX|TLY|TLZ|TLU|TLV|TLW} (式)

絶対座標

1つまたはそれ以上の絶対座標を指定します。座標の前に必ずコロン":"をつけてください。この絶対座標は省略可能です。

:{X|Y|Z|U|V|W|R|S|T|ST} (式)

解説

ポイント代入や動作コマンドでは、ポイント式を使用します。(下記例参照)

```
Go P1 + P2
P1 = P2 + XY(100, 100, 0, 0)
```

相対オフセット値の使用

ベースポイントに対して1つまたはそれ以上の座標をオフセットすることができます。次の例は、現在位置からロボットをX軸の正方向に20 mm動かしています。

```
Go Here +X(20)
```

これを再度実行すると、相対移動なので、ロボットは更にX軸方向に20 mm移動します。

垂直6軸型ロボット (Nシリーズを含む)では、座標軸周りの相対回転をするときは次のように指定します。次の例は、ロボットのツール姿勢を現在の姿勢を基準にX軸の正方向に20° 回転します。

```
Go Here +RX(20)
```

相対ツールオフセットを行うことができます。

```
Go Here +TLX(20) -TLY(5.5)
```

垂直6軸型ロボット (Nシリーズを含む)では、相対オフセットで得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは相対オフセット演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。LJM関数はこのような手首部の意図しない回転を防止できます。

```
Go LJM(Here +X(20))
```

絶対座標の使用

絶対座標を用いて、ベースポイントの1つまたはそれ以上の座標を変更することができます。次の例は、ロボットをX軸20 mmの位置に移動します

```
Go Here :X(20)
```

再度実行しても、ロボットのアームはすでに前の移動で指定された絶対座標位置に来ているので、動きません。

相対オフセット値と絶対座標は、ポイントデータを一時的に修正するときに便利です。次の例では、Zの相対オフセット10 mmを指定することで、ピック位置の上空10 mmのところを動き、ゆっくりとピックアップ位置に移動します。

```
Speed fast
Jump pick +Z(10)
```

```
Speed slow
Go pick
```

次の例は、第3関節の絶対値 "0"を指定することで、現在位置から垂直上に移動させます。

```
LimZ 0
Jump Here :Z(0)
```

ローカルの使用

"/"、または"@マークを使うことでローカル番号を指定できます。"/"と"@マークには、それぞれ異なる機能があります。

座標をローカル座標系で指定するときは、"/"を使います。下記ステートメント例では、"/1"を付加することで「P1の位置は、ローカル1の 0,0,0,0」と指定しています。

```
P1 = XY(0, 0, 0, 0) /1
```

座標をローカル座標に変換するときは、"@マークを使います。下記ステートメント例では、現在位置を示すローカル座標値をP1に登録しています。

```
P1 = Here @1
```

参照

[Go](#), [LJM関数](#), [Local](#), [Pallet](#), [PDel](#), [PList](#), [Hand](#), [Elbow](#), [Wrist](#), [J4Flag](#), [J6Flag](#), [J1Flag](#), [J2Flag](#)

ポイント指定使用例

代入ステートメントや動作コマンドでの、ポイント指定の例です。

```
P1 = XY(300,200,-50,100)
P2 = P1 /R
P3 = pick /1
P4 = P5 + P6
P(i) = XY(100, 200, CZ(P100), 0)
Go P1 -X(20) :Z(-20) /R
Go Pallet(1, 1) -Y(25.5)
Move pick /R
Jump Here :Z(0)
Go Here :Z(-25.5)
Go JA(25, 0, -20, 180)
pick = XY(100, 100, -50, 0)

P1 = XY(300,200,-50,100, -90, 0)
P2 = P1 /F /B
P2 = P1 +TLV(25)
```

3.19.3 PAgl関数

指定した座標値から関節位置を計算して返します。

書式

PAgl (座標値, 関節番号)

パラメーター

座標値

関節位置を計算する座標値を指定します。

関節番号

関節番号 (1～9の整数値)を、式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

計算した関節位置を、実数値で返します。回転関節 (単位: deg), 直動関節 (単位: mm)

参照

[Agl関数](#), [CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#), [PPIs関数](#)

PAgl関数使用例

```
Real joint1  
joint1 = PAgl(P10, 1)
```

3.19.4 Pallet

パレットの定義と定義パレットの表示を行います。

書式

(1) Pallet [Outside,] パレット番号, ポイント番号1, ポイント番号2, ポイント番号3 [, ポイント番号4], 分割数1, 分割数2

(2) Pallet [Outside,] パレット番号, 分割 (横/縦)座標1, 分割 (横/縦)座標2, 分割 (横/縦)座標3 [, 分割 (横/縦)座標4], 分割数1, 分割数2

(3) Pallet

パラメーター

Outside

指定した行および列の範囲外にアクセス可能なパレットを作成します。省略可能です。

パレット番号

パレット番号 (0～15の整数)を式または数値で指定します。

ポイント番号1～3

パレット定義 (標準の3点定義)に使用するポイント変数を指定します。

ポイント番号4

4点定義する場合に、ポイント番号1～3とともに用います。省略可能です。

分割数1

パレットのポイント番号1 (分割 (横/縦)座標1)とポイント番号2 (分割 (横/縦)座標2)の分割数を整数で指定します。範囲は1～32767です。(分割数1×分割数2 <32767)

分割数2

パレットのポイント番号1 (分割 (横/縦)座標1)とポイント番号3 (分割 (横/縦)座標3)の分割数を整数で指定します。範囲は1～32767です。(分割数1×分割数2 <32767)

分割 (横/縦)座標1～3

パレット定義 (標準の3点定義)に使用する座標系をポイントデータで直接指定します。

分割 (横/縦)座標4

4点定義する場合に、分割 (横/縦)座標1～3とともに用います。省略可能です。

結果

(3)パラメーターを省略すると、定義されているパレットをすべて表示します。

解説

パレットを定義します。

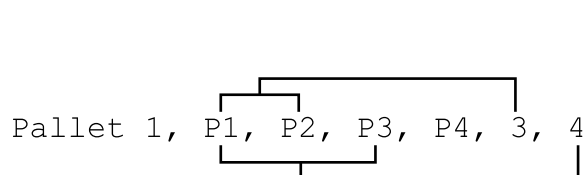
パレット上の3点または4点の位置と分割数を指定することで、パレットを定義することができます。

パレットが精度のよい方形であればコーナー4点のうち3点の位置指定だけで十分ですが、ほとんどの場合、全コーナー4点の位置を指定してパレット定義を行うことをお勧めします。

パレットを定義するには、まずコーナーの3または4点をティーチングし、以下のように進めます。

4点定義の場合:下記にP1, P2, P3およびP4 を示します。P1-P2 間には3点、P1-P3間には4点あり、合計12点を使って次の書式で定義します。

- 通常の場合:



P3	P4
10	11
7	8
4	5
1	2
P1	P2

P3	P4
1,4	2,4
1,3	2,3
1,2	2,2
1,1	2,1
P1	P2

1. ポイント番号を指定する場合の例:
パレットの青枠の位置にJumpします。

```
Pallet 1, P1, P2, P3, 3, 4
Jump Pallet (1, 7)
```

Pallet 1

10	11	12
7	8	9
4	5	6
1	2	3

2. 分割 (横/縦)座標を指定する場合の例:
パレットの青枠の位置にJumpします。

```
Pallet 1, P1, P2, P3, 3, 4
Jump Pallet (1, 2, 3)
```

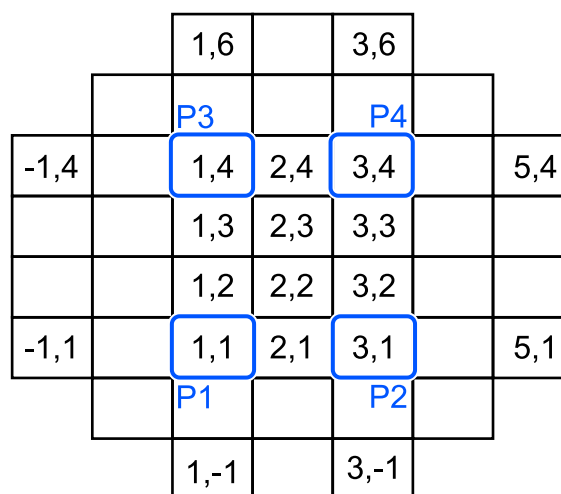
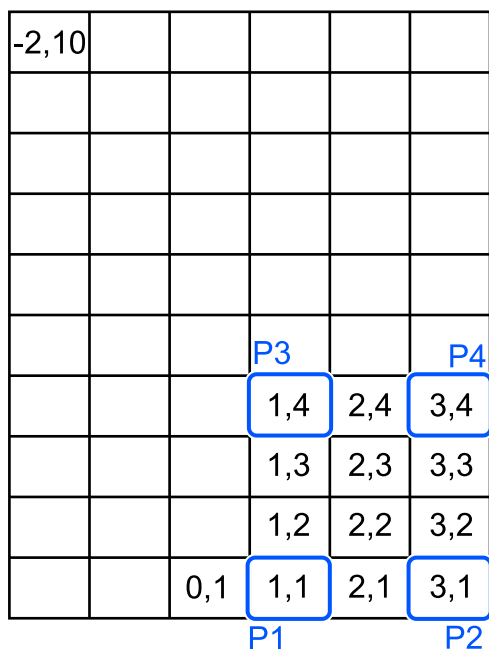
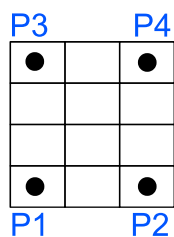
Pallet 1

1,4	2,4	3,4
1,3	2,3	3,3
1,2	2,2	3,2
1,1	2,1	3,1

- ティーチングした3または4点の外側にもアクセスしたい場合:

第1引数に"Outside"を指定することで、行および列の外側にアクセス可能なパレットを作成するよう指定できます。この場合、「分割 (横/縦)座標」でパレット内の位置を指定します。

Pallet outside 1, P1, P2, P3, P4, 3, 4



パレットの分割を示す各ポイントは、自動的に分割番号を割り当てられます。上の図の場合、P1から始まっています。この分割番号は、Pallet関数でも必要になります。

例:

```
Pallet Outside 1, P1, P2, P3, 3, 4
Jump Pallet(1, -2, 10)
```

3点指定のPallet命令により、1列のパレットも定義できます。1列の場合は、両端の2点をティーチングし、次のように入力、実行します。同一番号方向の分割数は1とします。

```
> Pallet 2, P20, P21, P20, 5, 1      '5×1パレットを定義
```

■ UVW座標値

Pallet命令に指定した3点(4点)のUVW座標値が異なる場合、ポイント番号1、および分割(横/縦)座標1のUVW座標値を使用します。

ポイント番号2～4、および分割(横/縦)座標2～4のUVW座標値は無視されます。

■ 付加軸座標値

Pallet命令に指定した3点(4点)の座標値が付加軸座標値(ST軸値)を保持している場合、付加軸座標値においても均等に分割されます。つまり、付加軸を走行軸として使用している場合、パレット定義は走行軸の動作も考慮して計算するため、走行軸の位置を考慮したロボット動作範囲を超える大きなパレットを定義することが可能です。逆にパレット定義に影響されない付加軸が定義されている場合でも、パレット定義時には付加軸の位置にも注意を払う必要があります。

参照

[Pallet関数](#)

Pallet使用例

下記はコマンドウィンドウからP1, P2, P3で定義されるパレットを設定する例です。パレット面は、12点を均等に配置、P1-P2間にパレットポイント番号1～3が並びます。この例の場合のパレットの指定位置は「2」の位置です。

```
> pallet 1, P1, P2, P3, 3, 4
> jump pallet(1, 2)      'パレットの指定位置へJump
```

この設定で以下のパレットが作成されます。

P3		P4
10	11	12
7	8	9
4	5	6
1	2	3
P1	P2	

3.19.5 Pallet関数

パレット上の位置を参照するポイントデータ関数です。

書式

(1) Pallet (パレット番号, パレット位置番号)

(2) Pallet (パレット番号, 分割横座標, 分割縦座標)

パラメーター

パレット番号

パレット番号 (0～15の整数)を数値で指定します。

パレット位置番号

分割ポイントを指定する番号 (整数)を、式または数値 (1～32767)で指定します。

分割横座標

パレットの定義で指定した横座標を、数値 (-32768～32767)で指定します。

分割縦座標

パレットの定義で指定した縦座標を、数値 (-32768～32767)で指定します。

解説

Palletは、あらかじめPalletステートメントで定義されたパレット上の1点の位置を返します。この関数を動作命令 (GoまたはJump命令など)とともに用いることにより、アームをパレット上の指定位置に動かすことができます。

パレット位置番号は、式または整数値で定義することができます。

注意

■ 垂直6軸型ロボット (Nシリーズを含む)でのパレット動作

垂直6軸型ロボット (Nシリーズを含む)では、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、手首部が意図しない方向へ回転することがあります。これは上記ポイント演算がロボットの機種に依存しない命令であるため、必要な姿勢フラグの変換を行わずに動作してしまうことが原因です。

LJM関数はこのような手首部の意図しない回転を防止できます。

■ RSシリーズでのパレット動作

同様にRSシリーズでも、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、第1アームが意図しない方向へ回転することがあります。LJM関数はこのような第1アームの意図しない回転を防止するために、ポイントデータの姿勢フラグを適切に変換します。

また、RSシリーズでは、姿勢フラグが変換された際にU軸が動作範囲外に動作しようとしてエラーが発生することがあります。LJM関数はこのようなU軸の動作範囲外エラーを防止するために、U軸の目標角度を動作範囲内の目標角度に補正します。これは、姿勢フラグ選択で2を指定することによって利用可能となります。

■ UVW座標値

Pallet命令に指定した3点 (4点)のUVW座標値が異なる場合、ポイント番号1、および座標系データ1のUVW座標値を使用します。

ポイント番号2～4、および座標系データ2～4のUVW座標値は無視されます。

■ 付加軸座標値

Pallet命令に指定した3点(4点)の座標値が付加軸座標値(ST軸値)を保持している場合、付加軸座標値においても均等に分割されます。つまり、付加軸を走行軸として使用している場合、パレット定義は走行軸の動作も考慮して計算するため、走行軸の位置を考慮したロボット動作範囲を超える大きなパレットを定義することが可能です。逆にパレット定義に影響されない付加軸が定義されている場合でも、パレット定義時には付加軸の位置にも注意を払う必要があります。

参照

[LJM関数](#), [Pallet](#)

Pallet関数使用例

下記は、ワークをパレット1からパレット2に移すプログラム例です。

Pallet 1

P3	P4
10	11 12
7	8 9
4	5 6
1	2 3
P1	P2

Pallet 2

P3	P4
9	10 11 12
5	6 7 8
1	2 3 4
P1	P2

```
Function main
  Integer index
  Pallet 1, P1, P2, P3, 3, 4      'パレット1を定義
  Pallet 2, P12, P13, P11, 4, 3  'パレット2を定義
  For index = 1 To 12
    Jump Pallet(1, index)        'パレット1のポイントインデックスへ移動
    On 1                          'ワークをピックアップ
    Wait 0.5
    Jump Pallet(2, index)        'パレット2のポイントインデックスへ移動
    Off 1                        'ワークをリリース
    Wait 0.5
  Next index
Fend
```

3.19.6 PalletClr

設定されたPalletをクリアします。

書式

PalletClr パレット番号

パラメーター

パレット番号

設定をクリアするパレット番号 (0～15の整数)を数値で指定します。

参照

[Pallet](#)

PalletClr使用例

```
PalletClr 1
```

3.19.7 ParseStr / ParseStr関数

文字列を分析し、トークンの配列を返します。

書式

ParseStr 文字列\$, トークン\$(), 区切り文字\$

トークン数 = ParseStr(入力文字列\$, トークン\$(), 区切り文字\$)

パラメーター

文字列\$

分析する文字列を指定します。

トークン\$()

トークンを格納する配列を指定します。ByRef宣言された配列は指定することができません。

区切り文字\$

1文字以上の区切り文字を指定します。

戻り値

関数として使用する場合、分析したトークンの数を返します。

解説

文字列の区切り文字が連続する箇所は、トークンが格納されず、トークン数もカウントされません。

分析したトークン数は、戻り値もしくは、UBound関数によるトークン配列の要素番号の最大値で求めてください。

参照

[Redim](#), [String](#), [UBound関数](#)

ParseStr / ParseStr関数使用例

```
String toks$(0)
Integer i

ParseStr "1 2 3 4", toks$(), " "

For i = 0 To UBound(toks$)
    Print "token ", i, " = ", toks$(i)
Next i
```

3.19.8 Pass

指定点の近傍を停止しないで通過するPTP動作です。

書式

Pass ポイント指定 [, {On | Off | MemOn | MemOff} ビット番号 [, ポイント指定 ...]] [LJM [姿勢フラグ選択]]

パラメーター

ポイント指定

P番号, P(式), ポイントラベルを指定します。

ポイントデータに抜けがなく、昇順、または降順に並んでいる場合は、2つのポイント番号をコロンで結合してP(1:5)のように指定できます。

ビット番号

オンオフするI/O出力ビットまたは、メモリーI/Oビットを指定します。整数か出力ラベルで指定します。

LJM

待避座標, 接近座標, 目標座標をLJM関数で変換します。省略可能です。

姿勢フラグ選択

LJM関数に与える姿勢フラグ選択パラメーターを指定します。省略可能です。

解説

Passは、指定ポイントデータ付近を通過してロボットアームを動かします。指定ポイントデータ自体は通過しません。

ポイントデータを指定するには、(P0, P1, ...) を使います。各ポイントはカンマで区切ってください。

動作実行中に出力ビットをオン、またはオフするには、ポイント間をカンマで区切ってOn/Offコマンドを挿入してください。On/Offが挿入された直前のポイントにロボットのアームが達する前に、On/Offが実行されます。

Passの直後にもう1つのPassステートメントが続くと、制御は、最初のPassの最終指定点付近で停止せず、次のPassに移ります。

Passの直後にPass以外の動作コマンドが続くと、ロボットはPassステートメントの最終指定点で止まりますが、Fineによる位置決めは行いません。

Passの直後に動作コマンド以外のコマンド、ステートメント、ファンクションなどが続くと、Passで指定した最終点にロボットのアームが到達する前に、それらのコマンド、ステートメント、ファンクションなどが実行されます。

目標位置へのFineによる正確な位置決めが求められる場合は、下記例のように、目標位置を指定してPassのあとにGoを入れてください。

```
Pass P5; Go P5; On 1; Move P10
```

加速値または減速値が大きいほど、アームは指定点の近くを通過します。Pass命令を使うことで、障害物を避けることができます。

LJMパラメーターを使うと、LJM関数を使用するプログラムを簡略化できます。

例えば、

```
P11 = LJM(P1, Here, 1)
P12 = LJM(P2, P11, 1)
P13 = LJM(P3, P12, 1)
Pass P11, P12, P13
```

という4行のプログラムを

```
Pass P1, P2, P3 LJM 1
```

という1行のプログラムに置き換えることができます。

LJMパラメーターは垂直6軸型ロボット (Nシリーズを含む)、およびRSシリーズロボットで有効です。

姿勢フラグ選択をデフォルト値で使用する場合は、省略が可能です。

```
Pass P1, P2, P3 LJM
```

参照

[Accel](#), [Go](#), [Jump](#), [Speed](#)

Pass使用例

下記は、Pass命令を使ってロボットアームを操作する例です。

```
Function main
  Jump P1
  Pass P2      'アームをP2に近づけP2に到達する前に次のコマンドを実行
  On 2
  Pass P3
  Pass P4
  Off 0
  Pass P5
Fend
```


3.19.9 Pause

一時停止可能なすべてのタスクを一時停止します。

書式

Pause

解説

Pauseを実行すると、一時停止可能なすべてのタスク (Xqt命令で、NoPauseやNoEmgAbortを指定しないタスク) を一時停止します。動作コマンド実行中のタスクは、すべて一時停止します。

バックグラウンドタスクは、Pauseでは一時停止しません。

注意

- QPのPauseへの影響

QP命令はPauseが実行されたとき、直ちにロボットの動作を停止するか、動作終了後にプログラムを一時停止するかを設定します。詳細はヘルプのQP命令を参照してください。

Pause使用例

次の例はPauseによりタスクを一時停止します。Pauseの行でプログラムは一時停止します。オペレーターウィンドウの[継続実行]ボタンを押してタスクを再開します。

```
Function main
  Xqt monitor
  Go P1
  On 1
  Jump P2
  Off 1
  Pause           'プログラムを一時停止
  Go P40
  Jump P50
Fend
```

3.19.10 PauseOn関数

一時停止状態 (Pause状態)を返します。

書式

PauseOn

戻り値

一時停止状態のときは"True"、それ以外のときは"False"を返します。

解説

本関数は、NoPauseタスク, NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)と、バックグラウンドタスクでのみ使用します。

参照

[ErrorOn関数](#), [EStopOn関数](#), [SafetyOn関数](#), [Wait](#), [Xqt](#)

PauseOn関数使用例

下記例は、コントローラーが一時停止になるのを監視し、一時停止が発生した場合I/Oをオンオフするプログラムです。ただし、安全扉が開いて一時停止状態になった場合はI/Oをオンオフしません。

```
Function main
    Xqt PauseMonitor, NoPause
    :
    :
Fend

Function PauseMonitor
    Boolean IsPause
    IsPause = False
    Do
        Wait 0.1
        If SafetyOn = True Then
            If IsPause = False Then
                Print "Safety On"
                IsPause = True
            EndIf
        ElseIf PauseOn = True Then
            If IsPause = False Then
                Print "InPause"
                If SafetyOn = False Then
                    Off 10
                    On 12
                EndIf
                IsPause = True
            EndIf
        Else
            If IsPause = True Then
                Print "OutPause"
                On 10
                Off 12
                IsPause = False
            EndIf
        EndIf
    Loop
Fend
```

3.19.11 PDef関数

指定したポイントデータが定義されているかを返します。

書式

PDef (ポイントデータ)

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルを指定します。互換性に関する注意点ポイントデータ引数に変数に指定できなくなりました。変数を使用する場合には、PDef(P(varName))と記述してください。

戻り値

ポイントデータが定義されている場合は"True"、それ以外は"False"を返します。

参照

[Here](#), [PDel](#)

PDef関数使用例

```
If Not PDef(1) Then
    Here P1
EndIf
Integer i
For i = 0 to 10
    If PDef (P(i)) Then
        Print "P(";i;) is defined"
    EndIf
Next
```

3.19.12 PDel

指定したポイントデータを削除します。

書式

PDel 開始ポイント番号 [, 終了ポイント番号]

パラメーター

開始ポイント番号

削除するポイントデータ範囲の開始番号を、整数値または式で指定します。

終了ポイント番号

削除するポイントデータ範囲の終了番号を、整数値または式で指定します。

解説

ロボットのメモリー上から指定したポイントデータを削除します。パラメーターの開始ポイント番号から終了ポイント番号までのポイントデータを削除します。開始ポイント番号は、終了ポイント番号より小さい値にしてください。

PDel使用例

```
> p1=10,300,-10,0/L
> p2=0,300,-40,0
> p10=-50,350,0,0
> pdel 1,2      'ポイント1と2を削除
> plist
P10 =   -50.000,   350.000,    0.000,    0.000 /R /0
> pdel 50      'ポイント50を削除
> pdel 100,200 'ポイント100から200を削除
>
```

3.19.13 PDescription

指定したポイントデータのコメントを設定します。

書式

PDescription ポイントデータ,新コメント

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルで指定します。ポイントデータ引数に変数は指定できません。変数を使用する場合には、PDescription (P(varName)), "新コメント"と記述してください。

新コメント

指定したポイントデータのコメントを指定する文字列式です。

解説

PDescriptionは、コントローラーのメモリー上の指定したポイントデータにコメントを保存します。

コントローラーのメモリー上に保存したコメントは、プロジェクトを作成したりプログラムを実行したりするとメモリーから消去されます。保存が必要な場合は、"SavePoints"を実行してポイントファイルに保存してください。

参照

[PDef関数](#), [PDescription\\$関数](#), [PLabel](#), [PLabel\\$関数](#)

PDescription使用例

```
PDescription 1, "Comment"
```

3.19.14 PDescription\$関数

指定したポイント番号に定義されたポイントのコメントを返します。

書式

PDescription\$(ポイントデータ)

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルで指定します。ポイントデータ引数に変数は指定できません。変数を使用する場合には、PDescription\$(P(varName))と記述してください。

戻り値

指定したポイント番号のコメントを文字列で返します。

参照

[PDef関数](#), [PDescription](#), [PLabel](#), [PLabel\\$関数](#)

PDescription\$関数使用例

```
Print PDescription$(1)
Print PDescription$(P(i))
```

3.19.15 PeakSpeedClear

関節のピーク速度をクリアし、初期化します。

書式

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

パラメーター

関節指定1 - 関節指定9

関節番号を整数値、または式で指定します。パラメーターを指定しない場合は、全関節のピーク速度値がクリアされます。付加軸のS軸は8、T軸は9になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

PeakSpeedClearは指定した関節のピーク速度値をクリアします。

PeakSpeed実行前に、必ずPeakSpeedClearを実行してください。

本コマンドは、PG付加軸はサポートしません。

参照

[AvgSpeed](#), [PeakSpeed](#)

PeakSpeedClear使用例

[例1] 全関節のピーク速度値をクリアしたのち、指定した関節番号の速度値を表示させるコマンド実行例です。

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
  -0.273
> PeakSpeed
  -0.273    -0.164
  -0.080     0.258
  -0.005     0.401
   0.000     0.000
   0.000
```

[例2] 垂直多関節ロボットにおいて、第1関節, 第4関節, 第5関節のピーク速度値をクリアしたのち、指定した関節番号のピーク速度値を表示させるコマンド実行例です。

```
> PeakSpeedClear 4, 1, 5
> Go P1
> PeakSpeed 1
  -0.273
> PeakSpeed 4
  0.258
```

3.19.16 PeakSpeed

指定した関節のピーク速度値を表示します。

書式

PeakSpeed [関節番号]

パラメーター

関節番号

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

結果

全関節について現在のピーク速度値を表示します。

解説

PeakSpeedは、関節速度の絶対値が最大となる速度に符号をつけた値を表示します。最大速度は1になります。ピーク速度は、-1～1までの実数値で表わされます。

PeakSpeedClearを実行してから、PeakSpeedを実行し、関節のピーク速度値を表示してください。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度からピーク速度を計算します。

本コマンドは、PG付加軸はサポートしません。

参照

[AvgSpeed](#), [PeakSpeedClear](#), [PeakSpeed関数](#)

PeakSpeed使用例

```
> PeakSpeedClear
> Go P1
> PeakSpeed 1
    -0.273
> PeakSpeed
    -0.273    0.163
    -0.080    0.258
    -0.005   -0.401
     0.000    0.000
     0.000
>
```


3.19.17 PeakSpeed関数

指定した関節のピーク速度を返します。

書式

PeakSpeed (関節番号)

パラメーター

関節番号

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

戻り値

-1～1までの実数値で返します。

解説

PeakSpeed関数は、関節速度の絶対値が最大となる速度に符号をつけた値を返します。最大速度は1になります。ピーク速度は、-1～1までの実数値で表わされます。

PeakSpeedClearを実行してから、PeakSpeedを実行し、関節のピーク速度値を表示してください。

仮想コントローラー、およびドライランの場合は、実速度ではなく指令速度からピーク速度を計算します。

本関数は、PG付加軸はサポートしません。

参照

[AvgSpeed](#), [PeakSpeedClear](#), [PeakSpeed](#)

PeakSpeed関数使用例

プログラム中にPeakSpeed関数を使用した例です。

```
Function DisplayPeakSpeed
  Integer i

  PeakSpeedClear
  Go P1
  Print "Peak Speeds:"
  For i = 1 To 6
    Print "Joint ", i, " = ", PeakSpeed (i)
  Next i
Fend
```

3.19.18 PerformMode

ロボットのモードを選択できます。

書式

(1) PerformMode [モード番号] [, ロボット番号]

(2) PerformMode

パラメーター

モード番号

設定するモード番号を整数値 (0~2) もしくは以下に示す定数で指定します。コマンドウィンドウからの実行時のみ省略可能です。

定数	値	内容
MODE_STANDARD	0	標準モードを設定します。
MODE_BOOST	1	ブーストモードを設定します。
MODE_LOW_VIBRATRION	2	低振動モードを設定します。

ロボット番号

設定するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

結果

- (1)の書式で指定すると、指定したモード番号で動作モードを設定します。
- (2)の書式で指定すると、現在選択されているロボットに設定されているモード番号を表示します。

解説

PerformModeは用途に応じて優先するロボットの性能 (モード) を変更する機能です。選択可能なモードは、以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

- 標準モード: タクトタイム, 動作デューティー, 動作停止時の振動のバランスを優先したモードです。あらゆるアプリケーションに使用できます。
- ブーストモード: 一動作の動作時間を短縮させることを優先したモードです。標準モードに比べ、動作デューティーと動作停止時の振動は悪化しますが、一動作時間が短くなります。高デューティーの用途で使用する場合、標準モードの方が、タクトタイムが短くなることがあります。

推奨アプリケーション: 搬送など

- 低振動モード: 動作停止時の振動を小さくすることを優先したモードです。標準モードに比べ、動作時間が長くなりますが、停止時の振動が小さくなります。

推奨アプリケーション: 精密部品の搬送, 組み立てなど

各モードの性能比較表

モード	比較項目		
	一動作時間 (*1)	停止時振動	動作デューティ (*2)
標準	○	○	○
ブースト	◎	△	△
低振動	△	◎	○

表中の記号は性能の度合いを表します。

○: 基準 ◎: 向上 △: やや低下

- (*1) 一動作時間とはロボットが現在位置から目標座標までにかかる移動時間です。
- (*2) 動作デューティとは、最大加減速度で過負荷エラーなく、動作できる動作時間の割合です。

注意

- 未対応機種の場合、ブーストモード、および低振動モードに変更すると、エラーが発生します。
- 対応動作指令: PTP動作 (Go, BGo, TGo, Jump, JTran, PTran, Pulse)
- CP動作の以下の性能は変化しません。
 - 軌跡精度
 - AccelS, AccelR, SpeedS, SpeedRの上限値
 - 加速度指令エラー, 速度指令エラーの発生確率
- モードが自動的に初期モード (標準モード)に変更される条件について

下表にモードが自動的に変更される条件を示します。

	モードの変化
コントローラー電源ON	初期モードに変わる
コントローラーの再起動	初期モードに変わる
モーターON	初期モードに変わる
ビルド / リビルド	変わらない
Reset実行時	変わらない
SFree実行時	初期モードに変わる

参照

[BGo](#), [Go](#), [Jump](#), [JTran](#), [PerformMode関数](#), [TGo](#)

PerformMode使用例

```
PerformMode MODE_STANDARD
Go P1
PerformMode 2
Go P2
```

3.19.19 PerformMode関数

ロボットの動作モードの状態を返します。

書式

PerformMode ([ロボット番号])

パラメーター

ロボット番号

状態を確認するロボットの番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

戻り値

現在設定されている動作モードの値を整数値で返します。

- 0 = 標準モード
- 1 = ブーストモード
- 2 = 低振動モード

参照

[PerformMode](#)

PerformMode関数使用例

```
Print PerformMode(1)
```

3.19.20 PG_FastStop

連続回転中のパルス出力軸を急停止します。

書式

PG_FastStop

解説

PG_FastStopを実行すると、現在選択されているPGロボットの連続回転動作を急停止します。通常の停止は減速停止 (PG_SlowStop)を使用してください。

参照

[PG_Scan](#), [PG_SlowStop](#)

PG_FastStop使用例

次の例はパルス出力軸を10秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_FastStop          '連続回転動作を急停止
End
```

3.19.21 PG_LSpeed

パルス出力軸の加速開始時のパルス速度および減速終了時のパルス速度を設定します。

書式

PG_LSpeed 速度設定値 [, 減速終了速度],

パラメーター

速度設定値

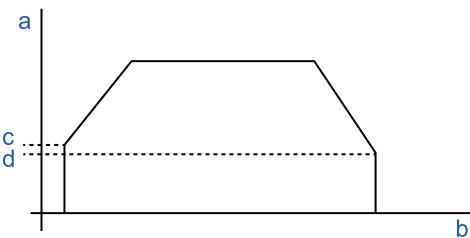
パルス速度 (1〜32767の整数、単位: pulse/sec)を式または数値で指定します。

減速終了速度

減速終了時のパルス速度 (1〜32767の整数、単位: pulse/sec)を、式または数値で指定します。

解説

PG_LSpeed は、パルス出力軸の加速開始時のパルス速度、および減速終了時のパルス速度を指定します。ステッピングモータの初速度、および終了速度を最大自起動周波数の範囲内で高く設定し、高性能なモータの性能を発揮したり、低く設定してステッピングモータの脱調を防止したりするのに使用します。デフォルト値は、300pulse/secが設定されており、通常はそのまま使用してください。



記号	説明
a	速度
b	時間
c	加速開始速度
d	減速終了速度

減速終了速度を省略すると、速度設定値が、減速終了速度の設定値となります。

下記のいずれかの場合、PG_LSpeed値は、初期化されます。

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

参照

[PG_LSpeed関数](#)

PG_LSpeed使用例

PG_LSpeed は、コマンドウィンドウでも、プログラム中でも、用いることができます。下記に両方の場合の使用例を示します。

```
Function pglspeedtst
      Motor On
      Power High
Speed 30;Accel 30,30
PG_LSpeed 1000
Go P0
Fend
```

コマンドウィンドウから PG_LSpeed値を設定する例です。

```
> PG_LSpeed 1000,1100
>
```

3.19.22 PG_LSpeed関数

現在設定されているパルス出力軸の加速開始時のパルス速度、および減速終了時のパルス速度を返します。

書式

PG_LSpeed [(パラメーター番号)]

パラメーター

パラメーター番号

下記に示す番号から、設定値番号を1つ指定します。省略した場合は1が指定されたとみなします。

- 1: 加速開始時パルス速度
- 2: 減速終了時パルス速度

戻り値

1～32767の整数値、単位: pulse/secを返します。

参照

[PG_LSpeed](#)

PG_LSpeed関数使用例

```
Integer savPGLSpeed  
  
savPGLSpeed = PG_LSpeed(1)
```


3.19.23 PG_Scan

パルス出力軸の連続回転動作を開始します。

書式

PG_Scan 回転方向

パラメーター

回転方向

連続回転の回転方向を指定します。

- 0: + (CW)方向
- 1: - (CCW)方向

解説

PG_Scanを実行すると、現在選択されているPGロボットの連続回転動作を開始します。

連続回転動作を実行するためには、ロボット設定にてPGパラメーターの連続回転を有効にする必要があります。

プログラム実行タスクが終了すると、連続回転は停止します。

参照

[PG_FastStop](#)

PG_Scan使用例

次の例はパルス出力軸を10秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  Power High
  Speed 10; Accel 10,10
  PG_Scan 0
  Wait 10
  PG_SlowStop
Fend
```

3.19.24 PG_SlowStop

連続回転中のパルス出力軸を減速停止します。

書式

PG_SlowStop

解説

PG_SlowStopを実行すると、現在選択されているPGロボットの連続回転動作を減速停止します。

参照

[PG_Scan](#), [PG_FastStop](#)

PG_SlowStop使用例

次の例はパルス出力軸を10秒間連続回転動作させて急停止します。

```
Function main
  Motor On
  PG_Scan 0
  Wait 10
  PG_SlowStop           '連続回転動作を急停止
Fend
```

3.19.25 PLabel

指定したポイントデータのラベルを設定します。

書式

PLabel ポイント番号,新ラベル

パラメーター

ポイント番号

ポイント番号を式または数値で指定します。

新ラベル

指定したポイントデータに使うラベルを指定する文字列式です。

参照

[PDef関数](#), [PDescription](#), [PDescription\\$関数](#), [PLabel\\$関数](#), [PNumber関数](#)

PLabel使用例

```
PLabel 1, "pick"
```

3.19.26 PLabel\$関数

指定したポイント番号に定義されたポイントラベルを返します。

書式

PLabel\$(ポイントデータ)

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルで指定します。互換性に関する注意点ポイントデータ引数に変数に指定できなくなりました。変数を使用する場合には、PLabel\$(P(varName))と記述してください。

参照

[PDef関数](#), [PDescription](#), [PDescription\\$関数](#), [PLabel](#), [PNumber関数](#)

PLabel\$関数使用例

```
Print PLabel$(1)
Print PLabel$(P(i))
```

3.19.27 Plane

進入検出平面の設定と表示をします。

書式

- (1) Plane 平面番号[, ロボット番号], 座標系データ
- (2) Plane 平面番号[, ロボット番号], 原点, X軸指定, Y軸指定
- (3) Plane 平面番号[, ロボット番号]
- (4) Plane

パラメーター

平面番号

進入検出平面の番号を指定します。進入検出平面は1～15の整数で、合計15個まで定義することができます。

ロボット番号

設定するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。

座標系データ

進入検出平面の座標系をポイントデータで直接指定します。

原点

進入検出平面の原点を定義するロボット座標系上の位置を、P#(整数)、またはP(式)で指定します。

X軸指定

進入検出平面のX軸上の点を定義するロボット座標系上の位置を、P#(整数)、またはP(式)で指定します。

Y軸指定

進入検出平面のY軸上の点を定義するロボット座標系上の位置を、P#(整数)、またはP(式)で指定します。

結果

- (3)の書式で指定すると、指定した平面番号の進入検出平面の設定を表示します。
- (4)の書式で指定すると、現在選択されているロボットの設定されているすべての進入検出平面の設定を表示します。

解説

Planeは、進入検出平面を設定します。進入検出平面は、現在選択しているツールに基づいて算出される手先位置が、設定した進入検出平面によって分けられる空間の一方に入っているかどうかを検出するために使用します。進入検出平面は、ロボットのベース座標系上に置かれる任意の座標系のXY平面によって設定されます。そして、その平面によって分けられる空間のうち、その座標系のZ軸+方向を含む空間内に手先位置が入ったとき進入検出平面に進入しているとみなします。

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

以下、各書式について説明します。

- (1) ベース座標系を基準とした並進量と回転量を表すポイントデータによって進入検出平面の元になる座標系を指定し、進入検出平面を設定します。

例:

```
Plane 1, XY(x, y, z, u, v, w)
Plane 1, P1
```

(2) 原点, X軸上ポイント, Y軸上ポイントの3点を指定することで、進入検出平面 (XY平面)を定義します。各ポイントのX, Y, Z座標のみを使用し、U, V, W座標は無視します。同時に右手系をなすようにZ軸が計算され、進入検出方向が設定されます。

例:

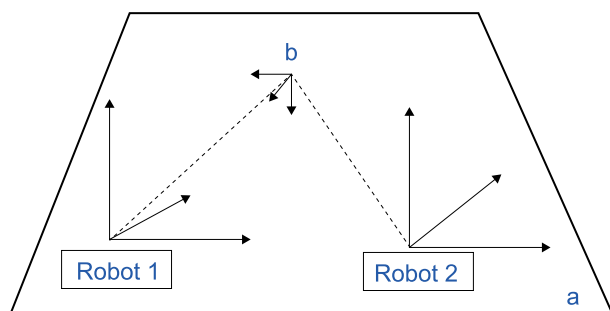
```
Plane 1, P1, P2, P3
```

(3) 指定された進入検出平面の設定を表示します。

(4) 設定されているすべての進入検出平面の設定を表示します。

進入検出平面に対する進入検出の結果は、GetRobotInsidePlane関数, InsidePlane関数を使って随時取得できます。またWaitコマンドの待ち条件式としてGetRobotInsidePlane関数を利用することもできます。また、検出結果をI/Oに出力するようにリモート出力設定することもできます。

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



記号	説明
a	進入検出平面
b	進入検出平面座標系

注意

■ ツール選択

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

■ 付加軸

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

参照

[Box](#), [GetRobotInsidePlane関数](#), [InsidePlane関数](#), [PlaneClr](#), [PlaneDef関数](#)

ヒント

- Planeをロボットマネージャーで設定する

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

Plane使用例

コマンドウィンドウからの操作例です。

ロボット座標系を基準にZ軸方向に-20mmの水平面で下向き方向を検出方向として定義する例:

```
> plane 1, xy(100, 200, -20, 90, 0, 180)
```

ロボット座標系を基準にX方向に50mm、Y方向に200mm移動し、Y軸周りに45度回転して得られる座標系が作るXY平面を進入検出平面として定義する例:

```
> plane 2, xy(50, 200, 0, 0, 45, 0)
```

ロボットのツール座標系を使って進入検出平面を直接設定する。(垂直6軸型ロボット)

```
> plane 3, here
```

3.19.28 Plane関数

設定された進入検出平面を返す関数です。

書式

Plane(平面番号[, ロボット番号])

パラメーター

平面番号

確認する進入検出平面番号 (1～15の整数)を、式または数値で指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。

戻り値

設定されている進入検出平面の元になる座標系データをポイントデータとして返します。

参照

[GetRobotInsidePlane関数](#), [InsidePlane関数](#), [Plane](#), [PlaneClr](#), [PlaneDef関数](#)

Plane関数使用例

```
P1 = Plane(1)
```


3.19.29 PlaneClr

進入検出平面をクリア (未定義)にします。

書式

PlaneClr 平面番号[, ロボット番号]

パラメーター

平面番号

設定をクリア (未定義)にする進入検出平面番号 (1～15の整数)を、式または数値で指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。

参照

[GetRobotInsidePlane関数](#), [InsidePlane関数](#), [Plane](#), [PlaneDef関数](#)

PlaneClr使用例

```
PlaneClr 1
```

3.19.30 PlaneDef関数

進入検出平面の設定状態を返します。

書式

PlaneDef(平面番号[, ロボット番号])

パラメーター

平面番号

状態を返す進入検出平面の番号 (1～15の整数)を指定します。

ロボット番号

設定するロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。

戻り値

平面番号で指定された進入検出平面が設定されている場合は"True"、設定されていない場合は"False"を返します。

参照

[GetRobotInsidePlane関数](#), [Box](#), [InsidePlane関数](#), [Plane](#), [PlaneClr](#)

PlaneDef関数使用例

```
Function DisplayPlaneDef(planeNum As Integer)

    If PlaneDef(planeNum) = False Then
        Print "Plane ", planeNum, "is not defined"
    Else
        Print "Plane 1: ",
        Print Plane(planeNum)
    EndIf
Fend
```

3.19.31 PList

現在のロボットのメモリー上のポイントデータを表示します。

書式

(1) Plist

(2) Plist ポイント番号

(3) Plist 開始ポイント番号,

(4) Plist 開始ポイント番号, 終了ポイント番号

パラメーター

ポイント番号

番号の範囲は0～999までです。

開始ポイント番号

表示する先頭のポイント番号を0～999までで指定します。

終了ポイント番号

表示する最後のポイント番号を0～999までで指定します。

結果

ポイントデータを返します。

解説

Plist は現在のロボットのメモリー上のポイントデータを表示します。

指定した範囲のポイントデータが存在しない場合、データは表示されません。

開始ポイント番号を終了ポイント番号より大きい値に指定した場合は、エラーになります。

- (1) PList: すべての位置データを表示します。
- (2) PList ポイント番号: 指定した1つのポイント番号の位置データを表示します。
- (3) PList 開始ポイント番号,: 開始ポイント番号から位置データの終わりまで表示します。
- (4) PList 開始ポイント番号,: 終了ポイント番号: 指定した開始ポイント番号から終了ポイント番号までの位置データを表示します。

PList使用例

表示される形式はロボットのタイプや付加軸の有無で変わります。

以下の例はスカラ型ロボットで、付加軸無しの場合です。

次の例は、指定したポイントデータを表示します。

```
> plist 1
P1   = XY( 200.000, 0.000, -20.000, 0.000 ) /R /0
>
```

次の例は、10～20までのポイントデータを表示します。この例では、指定した範囲内に3つのポイントデータが存在します。

```
> plist 10, 20
P10  = XY( 290.000, 0.000, -20.000, 0.000 ) /R /0
P12  = XY( 300.000, 0.000, 0.000, 0.000 ) /R /0
P20  = XY( 285.000, 10.000, -30.000, 45.000 ) /R /0
>
```

次の例は、10から終わりまでのポイントデータを表示します。

```
> plist 10,  
P10 = XY( 290.000, 0.000, -20.000, 0.000 ) /R /0  
P12 = XY( 300.000, 0.000, 0.000, 0.000 ) /R /0  
P20 = XY( 285.000, 10.000, -30.000, 45.000 ) /R /0  
P30 = XY( 310.000, 20.000, -50.000, 90.000 ) /R /0
```

3.19.32 PLocal

指定したポイントデータのローカル属性を設定します。

書式

PLocal(ポイントデータ) = ローカル番号

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルを指定します。互換性に関する注意点ポイントデータ引数に変数に指定できなくなりました。変数を使用する場合には、PLocal(P(varName))と記述してください。

ローカル番号

設定するローカル属性の番号を、0～15の整数値または式で指定します。

参照

[PLocal関数](#)

PLocal使用例

```
PLocal(pick) = 1
```

3.19.33 PLocal関数

指定したポイント番号に設定されたローカル番号を返します。

書式

PLocal(ポイントデータ)

パラメーター

ポイントデータ

整数値, P番号, P(式), ポイントラベルを指定します。互換性に関する注意点ポイントデータ引数に変数に指定できなくなりました。変数を使用する場合には、PLocal(P(varName))と記述してください。

戻り値

指定したポイント番号のローカル番号を返します。

参照

[PLocal](#)

PLocal関数使用例

```
Integer localNum  
  
localNum = PLocal(pick)
```

3.19.34 Pls関数

現在位置の各関節パルス値を返します。

書式

Pls(関節番号)

パラメーター

関節番号

現在のパルス値を求める関節を指定します。付加軸のS軸は8、T軸は9になります。

戻り値

関節番号で指定された関節の、現在位置のパルス値を数値で返します。

解説

Plsは、各関節の現在のパルス値を読むために使います。求められた値は保存し、後でPulseコマンドとともに用いることができます。

参照

[CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#), [Pulse](#)

Pls関数使用例

下記は、各関節のパルス値を求めて、その値を出力する簡単な例です。

```
Function plstest
  Real t1, t2, z, u
  t1 = pls(1)
  t2 = pls(2)
  z = pls(3)
  u = pls(4)
  Print "T1 joint current Pulse Value: ", t1
  Print "T2 joint current Pulse Value: ", t2
  Print "Z joint current Pulse Value: ", z
  Print "U joint current Pulse Value: ", u
Fend
```

3.19.35 PNumber関数

ポイントラベルに対応するポイント番号を返します。

書式

PNumber(ポイントラベル)

パラメーター

ポイントラベル

現在のポイントファイルにあるポイントラベル、またはポイントラベルを表わす文字列を指定します。

参照

[PDef関数](#), [PLabel\\$関数](#)

PNumber関数使用例

```
Integer pNum
String pointName$

pNum = PNumber(pick)

pNum = PNumber("pick")

pointName$ = "place"
pNum = PNumber(pointName$)
```


3.19.36 PosFound関数

Find命令時実行の状態を返します。

書式

PosFound

戻り値

アーム移動中に座標位置が見つかった場合は"True"、それ以外は"False"を返します。

参照

[Find](#)

PosFound関数使用例

```
Find Sw(5) = ON
Go P10 Find
If PosFound Then
    Go FindPos
Else
    Print "Error: Cannot find the sensor signal."
EndIf
```

3.19.37 Power

パワーモードをHighまたはLowに設定します。

書式

(1) Power { High | Low } [, Forced]

パラメーター

High | Low

High またはLowを設定します。デフォルト設定は、Lowです。

Forced

省略可能です。通常は省略します。

結果

パラメーターを省略すると、現在のパワーモードを表示します。

解説

パワーモードをHighまたはLowにします。また、現在のパワーモードを表示します。

- Low: パワーモードがLowに設定されていると、ローパワーモードがオンになります。これは、ロボットがゆっくりと (250mm/s以下の速度で)動作することを意味します。また、モーターパワー出力を低く制限します。
- High: パワーモードがHighに設定されていると、ローパワーモードがオフになります。これは、ロボットが Speed, Accel, SpeedS, AccelSで指定した速度、加減速度で動作することを意味します。

下記に、ローパワーモードへの切り替え操作を示します。この場合、速度と加減速度は各ロボットのデフォルト値に制限されます。デフォルト値については各ロボットの仕様表に記載されています。以下のマニュアルも参照してください。

"Epson RC+ ユーザーズガイド - 安全について"

ローパワーモードになる条件

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

デフォルト値に制限される設定

- Speed
- Accel
- SpeedS
- AccelS

注意

- ローパワーモード (Power Low)と最大速度の関係

ローパワーモードではモーター出力が制限され、実際の動作速度は初期値の範囲内になります。ローパワーモード設定時に、コマンドウィンドウまたはプログラム中で、高速にする指示が出されても、初期値速度で動作します。より高速での動作が必要ならば、Power Highに設定しなくてはなりません。

ロボットがハイパワーで動作中に、ローパワーモードに切り替えると、過速度エラーやローパワートルクエラーが発生する可能性があります。

■ ハイパワーモード (Power High)と最大速度の関係

ハイパワーモードでは初期値よりも高速の速度を用いることができます。

■ Forcedフラグ

ロボット動作中 (一時停止中も含む)にパワーモードの変更が可能になります。

ロボットがローパワーで動作中にハイパワーモードに切り替えると、次の動作から設定速度で高速に動作が可能になります。

ロボットがハイパワーで動作中にローパワーモードに切り替えると、過速度エラーやローパワートルクエラーが発生する可能性があります。

ロボットを一時停止してから、Forcedフラグを指定してローパワーモードに切り替えてください。

参照

[Accel](#), [AccelS](#), [Speed](#), [SpeedS](#)

Power使用例

下記にPowerを用いたコマンドウィンドウからの操作例を示します。

```
> Speed 50          'Low Powerモードで高速度を設定

> Accel 100, 100    '高加速度を設定

> Jump P1          '低速度、低加速度で移動

> Speed            '現在の速度を表示
Low Power Mode
  50
  50      50

> Accel            '現在の加速度を表示
Low Power Mode
  100      100
  100      100
  100      100

> Power High       'High Powerモードに設定

> Jump P2          'ロボットを高速度で動作
```

3.19.38 Power関数

パワーモードを返します。

書式

Power [(ロボット番号)]

パラメーター

ロボット番号

状態を確認するロボット番号を整数値で指定します。省略した場合、現在選択されているロボットが対象となります。

戻り値

- 0 = ローパワーモード
- 1 = ハイパワーモード

参照

[Power](#)

Power関数使用例

```
If Power = 0 Then
    Print "Low Power Mode"
EndIf
```

3.19.39 PPIs関数

指定した座標値から関節パルスを計算して返します。

書式

PPIs (座標値, 関節番号)

パラメーター

座標値

ポイントデータを指定します。

関節番号

関節番号 (1～9の整数)を式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

計算した関節パルスをLong型の値で返します。

参照

[Agl関数](#), [CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#), [PAgl関数](#)

PPIs関数使用例

```
Long pulses1
pulses1 = PPIs(P10, 1)
```

3.19.40 Print

Runウィンドウ, オペレーターウィンドウ, コマンドウィンドウ, マクロウィンドウなどのディスプレイヘータを表示します。

書式

Print 表示データ [,表示データ...] [,]

Print

パラメーター

表示データ

数値または文字列式で指定します。省略可能です。

, (カンマ)

ステートメントの終わりにカンマがあると、改行されません。省略可能です。

結果

変数データまたは指定された文字列を返します。

解説

ディスプレイデバイスに、変数データまたは文字列を表示します。

行末にカンマがない限り、自動的に改行されます。

注意

本命令が一度にあつかえる最大データ長は256 Byteです。

- ループ文の中でPrint命令だけを使用しないでください

ループ文の中でPrint命令だけを使用すると、コントローラーがハングアップ状態になる場合があります。

Wait命令、または、動作命令と一緒に使用してください。

悪い例

```
Do
    Print "1234"
Loop
```

良い例

```
Do
    Print "1234"
    Wait 0.1
Loop
```

参照

Print #

Print使用例

下記例では、P100の座標値からU軸座標値を引いて、その値を変数uvarに入れた値を、現在のディスプレイに表示しています。

```
Function test
```

```
  Real uvar
```

```
  uvar = CU(P100)
```

```
  Print "The U Axis Coordinate of " + Chr$(34) + "P100" + Chr$(34) + " is ", uvar
```

```
Fend
```

3.19.41 Print

指定されたファイル、通信ポート、データベースまたはデバイスにデータを出力します。

書式

Print #ポート番号, 出力データ [, 出力データ...][,]

パラメーター

ポート番号

ファイル、通信ポート、データベースまたはデバイスを示すID番号です。ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。通信ポート番号はOpenCom (RS-232C)またはOpenNet (TCP/IP)ステートメントで指定した番号です。データベース番号は、OpenDBステートメントで指定した番号です。

デバイスIDは以下の数値です。

- 21 RC+
- 20 TP4

出力データ...

数値、あるいは文字列を指定します。

, (カンマ)

ステートメントの終わりにカンマがあると、改行されません。省略可能です。

解説

Print # は、ポート番号で指定された通信ポートまたはデバイスに、変数データ、数値、あるいは文字列を出力します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

- 最大データ長
 - 最大データ長

本命令が一度にあつかえる最大データ長は、256 Byteです。ただし、データベースが対象の場合は、4096 Byteが最大データ長になります。通信ポート(TCP/IP)が対象の場合は、1024 Byteが最大データ長になります。

- 他のコントローラーと変数のやりとりをする場合

複数の文字列変数、および、数値変数と文字列変数の両方を指定するときには、文字列データはデリミター(",")を明示的に追加する必要があります。

[コントローラー間で、通信ポートを使って文字列変数、数値変数をやりとりする例]

送信側 (どちらのパターンでもOKです。)

```
Print #PortNum, "$Status,", InData, OutData
Print #PortNum, "$Status", ",", InData, OutData
```

受信側

```
Input #PortNum, Response$, InData, OutData
```


- ファイルへの書き込みはバッファリングされます

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

- Print #命令は、Wait命令、または、動作命令と一緒に使用してください
ループ文の中でPrint #命令だけを使用しないでください

ループ文の中でPrint #命令だけを使用すると、コントローラーがハングアップ状態になる場合があります。

コントローラーの負荷状態によってはWait命令、または、動作命令を使用していても、情報が正しく表示されない場合があります。Wait時間を0.1 (秒)以上設けてください。

悪い例

```
Do
    Print #24, "1234"
Loop
```

良い例

```
Do
    Print #24, "1234"
    Wait 1
Loop
```

参照

Input #, [Print](#), [Write](#), [WriteBin](#)

Print #使用例

Print # を使った簡単な例です。

```
Function printex
String temp$
Print #1, "5"      ' "5"をポート1に出力 temp$ = "hello"
Print #1, temp$
Print #2, temp$
Print #1 " Next message for " + Chr$(34) + "port 1" + Chr$(34)
Print #2 " Next message for " + Chr$(34) + "port 2" + Chr$(34)
Fend
```

3.19.42 ProjectName\$ 関数

現在ロードされているプロジェクトの名前を返します。

書式

ProjectName\$

戻り値

プロジェクトの名前を文字列で返します。

解説

コントローラーで現在ロードされているプロジェクトの名前を返します。

3.19.43 PTCLR

関節のピークトルクをクリアし、初期化します。

書式

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

パラメーター

関節指定1 - 関節指定9

関節番号を整数値または式で指定します。パラメーターを指定しない場合は、全関節のピークトルク値がクリアされます。付加軸のS軸は8、T軸は9になります。存在しない関節番号を指定した場合、エラーが発生します。

解説

PTCLRは指定した関節のピークトルク値をクリアします。

PTRQ実行前に、必ずPTCLRを実行してください。

参照

[ATRQ](#), [PTRQ](#)

PTCLR使用例

[例1] 全関節のピークトルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

```
> ptclr  
> go pl  
> ptrq 1  
    0.227  
> ptrq  
    0.227    0.118  
    0.249    0.083  
    0.000    0.000  
>
```

[例2] 垂直多関節ロボットにおいて、J1, J4, J5のピークトルク値をクリアしたのち、指定した関節番号のトルク値を表示させるコマンド実行例です。

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

3.19.44 PTPBoost

PTP (point to point)動作微小移動時の加減速アルゴリズム調整パラメーターの設定と表示を行います。

書式

(1) PTPBoost 調整設定値 [, Jump退避調整] [, Jump接近調整]

(2) PTPBoost

パラメーター

調整設定値

調整設定値 (0～100の整数)を式または数値で指定します。

Jump退避調整

Jump動作中のZ座標の退避調整設定値 (0～100の整数)を式または数値で指定します。省略可能です。

Jump接近調整

Jump動作中のZ座標の接近調整設定値 (0～100の整数)を式または数値で指定します。省略可能です。

結果

パラメーターを省略すると、現在のPTPBoost設定値を表示します。

解説

PTP動作の微小距離移動時の加減速アルゴリズムを設定します。本設定値は移動距離が微小なときのみ有効です。目標座標への移動距離が微小かどうかはPTPBoostOK関数で確認します。

PTPBoostは通常の使用では設定値を変更する必要はありません。微小移動動作時に少しでもサイクルタイムを短くしたい場合や、逆にサイクルタイムを長くしても残留振動を軽減させたいような場合にのみ使用して下さい。

設定値を大きくするとサイクルタイムは短くなりますが、停止時の振動は発生しやすくなります。また、設定値を小さくするとサイクルタイムは長くなりますが、停止時の振動は発生しにくくなります。停止時の振動が大きい状態でロボットを使用すると、エラーや衝撃を生じ、ロボットの機能が十分に発揮できないばかりでなく、マニピュレーターの寿命を縮める原因となりますのでご注意ください。

下記のいずれかの場合、PTPBoost値は、初期化されます。

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

参照

[PTPBoost関数](#), [PTPBoostOK関数](#)

PTPBoost使用例

```
PTPBoost 50, 30, 30
```

3.19.45 PTPBoost関数

指定されたPTPBoost値を返します。

書式

PTPBoost(パラメーター番号)

パラメーター

設定番号

以下に示す設定値を整数で設定します。

- 1: 調整設定値
- 2: Jump退避調整設定値
- 3: Jump接近調整設定値

戻り値

調整設定値 (0～100の整数値)を返します。

参照

[PTPBoost](#), [PTPBoostOK関数](#)

PTPBoost関数使用例

```
Print PTPBoost(1)
```

3.19.46 PTPBoostOK関数

現在位置から目標座標へのPTP (point to point)動作が微小移動であるかどうかを返します。

書式

PTPBoostOK(目標座標)

パラメーター

目標座標

目標座標をポイントデータで指定します。

戻り値

現在位置から目標座標までのPTP動作微小移動であれば"True"を、そうでない場合は"False"を返します。

解説

現在位置から目標座標へのPTP動作が、PTPBoost命令による加減速アルゴリズム調整が有効な微小移動であるかどうかを確認します。

参照

[PTPBoost](#)

PTPBoostOK関数使用例

```
If PTPBoostOK(P1) Then
  PTPBoost 50
EndIf
Go P1
```

3.19.47 PTPTime関数

PTP動作命令の推定所要時間を返します。PTP動作は行いません。

書式

(1) PTPTime(目標座標, 目標アーム, 目標ツール)

(2) PTPTime(開始座標, 開始アーム, 開始ツール, 目標座標, 目標アーム, 目標ツール)

パラメーター

開始座標

開始位置をポイント式で指定します。

目標座標

目標位置をポイント式で指定します。

目標アーム

目標位置のアーム番号を整数値または式で指定します。

目標ツール

目標位置のツール番号を整数値または式で指定します。

開始アーム

開始位置のアーム番号を整数値または式で指定します。

開始ツール

開始位置のツール番号を整数値または式で指定します。

戻り値

実数値を秒単位で返します。

解説

PTPTime関数を使って、PTP動作命令 (Go)の所要時間の見積りができます。現在位置から目標位置までの所要時間の見積りには書式 (1)を、開始位置から目標位置までの所要時間の見積りには書式 (2)を使用してください。

この関数を実行しても、実際のPTP動作は行われません。現在の位置設定, アーム設定, ツール設定は変更されません。

到達できない位置を設定したり、アームやツールを誤って設定すると、戻り値は0になります。

付加軸を含むロボットの場合、付加軸がサーボ軸で構成される場合は、付加軸の動作時間も加味されます。付加軸がパルス出力軸の場合は、ロボット単独の動作時間を返します。

参照

[ATRQ](#), [Go](#), [PTRQ](#)

PTPTime関数使用例

```
Real secs

secs = PTPTime(P1, 0, 0, P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs

Go P1
secs = PTPTime(P2, 0, 1)
Print "Time to go from P1 to P2 is:", secs
```

3.19.48 PTran

現在位置から指定したパルス量を、1関節のみPTP動作します。

書式

PTran 関節番号, パルス

パラメーター

関節番号

移動する関節番号の整数値を指定します。付加軸のS軸は8、T軸は9になります。

パルス

移動するパルス量を指定します。

解説

Ptranは現在位置から指定したパルス量を、1関節のみ移動します。

参照

[Go](#), [JTran](#), [Jump](#), [Move](#)

Ptran使用例

```
PTran 1, 2000
```


3.19.49 PTRQ

指定した関節のピークトルク値を表示します。

書式

PTRQ [関節番号]

パラメーター

関節番号

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

結果

全関節について現在のピークトルク値を表示します。

解説

PTCLRを実行してから、PTRQを実行し、関節のピークトルク値を表示してください。

ピークトルクは、0～1までの実数値で表わされます。

参照

[ATRQ](#), [PTCLR](#), [PTRQ関数](#)

PTRQ使用例

```
> ptclr
> go pl
> ptrq 1
    0.227
> ptrq
    0.227    0.118
    0.249    0.083
    0.000    0.000
>
```

3.19.50 PTRQ関数

指定した関節のピークトルクを返します。

書式

PTRQ(関節番号)

パラメーター

関節番号

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

戻り値

0～1までの実数値で返します。

参照

[ATRQ](#), [PTCLR](#), [PTRQ](#)

PTRQ関数使用例

プログラム中にPTRQ関数を使用した例です。

```
Function DisplayPeakTorque
  Integer i

  Print "Peak torques:"
  For i = 1 To 4
    Print "Joint ", i, " = ", PTRQ(i)
  Next i
Fend
```

3.19.51 Pulse

各関節のパルス値で指定されたポイントまで、ロボットアームをPTP動作で動かします。

書式

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

(2) Pulse

パラメーター

第1から第4関節パルス値

最初に4つの各関節のパルス値を指定します。Range命令で指定された範囲内の値を、整数またはLong式で指定します。

第5関節パルス値、第6関節パルス値

垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボットで使用します。省略可能です。

第7関節パルス値

ジョイント型7軸ロボットで使用します。省略可能です。

第8関節パルス値、第9関節パルス値

付加軸で使用します。省略可能です。

結果

パラメーター省略時は、現在のロボットの位置を示すパルス値が表示されます。

解説

Pulseでは、ロボットアームの位置を表わすのに、直交座標系の座標ではなく、0パルス位置からの各関節のパルス値を使います。Pulse命令は、ロボットアームをPTP動作で動かします。

Pulse命令で使える上限値と下限値は、Range命令で設定されます。

注意

- Pulse使用前に、軌道上に障害物がないことを確認してください

Jumpと異なり、Pulseは、第3関節の上昇と目標座標までの下降を含め、全関節を同時に動かします。したがってPulse使用時には、ハンドが障害物にあたらないよう、十分注意してください。

起こりやすいエラー

- 制限値を超えるパルス値

Pulse命令で指示されたパルス値が、Rangeで設定された制限値範囲を超えると、エラーになります。

参照

[Go](#), [Accel](#), [Range](#), [Speed](#), [Pls関数](#), [Pulse関数](#)

Pulse使用例

下記はコマンドウィンドウからの操作例です。

各関節のパルス値で指定された位置までロボットアームを動かします。

```
> pulse 16000, 10000, -100, 10
```

現在のロボットアーム位置の、第1から第4関節までのパルス値を表示します。

```
> pulse  
PULSE: 1: 27306 pls 2: 11378 pls 3: -3072 pls 4: 1297 pls  
>
```

3.19.52 Pulse関数

各関節にパルスで指定されたロボット座標値を返します。

書式

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

パラメーター

第1から第4関節パルス値

第1関節から第4関節の各関節のパルス値を指定します。Range命令で指定された範囲内の値を、整数またはLong式で指定します。

第5関節パルス値、第6関節パルス値

垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボットで使用します。省略可能です。

第7関節パルス値

ジョイント型7軸ロボットで使用します。省略可能です。

第8関節パルス値、第9関節パルス値

付加軸で使用します。省略可能です。

戻り値

指定したパルスで使用するロボット座標値を返します。

参照

[Go](#), [JA関数](#), [Jump](#), [Move](#), [Pulse](#), [XY関数](#)

Pulse関数使用例

```
Jump Pulse(1000, 2000, 0, 0)
```

3.20 Q

3.20.1 QP

クイックポーズ機能の設定, 解除と、現在の設定の表示をします。

書式

(1) QP { On | Off }

(2) QP

パラメーター

On | Off

クイックポーズのOn (設定)またはOff (解除)を指定します。

結果

パラメーターを省略すると、現在のQP設定を表示します。

解説

動作コマンド実行時に、Pauseスイッチが押されるか、コントローラーにポーズ入力があった場合、ロボットを直ちに停止させるか、動作コマンドの実行終了を待って停止させるか、クイックポーズ機能で設定します。

直ちに減速, 停止させることを「クイックポーズさせる」といいます。

パラメーター"On"が指定されると、QPはクイックポーズに設定します。パラメーター"Off"が指定されると、QPはクイックポーズを解除します。

QPは、ロボットがポーズ入力に反応して直ちに停止するか、あるいは動作を完了させてから停止するか、現在の設定状態を表示します。QPは、クイックポーズ機能が設定されているか解除されているか、単に状態を表示する命令です。

注意

- クイックポーズ機能は電源オン時にはデフォルトで設定されています

QP命令で設定したクイックポーズ機能設定は、Reset命令実行後も保存されます。しかし、コントローラーまたはドライブユニットを電源オフした後、再度電源オンすると、クイックポーズ機能の設定はデフォルトの"オン(設定)"になります。

- QPと安全扉入力

クイックポーズ機能の設定が解除されていても、安全扉入力が"開"になると、ロボットは直ちに停止します。

参照

[Pause](#)

QP使用例

下記のコマンドウィンドウからの操作例は、ポーズ入力でロボットが直ちに停止するか(クイックポーズ機能が設定されているか)表示させます。

```
> qp
QP ON
```

> qp on 'クイックポーズモードに設定
>

3.20.2 QPDecelR

CP動作時のツール姿勢変化について、クイックポーズ減速度を設定します。

書式

(1) QPDecelR QP減速度設定値

(2) QPDecelR

パラメーター

QP減速度設定値

CP動作におけるクイックポーズ時の減速度を実数値で指定します。(単位: deg/sec²)

結果

パラメーターを省略すると、現在のQPDecelR設定値を表示します。

解説

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

上記動作中にクイックポーズが実行されたとき、関節過加速度エラーが発生することがあります。これは、通常のクイックポーズ動作で自動的に設定されるクイックポーズ減速度が関節許容減速度を超えてしまうために発生します。特に、CP動作のAccelR設定値が大きい場合やロボットの特異姿勢近傍を通過するケースで発生しやすくなります。このようなエラーが発生する場合には、QPDecelRでクイックポーズ減速度を低く設定してください。QPDecelRは小さくしすぎるとクイックポーズに必要な移動量が増加するので極力大きな値を設定してください。通常はQPDecelRを設定する必要はありません。

QPDecelRはAccelRで設定されたCP動作の姿勢変化減速度より小さな減速度は設定できません。その場合は、パラメーター範囲外エラーが発生します。

また、QPDecelR設定後に、設定したQP減速度設定値より大きな減速度がAccelRで設定されたとき、QPDecelRはAccelRで設定された減速度と同じQP減速度を自動的に設定します。

下記のいずれかの場合、QPDecelR値は、デフォルトの最大減速度に初期化されます。

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

参照

[QPDecelR関数](#), [QPDecelS](#), [AccelR](#)

QPDecelR使用例

下記は、Move命令のQPDecelRを設定する簡単な動作プログラム例です。

```
Function QPDecelTest
  AccelR 3000
  QPDecelR 4000
  SpeedR 100
  Move P1 ROT
  .
  .
```


・
Fend

3.20.3 QPDecelR関数

CP動作のツール姿勢変化についてのクイックポーズ減速度の設定値を返す関数です。

書式

QPDecelR

戻り値

CP動作のツール変化についてのクイックポーズ減速度設定値 (実数値、単位: deg/s^2) を返します。

参照

[QPDecelR](#), [QPDecelS関数](#)

QPDecelR関数使用例

```
Real savQPDecelR  
  
savQPDecelR = QPDecelR
```

3.20.4 QPDecelS

CP動作時のクイックポーズ減速度を設定します。

書式

(1) QPDecelS QP減速度設定値 [, Jump3退避QP減速度設定値, Jump3接近QP減速度設定値]

(2) QPDecelS

パラメーター

QP減速度設定値

CP動作におけるクイックポーズ時の減速度を、実数値で指定します。(単位: mm/s²)

Jump3退避QP減速度設定値

Jump3退避動作におけるクイックポーズ時の減速度を、実数値で指定します。(単位: mm/s²)

Jump3接近QP減速度設定値

Jump3接近動作におけるクイックポーズ時の減速度を、実数値で指定します。(単位: mm/s²)

結果

パラメーターを省略すると、現在のQPDecelS設定値を表示します。

解説

CP動作中にクイックポーズが実行されたとき、関節過加速度エラーが発生することがあります。これは、通常のクイックポーズ動作で自動的に設定されるクイックポーズ減速度が、関節許容減速度を超えてしまうために発生します。特に、CP動作のAccelS設定値が大きい場合やロボットの特異姿勢近傍を通過するケースで発生しやすくなります。このようなエラーが発生する場合には、QPDecelSでクイックポーズ減速度を低く設定してください。QPDecelSは小さくしすぎるとクイックポーズに必要な移動量が増加するので極力大きな値を設定してください。通常はQPDecelSを設定する必要はありません。

QPDecelSはAccelSで設定されたCP動作の減速度より小さな減速度は設定できません。その場合は、パラメーター範囲外エラーが発生します。

また、QPDecelS設定後に、設定したQP減速度設定値より大きな減速度がAccelSで設定されたとき、QPDecelSはAccelSで設定された減速度と同じQP減速度を自動的に設定します。

下記のいずれかの場合、QPDecelS値は、デフォルトの最大減速度に初期化されます。

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

参照

[QPDecelS関数](#), [QPDecelR](#), [AccelS](#)

QPDecelS使用例

下記は、Move命令のQPDecelSを設定する簡単な動作プログラム例です。

```
Function QPDecelTest
  AccelS 3000
  QPDecelS 4000
  SpeedS 100
  Move P1
  .
```

・
・
Fend

3.20.5 QPDecelS関数

CP動作のクイックポーズ減速度の設定値を返す関数です。

書式

QPDecelS(設定値番号)

パラメーター

設定値番号

下記のいずれかの値を、整数値または式で指定します。

- 1: CP動作時のクイックポーズ減速度設定値
- 2: Jump3時とJump3CP時の退避動作クイックポーズ減速度設定値
- 3: Jump3時とJump3CP時の接近動作クイックポーズ減速度設定値

戻り値

クイックポーズ減速度設定値 (実数値、単位: mm/s²)を返します。

参照

[QPDecelS](#), [QPDecelR関数](#)

QPDecelS関数使用例

```
Real savQPDecelS  
  
savQPDecelS = QPDecelS(1)
```

3.20.6 Quit

指定されたタスク、またはすべてのタスクの実行を終了します。

書式

Quit { タスク識別子 | All }

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1から32
- バックグラウンドタスク: 65から80
- トラップタスク: 257から267
- All: バックグラウンドタスク以外のすべてのタスクを終了する場合、指定します。

解説

Quit は現在実行中のタスクや、Haltで一時停止されていたタスクを終了します。

Quitは、指定されたタスクがNoPause タスク、NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)、バックグラウンドタスクの場合も、タスクの実行を終了します。また、Quit Allではこれらのタスクを含めてバックグラウンドタスク以外のすべてのタスクの実行を終了します。

Quit Allを実行すると、ロボット制御パラメーターを下記の設定にします。

ロボット制御パラメーター

- SpeedとSpeedR, SpeedS の設定値: (初期値に初期化されます。)
- AccelとAccelR, AccelSの設定値: (初期値に初期化されます。)
- QPDecelR, QPDecelSの設定値: (初期値に初期化されます。)
- LimZパラメーターの設定値: (0に初期化されます。)
- CPパラメーターの設定値: (Offに初期化されます。)
- SoftCPパラメーターの設定値: (Offに初期化されます。)
- Fineの設定: (初期値に初期化されます。)
- Power Low設定: (ローパワーモードになります。)
- PTPBoostの設定値: (初期値に初期化されます。)
- TCLim, TCSpeedの設定値: (初期値に初期化されます。)
- PgLSpeed の設定値: (初期値に初期化されます。)

参照

[Exit](#), [Halt](#), [Resume](#), [Xqt](#)

Quit使用例

下記は、10秒後に2つのタスクを終了する例です。

```
Function main
  Xqt winc1    'タスクwinc1を開始
  Xqt winc2    'タスクwinc2を開始
  Wait 10
```

```
Quit winc1  'タスクwinc1を終了
Quit winc2  'タスクwinc2を終了
Fend

Function winc1
  Do
    On 1; Wait 0.2
    Off 1; Wait 0.2
  Loop
Fend

Function winc2
  Do
    On 2; Wait 0.5
    Off 2; Wait 0.5
  Loop
Fend
```

3.21 R

3.21.1 RadToDeg関数

ラジアンを角度に変換します。

書式

RadToDeg(ラジアン)

パラメーター

ラジアン

角度に変換するラジアンを実数値で指定します。

戻り値

角度を表わすDouble型の値を返します。

参照

[Atan関数](#), [Atan2関数](#), [DegToRad関数](#)

RadToDeg関数使用例

```
s = Cos (RadToDeg (x) )
```


3.21.2 Randomize

乱数系列の初期化を行います。

書式

(1) Randomize シード値

(2) Randomize

パラメーター

シード値

乱数を求めるための基となる値を0以上の実数値で指定します。

参照

[Rnd関数](#)

Randomize使用例

```
Function main
  Real r
  Randomize
  Integer randNum

  randNum = Int(Rnd(10)) + 1
  Print "Random number is:", randNum
Fend
```

3.21.3 Range

各サーボ関節の許容動作エリアを設定, 表示します。

書式

(1) Range 設定値1, 設定値2, 設定値3, 設定値4, 設定値5, 設定値6, 設定値7, 設定値8 [, 設定値9, 設定値10, 設定値11, 設定値12][, 設定値13, 設定値14] [, 設定値15, 設定値16, 設定値17, 設定値18]

(2) Range

パラメーター

設定値1

第1関節の下限パルス値 (単位: パルス)

設定値2

第1関節の上限パルス値 (単位: パルス)

設定値3

第2関節の下限パルス値 (単位: パルス)

設定値4

第2関節の上限パルス値 (単位: パルス)

設定値5

第3関節の下限パルス値 (単位: パルス)

設定値6

第3関節の上限パルス値 (単位: パルス)

設定値7

第4関節の下限パルス値 (単位: パルス)

設定値8

第4関節の上限パルス値 (単位: パルス)

設定値9

第5関節の下限パルス値 (単位: パルス)垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボット専用パラメーターです。

設定値10

第5関節の上限パルス値 (単位: パルス)垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボット専用パラメーターです。

設定値11

第6関節の下限パルス値 (単位: パルス)垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボット専用パラメーターです。

設定値12

第6関節の上限パルス値 (単位: パルス)垂直6軸型ロボット (Nシリーズを含む)、およびジョイント型6軸ロボット専用パラメーターです。

設定値13

第7関節の下限パルス値 (単位: パルス)ジョイント型7軸ロボット専用パラメーターです。

設定値14

第7関節の上限パルス値 (単位: パルス)ジョイント型7軸ロボット専用パラメーターです。

設定値15

第8関節の下限パルス値 (単位: パルス)付加軸S関節専用パラメーターです。

設定値16

第8関節の上限パルス値 (単位: パルス)付加軸S関節専用パラメーターです。

設定値17

第9関節の下限パルス値 (単位:パルス) 付加軸T関節専用パラメーターです。

設定値18

第9関節の上限パルス値 (単位:パルス) 付加軸T関節専用パラメーターです。

結果

パラメーターを省略すると、現在のRange 値を表示します。

解説

Rangeは、各モーター関節の下限と上限パルス値を設定します。単位はパルスです。これによりユーザーは、各関節ごとに最大および最小許容動作範囲を定義することができます。同様に、XY座標方向の制限値は、XYLim命令を使って定義します。

Rangeの初期値はロボットの機種ごとに異なります。この命令で設定された値は、電源オフ後も保存されます。

パラメーター省略時は、現在のRange値が表示されます。

起こりやすいエラー

- 許容範囲外に動かそうとしたとき

1関節でも、ロボットアームの許容動作範囲を超えて動かそうとすると、エラーになります。

- 関節が動かない

下限パルス値が、上限パルス値と同じか大きい場合、関節は動きません。

注意

第6関節の上限パルス値と下限パルス値の設定範囲は、機種によって異なります。

設定範囲については、各マニピュレーターマニュアルの動作エリアを参照してください。

参照

[JRange](#), [SysConfig](#), [XYLim](#)

Range使用例

コマンドウィンドウから現在のRange設定値の表示と変更を行う簡単な例です。

```
> range
-18205, 182045, -82489, 82489, -36864, 0, -46695, 46695
>
> range 0, 32000, 0, 32224, -10000, 0, -40000, 40000
>
```

3.21.4 Read

ファイル、または通信ポートから指定した文字数を読み込みます。

書式

Read #ポート番号, 文字列変数\$, 文字数

パラメーター

ポート番号

ファイル、または通信ポートを示すID番号です。ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C)またはOpenNet (TCP/IP)ステートメントで指定した番号です。

文字列変数\$,

文字列を受信する文字列変数名を指定します。

文字数

読み込むバイト数を指定します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

参照

[ChkCom関数](#), [ChkNet関数](#), [OpenCom](#), [OpenNet](#), [Write](#), [ReadBin](#)

Read使用例

```
Integer numOfChars
String data$

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    Read #1, data$, numOfChars
EndIf
```

3.21.5 ReadBin

ファイル、または通信ポートからバイナリーデータを読みます。

書式

ReadBin #ポート番号, 変数名

ReadBin #ポート番号, 配列変数名(), バイト数

パラメーター

ポート番号

ファイル、または通信ポートを示すID番号です。ファイル番号は、BOpen などのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C)またはOpenNet (TCP/IP)ステートメントで指定した番号です。

変数名

データバイトを受け取る、Byte型変数, 整数変数, またはLong型変数の名前を指定します。

配列変数名()

データバイトを受け取る、Byte型変数, 整数変数, またはLong型変数の名前を指定します。指定可能なのは1次元の配列変数です。

バイト数

読み込むバイト数を指定します。配列の最大要素番号以下で、かつ256Byte以下である必要があります。通信ポート (TCP/IP)が対象の場合は、配列の最大要素番号以下で、かつ1024Byte以下である必要があります。

参照

[Write](#), [WriteBin](#), [Read](#)

ReadBin使用例

```
Integer data
Integer dataArray(10)

numOfChars = ChkCom(1)

If numOfChars > 0 Then
    ReadBin #1, data
EndIf

numOfChars = ChkCom(1)

If numOfChars > 10 Then
    ReadBin #1, dataArray(), 10
EndIf
```

3.21.6 Real

Real型の変数を宣言します。(4バイトの実数値)

書式

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

パラメーター

変数名

Real型に宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。- ローカル変数: 2,000 - バックアップ変数(Global Preserve): 4,000 - グローバル変数とモジュール変数: 100,000

解説

RealはReal型変数の宣言をします。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

Real型の有効桁数は6桁です。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Real使用例

次の例は、Realを使ってReal型の変数を宣言するプログラムです。

```
Function realtest
  Real var1
  Real A(10)          'Real型の1次元配列
  Real B(10, 10)      'Real型の2次元配列
  Real C(5, 5, 5)     'Real型の3次元配列
  Real arrayVar(10)
  Integer i
  Print "Please enter a Real Number:"
  Input var1
  Print "The Real variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter a Real Number:"
    Input arrayVar(i)
    Print "Value Entered was ", arrayVar(i)
  Next i
End
```

3.21.7 RealAccel関数

OLAccel調整後の加減速度の設定値を返す関数です。

書式

RealAccel(設定値番号)

パラメーター

設定値番号

整数値で下記のそれぞれの値を指定します。

- 1: 加速設定値
- 2: 減速設定値
- 3: Jump動作時の退避加速設定値
- 4: Jump動作時の退避減速設定値
- 5: Jump動作時の接近加速設定値
- 6: Jump動作時の接近減速設定値

戻り値

1以上の整数 (%)を返します。

用途

RealAccelを使用することにより、ロボットが連続動作できる最大の加減速度を知ることができます。

手順を下記に記載します。

1. OLAcelコマンドをOnの状態でロボットを動作させます。
2. OLRateコマンドを実行し、過負荷率が上昇するか確認します。
3. 過負荷率が上昇する場合、過負荷率が0.5以上になった時点から加減速度の自動調整が開始されます。
4. 一定時間経過後、OLRateコマンドを実行し過負荷率が上昇しないことを確認してください。
5. 過負荷率が上昇しないことを確認した後、RealAccel関数を実行します。
6. RealAccel関数が返した値が、(1)の動作でロボットが連続動作できる最大の加減速度です。
 - 過負荷率が上昇している最中にRealAccel関数を使用した場合、ロボットが連続動作できる最大の加減速度を知ることができません。
 - オーバーヒートエラーが発生する場合、上記の手順を行ってもロボットが連続動作できる最大の加減速度を知ることができません。

参照

[Accel](#), [OLAccel](#), [OLRate](#)

RealAccel関数使用例

プログラム中にRealAccel関数を使う例です。

```
Integer RealAccel1, RealDecel1

Accel 100, 100
OLAccel on

・現在の加減速度を取得
RealAccel1 = RealAccel (1)
RealDecel1 = RealAccel (2)

現在の加速度を表示
Print RealAccel1
```

現在の減速度を表示

Print RealDecel1

3.21.8 RealPls関数

指定した関節のパルス値を返します。

書式

RealPls(関節番号)

パラメーター

関節番号

現在のパルス値を返す関節を指定します。付加軸のS軸は8、T軸は9になります。

戻り値

関節番号で指定した関節の、現在のエンコーダーパルス値を整数値で返します。

解説

各関節の現在のエンコーダー位置またはパルス値をRealPls関数で返します。この値は保存し、Pulseコマンドで使用できます。

参照

[CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#), [Pulse](#)

RealPls関数使用例

```
Function DisplayPulses
    Long joint1Pulses
    joint1Pulses = RealPls(1)
    Print "Joint 1 Current Pulse Value: ", joint1Pulses
End
```

3.21.9 RealPos関数

指定したロボットの現在の位置を返します。

書式

RealPos

戻り値

指定したロボットの現在位置のポイントを返します。

解説

RealPos関数はロボットの現在位置を返します。

参照

[CurPos関数](#), [CX](#), [CY](#), [CZ](#), [CU](#), [CV](#), [CW](#), [CR](#), [CS](#), [CT](#), [RealPls関数](#)

RealPos関数使用例

```
Function ShowRealPos
    Print RealPos
Fend

P1 = RealPos
```

3.21.10 RealTorque関数

指定した関節の現在のトルク指令値を返します。

書式

RealTorque(関節番号)

パラメーター

関節番号

トルク指令値を取得したい関節の関節番号を式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

現在のパワーモードにおける最大トルクに対する割合を-1～1の実数値で返します。

正値は関節角の正方向を表し、負値は関節角の負方向を表します。

参照

[TC](#), [TCSpeed](#), [TCLim](#)

RealTorque関数使用例

```
Print "現在のZ軸トルク指令値(スカラロボット):", RealTorque(3)
```

3.21.11 Recover

安全扉が開いた時点の位置へ復帰動作を実行し、状態を返します。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

(1) Recover ロボット番号 | All

(2) Recover ロボット番号 | All , WithMove | WithoutMove

パラメーター

ロボット番号

復帰動作を行うロボット番号を指定します。

All

すべてのロボットで復帰動作を実行します。省略時はAllとなります。

WithMove

値0の定数です。

励磁の復帰と安全扉が開いた時点の位置への移動を行います。省略時はWithMoveとなります。

WithoutMove

値1の定数です。

励磁の復帰のみを行います。通常は使用しません。AbortMotionと組み合わせて特殊な復帰を実現します。

解説

本命令をプログラムから実行するには、Epson RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスをチェックする必要があります。

Recoverは、安全扉が閉じられてから、モーターを再度オンし、ロボットを安全扉が開いた時点の位置までローパワーのPTP動作で戻ります。復帰動作が完了すると、ContまたはContManualRecoverを用いてサイクルを継続させることができます。

コントローラーに複数のロボットが設定されており、Allが指定された場合は、ロボット番号が小さい順に復帰動作を実行します。

参照

[AbortMotion](#), [Cont](#), [Recover関数](#), [RecoverPos関数](#), [ContManualRecover](#)

Recover使用例

注意

プログラムからRecoverコマンドを実行する場合、コマンドの仕様を理解して、システムとして復帰動作可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

```
Function main
  Xqt 2, monitor, NoPause
  Do
    Jump P1
    Jump P2
  Loop
Fend
```

```
Function monitor
  Do
    If Sw(SGOpenSwitch) = On then
      Wait Sw(SGOpenSwitch) = Off and Sw(RecoverSwitch) = On
      Recover All
    EndIf
  Loop
Fend
```

3.21.12 Recover関数

安全扉が開いた時点の位置へ復帰動作を実行し、状態を返します。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

(1) Recover

(2) Recover(ロボット番号 | All)

(3) Recover(ロボット番号 | All , WithMove | WithoutMove)

パラメーター

ロボット番号

復帰動作を行うロボット番号を指定します。ロボット番号を省略した場合は、すべてのロボットで復帰動作を実行します。

All

すべてのロボットで復帰動作を実行します。省略時はAllとなります。

WithMove

値0の定数です。

励磁の復帰と安全扉が開いた時点の位置への移動を行います。省略時はWithMoveとなります。

WithoutMove

値1の定数です。

励磁の復帰のみを行います。通常は使用しません。AbortMotionと組み合わせて特殊な復帰を実現します。

戻り値

Boolean型の値を返します。復帰動作が完了すると、"True"を返します。完了していなければ、"False"を返します。

解説

本命令をプログラムから実行するには、Epson RC+ の[セットアップ] - [システム設定] - [コントローラー] - [環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックする必要があります。

Recoverは、安全扉が閉じられてから、モーターを再度オンし、ロボットを安全扉が開いた時点の位置までローパワーのPTP動作で戻ります。復帰動作が完了すると、ContまたはContManualRecoverを用いてサイクルを継続させることができます。復帰動作が問題なく完了すると、"True"が返されますが、復帰動作中に、一時停止、中断、または安全扉が開かれると、Recoverは、"False"を返します。

コントローラーに複数のロボットが設定されており、Allが指定された場合は、ロボット番号が小さい順に復帰動作を実行します。

注意

プログラムからRecoverコマンドを実行する場合、コマンドの仕様を理解して、システムとして復帰動作可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

[AbortMotion](#), [Cont](#), [Recover](#), [RecoverPos関数](#), [ContManualRecover](#)

Recover関数使用例

```
Boolean sts
Integer answer

sts = Recover
If sts = True Then
MsgBox "Ready to continue", MB_ICONQUESTION + MB_YESNO, "MyProject", answer
If answer = IDYES Then
    Cont
EndIf
EndIf
```

3.21.13 RecoverPos関数

安全扉が開いた時点の位置を返します。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

RecoverPos ([ロボット番号])

パラメーター

ロボット番号

ロボットの番号を整数値で指定します。ロボット番号を省略した場合、現在選択されているロボットが対象となります。

結果

安全扉が開いた時点の位置を返します。

安全扉が開いていない場合や、ロボットが復帰動作を終了している場合は、XからWの値に0が入ったポイントデータを返します。

解説

ContまたはRecoverで復帰動作を行う場合のロボットの復帰位置を返します。

参照

[AbortMotion](#), [Cont](#), [Recover](#), [Recover関数](#), [RealPos関数](#)

RecoverPos関数使用例

復帰動作の直線距離が10mm未満の場合は復帰動作を行い、それ以上場合はプログラムを終了します。

```
If Dist (RecoverPos, RealPos) < 10 Then
Recover All
Else
Quit All
EndIf
```


3.21.14 Redim

ランタイム時に配列の最大要素番号を変更します。

書式

Redim [Preserve] 配列名(配列変数の最大要素番号)

パラメーター

Preserve

配列の前の値を保存します。省略可能で、省略すると配列はクリアされます。

配列名

配列変数の名前を指定します。通常の変数名の慣例にしたがって指定します。あらかじめ配列が宣言されていなければなりません。

配列変数の最大要素番号

配列変数の新しい最大要素番号を指定します。変数宣言時と同じ数の最大要素番号を与えてください。次の書式を用います。

(最大要素番号1, [最大要素番号2], [最大要素番号3])

要素番号は0から始まるので、要素数は最大要素番号に1加えた数となります。

各最大要素番号は、全要素数が以下の最大値を超えない範囲で指定します。

	String型以外	String型
ローカル変数	2,000	200
バックアップ変数 (Global Preserve)	4,000	400
グローバル変数とモジュール変数	100,000	10,000

解説

Redimは、ランタイム時に配列変数の最大要素番号を変更するときに使います。前の値を保存するときは、Preserve を指定します。

Byref指定された配列変数はRedimすることはできません。

頻繁なRedimの実行は、プログラムの実行速度を低下させます。特にバックアップ変数に対するRedimの実行は必要最小限にすることを推奨します。

参照

[UBound関数](#)

Redim使用例

```
Integer i, numParts, a(0)

Print "Enter number of parts "
Input numParts

Redim a(numParts)

For i = 0 to UBound(a)
    a(i) = i
Next

'最大要素番号を20追加します
```

```
Redim Preserve a(numParts + 20)
```

```
'最初の要素番号を保持
```

```
For i = 0 to UBound(a)
```

```
    Print a(i)
```

```
Next
```

3.21.15 Rename

ファイル名を変更します。

書式

Rename 変更元ファイル名, 変更先ファイル名

パラメーター

変更元ファイル名

名前を変更するファイル名とパスの文字列式を指定します。詳細は、ChDiskを参照してください。

変更先ファイル名

変更元ファイル名で指定したファイルに、新たにつける名前を指定します。詳細は、ChDiskを参照してください。

解説

変更元ファイル名で指定したファイルの名前を、変更先ファイル名で指定する名前に変更します。パスを省略すると、このステートメントは、カレントディレクトリーから変更元ファイル名を探します。

変更元ファイル名と変更先ファイル名が、同一ドライブ内で指定されている場合のみ実行可能です。

同じパスに存在する他のファイルと同じ名前に変更することはできません。

ワイルドカードとして使われる文字は、変更元ファイル名、変更先ファイル名のパラメーターには使えません。

参照

[Copy](#)

Rename使用例

コマンドウィンドウでの実行例

```
> Rename A.PRG B.PRG
```

3.21.16 RenDir

ディレクトリー名を変更します。

書式

Rendir 変更元ディレクトリー名, 変更先ディレクトリー名

パラメーター

変更元ディレクトリー名

変更するディレクトリーのパスとディレクトリー名を文字列式で指定します。

変更先ディレクトリー名

変更後のディレクトリーのパスとディレクトリー名を文字列式で指定します。パスの詳細は、ChDiskを参照してください。

解説

変更先ディレクトリーのパスは、変更元ディレクトリーのパスに存在しなければなりません。

変更元ディレクトリー名、変更先ディレクトリー名のパスを省略し、ディレクトリー名だけを指定した場合、カレントディレクトリーにあるディレクトリーを指します。

ワイルドカードとして使われる文字は、変更元ディレクトリー名、変更先ディレクトリー名のパラメーターには使えません。

注意

ディスクがPCの場合に実行可能です。

参照

[MkDir](#)

RenDir使用例

```
RenDir "c:\mydata", "c:\mydata1"
```

3.21.17 Reset

コントローラーを初期状態にリセットします。

書式

(1) Reset

(2) Reset Error

解説

- Reset は、下記をリセットします。
- Reset Errorは、すべての通常タスクを終了して、エラー状態とロボット制御パラメーターのみリセットします。
- Reset Error命令をプログラムから実行するには、Epson RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。
 - 非常停止状態 (Reset 時のみ)
 - エラー状態
 - 出力ビット (Reset 時のみ): リモート出力に割り当てられているI/Oと、ハンドに割りあてられているI/Oを除く、すべての出力ビットはオフになります。この機能はEpson RC+から解除することができます。
 - ロボット制御パラメーター
 - SpeedとSpeedR, SpeedS の設定値: (初期値に初期化されます。)
 - AccelとAccelR, AccelSの設定値: (初期値に初期化されます。)
 - QPDecelR, QPDecelSの設定値: (初期値に初期化されます。)
 - LimZパラメーターの設定値: (0に初期化されます。)
 - CPパラメーターの設定値: (Offに初期化されます。)
 - SoftCPパラメーターの設定値: (Offに初期化されます。)
 - Fineの設定: (初期値に初期化されます。)
 - Power Low設定: (ローパワーモードになります。)
 - PTPBoostの設定値: (初期値に初期化されます。)
 - TCLim, TCSpeedの設定値: (初期値に初期化されます。)
 - PgLSpeed の設定値: (初期値に初期化されます。)

サーボ関連のエラーや非常停止状態、その他のリセットが必要となる状態では、Reset以外のコマンドは受けつけられません。この場合、まずResetを実行してから、その他の必要な処理を行ってください。

例えば、非常停止の後では、まず周囲と操作の安全を確認し、Reset を実行します。その後、Motor Onを実行してください。

重大エラー状態は、Resetでは解除されません。

重大エラーが発生した場合、コントローラーの電源をオフし、エラーの原因を取りのぞいてください。

Reset命令は、バックグラウンドタスクやTrap Emergency, Trap Errorで起動されるタスクなどから実行することはできません。非常停止状態は、プログラムから解除することはできません。

注意

- [Resetで出力ポートをOff]チェックボックス

Epson RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[Resetで出力ポートをOff]チェックボックスがチェックされている場合は、Reset命令が発行されると、ハンドに設定されたビットを除く、すべての出力はオフになります。この設定により、出力オフになったとき、ワークを落下させたり、それと似たような状況が生じないように、配線に配慮することが重要です。

ハンドについては、以下のマニュアルを参照してください。

"Hand機能"

参照

[Accel](#), [AccelS](#), [Fine](#), [LimZ](#), [Motor](#), [Off](#), [On](#), [PTPBoost](#), [SFree](#), [SLock](#), [Speed](#), [SpeedS](#)

Reset使用例

Reset命令をコマンドウィンドウから実行する例です。

```
>reset  
>
```

3.21.18 ResetElapsedTime

ElapsedTime関数で使用するタクトタイム計測用タイマーをリセットします。

書式

ResetElapsedTime

解説

タクトタイム計測用タイマーをリセットし、スタートします。

参照

[ElapsedTime関数](#)

ResetElapsedTime使用例

```
ResetElapsedTime      'タクトタイム計測用タイマーをリセット
For i = 1 To 10        '10回実行
    GoSub Cycle
Next
Print ElapsedTime / 10 'タクトタイムを計算して表示
```

3.21.19 Restart

現在のメインプログラムを再実行します。

本命令は上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

Restart

解説

Restartは実行中のタスクをすべて中断し、最後に実行していたメインプログラムを再実行します。

バックグラウンドタスクは中断せず、そのまま継続実行します。

すべてのTrap設定は解除されるため、本命令でタスクを中断しても Trap Abortは実行されません。

Pause状態は本命令を実行することで解除されます。

エラー状態で本命令を実行するとエラーになります。Reset Error命令などで、まずはエラーを解除してください。

非常停止状態で本命令を実行するとエラーになります。非常停止状態はプログラムから解除することはできません。

注意

プログラムからRestartコマンドを実行する場合、コマンドの仕様を理解して、システムとして再実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

注意

リモートI/Oコントロールを使用している場合、SPEL+プログラムのRestart命令とリモート入力のStart信号を同じタイミングで実行しないでください。プログラムが2重で実行され、2503エラーが発生する可能性があります。

参照

[Quit](#), [Reset](#), [Trap \(ユーザー定義トリガー\)](#), [Xqt](#)

Restart使用例

```
Function main
    Trap Error Xqt eTrap
    Motor On
    Call PickPlac
Fend

Function eTrap
Wait Sw(ERresetSwitch)
Reset Error
Wait Sw(RestartSwitch)
Restart
Fend
```


3.21.20 Resume

Halt命令により一時停止中のタスクを継続実行します。

書式

Resume { タスク識別子 | All }

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

All

すべてのタスクを継続実行するとき指定します。

解説

Resume は、Halt命令により一時停止中のタスクを継続実行します。

参照

[Halt](#), [Quit](#), [Xqt](#)

Resume使用例

Halt命令の後に、Resume命令を使用した例です。

```
Function main
  Xqt 2, flicker      'flickerをタスク2で実行

  Do
    Wait 3            'flickerを3秒間実行
    Halt flicker      'flickerタスクを一時停止
    Wait 3
    Resume flicker    'flickerタスクを継続実行
  Loop
Fend

Function flicker
  Do
    On 1
    Wait 0.2
    Off 1
    Wait 0.2
  Loop
Fend
```

3.21.21 Return

Returnステートメントは、GoSubステートメントと組み合わせて使います。GoSubはプログラム制御をサブルーチンに移します。サブルーチンが完了すると、Returnで、サブルーチンを始めたGoSub命令の次の行にプログラム実行が継続するようにします。

書式

Return

解説

ReturnステートメントはGoSubステートメントと組み合わせて使います。Returnステートメントの主要な目的は、サブルーチンに制御を移したGoSub命令の後に続く命令に、プログラム制御を戻すことです。

GoSub命令は、プログラム制御をユーザー指定のステートメント行、またはラベルに枝分かれさせます。プログラムは、その移行先の行とそれに続く行を、Return命令まで実行していきます。Return命令は、サブルーチンへの移行を指示したGoSubの次の行に、プログラム制御を戻します。(つまりGoSub命令はサブルーチンの実行を開始し、ReturnでそのGoSubの次のステートメントに戻ります。)

起こりやすいエラー

- GoSubがないのにReturnを使用した場合

Return命令は、サブルーチンからGoSubを発行した元のプログラムに戻るために使われます。GoSubがないのにReturn命令が使われるとエラー2383が発生します。システムはどこに戻ってよいのか分からないので、単独のReturn命令は意味がありません。

参照

[OnErr](#), [GoSub...Return](#), [GoTo](#)

Return使用例

下記は、GoSub命令で、checkioというラベルに枝分かれして、最初の16個のユーザー入力をチェックする簡単な例です。その後、そのサブルーチンからメインプログラムに戻ります。

```
Function main
  Integer var1, var2
  GoSub checkio
  On 1
  On 2
  Exit Function

checkio:      ' サブルーチンの開始位置
  var1 = In(0)
  var2 = In(1)
  If var1 <> 0 Or var2 <> 0 Then
    Print "Message to Operator here"
  EndIf
finished:
  Return      ' サブルーチンの終了位置
Fend
```

3.21.22 Right\$関数

文字列の最後から、指定数の文字列を取り出す関数です。

書式

Right\$(文字列, 文字数)

パラメーター

文字列

指定数の文字列を最後からを取り出します。最長255文字の文字列式または直接文字列を指定します。

文字数

文字列の最後からコピーしてくる文字数 (正の整数) を、式または直接数値で指定します。

戻り値

指定した文字列の最後から、指定数の文字列を取り出して返します。

解説

Right\$は指定した文字列の最後から指定数の文字列を返します。Right\$は、直接文字列にあるだけの文字数を返すことができます。

参照

[Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Left\\$関数](#), [Len関数](#), [Mid\\$関数](#), [Space\\$関数](#), [Str\\$関数](#), [Val関数](#)

Right\$使用例

下記は、ワークのデータ文字列を入力すると、ワークのパーツ番号, 名前, パーツカウントなどの情報を返すプログラムの例です。

```
Function SplitPartData(DataIn$ As String, ByRef PartNum$ As String, ByRef PartName$ As String, ByRef PartCount As Integer)

    PartNum$ = Left$(DataIn$, 10)

    DataIn$ = Right$(DataIn$, Len(DataIn$) - pos)
    pos = Instr(DataIn$, ",")

    PartName$ = Mid$(DataIn$, 11, 10)

    PartCount = Val(Right$(DataIn$, 5))

End
```

また、下記はコマンドウィンドウからの操作例です。

```
> Print Right$("ABCDEFGH", 2)
FG

> Print Right$("ABC", 3)
ABC
```

3.21.23 Rmdir

サブディレクトリーを削除します。

書式

Rmdir ディレクトリー名

パラメーター

ディレクトリー名

削除するディレクトリーのパスと名前を文字列式で指定します。パスを省略しディレクトリー名だけを指定した場合、カレントディレクトリーにあるサブディレクトリーを指します。パスの詳細は、ChDiskを参照してください。

解説

指定したサブディレクトリーを削除します。このステートメントを実行する前に、サブディレクトリー内のファイルを削除しておかなければなりません。

カレントディレクトリーや親ディレクトリーは削除することができません。

コマンドウィンドウから実行すると、引用符は省略することができます。

注意

ディスクがPCの場合に実行可能です。

Rmdir使用例

コマンドウィンドウでの実行例

```
> Rmdir \mydata
```

3.21.24 Rnd関数

乱数を返します。

書式

Rnd(最大値)

パラメーター

最大値

最大の戻り値を実数値で設定します。

戻り値

0から、パラメーターで指定した最大値までの、実数値の乱数を返します。

解説

乱数を生成するために使います。

参照

[Int関数](#), [Randomize](#)

Rnd関数使用例

1～10までの乱数を生成する例です。

```
Function main
  Real r
  Integer randNum

  Randomize
  randNum = Int(Rnd(9)) + 1
  Print "Random number is:", randNum
Fend
```

3.21.25 Robot

ロボットを選択します。

書式

Robot ロボット番号

パラメーター

ロボット番号

ロボット番号を指定します。範囲は1から、設置されているロボット数です。

解説

Robotステートメントは、次に動作命令を実行するロボットを選択します。

ロボットが1台の場合、Robotステートメントを使用する必要はありません。

参照

[Accel](#), [AccelS](#), [Arm](#), [ArmSet](#), [Go](#), [Hofs](#), [Home](#), [Hordr](#), [Local](#), [Move](#), [Pulse](#), [Robot関数](#), [Speed](#), [SpeedS](#)

Robot使用例

```
Function main
  Integer I
  For I = 1 to 100
    Robot 1
    Go P(i)
    Robot 2
    Go P(i)
  Next I
Fend
```

3.21.26 Robot関数

現在のロボット番号を返します。

書式

Robot

戻り値

現在のロボット番号の整数値を返します。

参照

[Robot](#)

Robot関数使用例

```
Print "The current robot is: ", Robot
```

3.21.27 RobotInfo関数

ロボットの状態情報を返します。

書式

RobotInfo(インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報の整数値を返します。

解説

戻り値のビット情報は次の表のとおりです。

インデックス	ビット	値	解説
0	0	&H1	未定義
	1	&H2	リセットできるエラー発生
	2	&H4	リセットできないエラー発生
	3	&H8	モーターON
	4	&H10	パワーHigh
	5	&H20	未定義
	6	&H40	未定義
	7	&H80	未定義
	8	&H100	ロボットはHalt状態
	9	&H200	ロボットはHalt状態ではない (動作中またはクイックポーズ中)
	10	&H400	一時停止または安全扉で、ロボット停止
	11		未定義
	12		未定義
	13		未定義
	14	&H4000	動作コマンドの後、TILL条件を満たした
	15	&H8000	動作コマンドの後、SENSE条件を満たした
	16-31		未定義

インデックス	ビット	値	解説
1	0	&H1	追従動作中 (コンベヤトラッキング中)
	1	&H2	復帰動作待ち (WaitRecover状態)
	2	&H4	復帰動作実行中
	3-31		未定義
2	0	&H1	ロボットはホーム位置
	1-31		未定義
3	0	&H1	第1関節サーボ励磁中
	1	&H2	第2関節サーボ励磁中
	2	&H4	第3関節サーボ励磁中
	3	&H8	第4関節サーボ励磁中
	4	&H10	第5関節サーボ励磁中
	5	&H20	第6関節サーボ励磁中
	6	&H40	第7関節サーボ励磁中
	7	&H80	S関節サーボ励磁中
	8	&H100	T関節サーボ励磁中
	9-31		未定義
4	N/A	0 - 32 -1	ロボットコマンドを実行しているタスク番号 0 = コマンドウィンドウまたはマクロからコマンドを実行 -1 = タスクはマニピュレーターを未使用
5	0	&H1	第1関節ブレーキON
	1	&H2	第2関節ブレーキON
	2	&H4	第3関節ブレーキON
	3	&H8	第4関節ブレーキON
	4	&H10	第5関節ブレーキON
	5	&H20	第6関節ブレーキON
	6	&H40	第7関節ブレーキON
	7	&H80	S関節ブレーキON
	8	&H100	T関節ブレーキON

インデックス	ビット	値	解説
	9-31		未定義

参照

[CtrlInfo関数](#), [RobotInfo\\$関数](#), [TaskInfo関数](#)

RobotInfo関数使用例

```
If (RobotInfo(3) And &H1) = &H1 Then
  Print "Joint 1 is locked"
Else
  Print "Joint 1 is free"
EndIf
```

3.21.28 RobotInfo\$関数

ロボットのテキスト情報を返します。

書式

RobotInfo\$(インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報の文字列を返します。

解説

- 0: ロボット名
- 1: 型名
- 2: デフォルトポイントファイル名
- 3: 未定義
- 4: ロボットのシリアル番号

参照

[CtrlInfo関数](#), [RobotInfo関数](#), [TaskInfo関数](#)

RobotInfo\$関数使用例

```
Print "Robot Name: ", RobotInfo$(0)
```

3.21.29 RobotModel\$関数

ロボットの型名を返します。

書式

RobotModel\$

戻り値

型名の文字列を返します。これはロボットの後方パネルに記載されている、ロボットの名前です。

参照

[RobotType関数](#)

RobotModel\$関数使用例

```
Print "The robot model is ", RobotModel$
```

3.21.30 RobotName\$関数

ロボット名を返します。

書式

RobotName\$

戻り値

ロボット名を文字列で返します。

参照

[RobotInfo関数](#), [RobotModel\\$関数](#)

RobotName\$関数使用例

```
Print "The robot name is ", RobotName$
```

3.21.31 RobotSerial\$関数

ロボットのシリアル番号を返します。

書式

RobotSerial\$

戻り値

ロボットのシリアル番号を文字列で返します。

参照

[RobotInfo関数](#), [RobotName\\$関数](#), [RobotModel\\$関数](#)

RobotSerial\$関数使用例

```
Print "The robot serial number is ", RobotSerial$
```

3.21.32 RobotType関数

ロボットタイプを返します。

書式

RobotType

戻り値

- 1: ジョイント型
- 2: 直角座標型
- 3: 水平多関節型
- 5: 垂直6軸型
- 6: RSシリーズ
- 7: Nシリーズ

参照

[RobotModel\\$関数](#)

RobotType関数使用例

```
If RobotType = 3 Then
  Print "Robot type is SCARA"
EndIf
```

3.21.33 ROpen

ファイルを読み取り専用モードで開きます。

書式

ROpen ファイル名As #ファイル番号..Close #ファイル番号

パラメーター

ファイル名

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

ファイル番号

30～63の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは指定したファイルからデータを読み出すときに使います。

注意

- PCディスクのみ
- ネットワークパスが使用できます。

指定したファイル番号は、ファイルを開いている間、そのファイルを識別するために使います。したがって、そのファイルを閉じるまでは、他のファイルに同じファイル番号を使うことはできません。ファイル番号は、ファイル操作命令 (Input#, Read, Seek, Eof, Close)で使います。

Closeステートメントでファイルを閉じし、ファイル番号を解放します。

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

参照

Close, Input #, AOpen, BOpen, UOpen, WOpen, FreeFile関数

ROpen使用例

```
Integer fileNum, i, j

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

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


3.21.34 ROTOK関数

目標座標への動作命令時に、ROT修飾パラメーターが付加可能かどうかを返します

書式

ROK (目標座標)

ROK (基準座標, 目標座標)

パラメーター

目標座標

ROT修飾パラメーターが付加可能かどうかを、調べる目標座標をポイントデータで指定します。

基準座標

ROT修飾パラメーターが付加可能かどうかを、調べる基準座標をポイントデータで指定します。

戻り値

ROT修飾パラメーターを付加可能であれば"True"を、そうでない場合は"False"を返します。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

解説

ROT修飾パラメーターが付加可能かを、実際に(ロボットを)動かす前に確認します。

ROT修飾パラメーターとは、Moveなどの直線補間動作命令に付加可能で、ツール姿勢変化の加減速を優先することを指定するパラメーターです。

ROK関数では、基準座標から目標座標にMoveなど、直線補間動作命令で移動する場合に、ROTを付加するとエラーになるかどうかを、動作前に判定できます。

基準座標を省略した場合は、現在位置(Here)を基準として判定した結果を返します。

姿勢変化角度が"0"、または微小変化の場合、ROT修飾パラメーターを付加すると発生するエラーを判定します。ただし、4242エラーなど、動作中にロボットの関節速度や関節加速度が、マニピュレーターの限度越えるかどうかの判定はできません。その場合は、SpeedS, SpeedR, AccelS, AccelRの各値を下げるなどの調整をしてください。

参照

[Move](#)

ROK関数使用例

```
If ROTOK(P1) = True Then
  Move P1 ROT
Else
  Move P1
EndIf
```

3.21.35 RSet\$関数

文字列の先頭に半角スペースを追加して、指定の長さにした文字列を返します。

書式

RSet\$(文字列, 文字列の長さ)

パラメーター

文字列

文字列式を指定します。

文字列の長さ

返される文字列の長さを示す、整数または式を指定します。

戻り値

先頭に半角スペースを追加した指定文字列を返します。

参照

[LSet\\$関数](#), [Space\\$関数](#)

RSet\$関数使用例

```
temp$ = "123"  
temp$ = RSet$(temp$, 10)   ' temp$ = "      123"
```

3.21.36 RShift関数

数値データの右論理シフト

書式

RShift(数値データ, シフトビット数)

パラメーター

数値データ

論理シフトする数値を、式または直接数値で指定します。

シフトビット数

右論理シフトするビット数値 (0～31の整数値)を指定します。

戻り値

指定した数値データを、右論理シフトした値を返します。

解説

RShiftは、指定した数値データを右 (下位方向)に指定ビット数シフトします。シフトした上位のビットは、常に0がセットされます。

最も簡単な説明としては、Rshiftは数値データを2のシフトビット乗で割った数を返します。

注意

■ 数値データ型

数値データは、有効な数値データ型ならば、どんなデータ型もとります。

RShiftは、次のデータ型に対応しています。

Byte型, Double型, Int32型, Integer型, Long型, Real型, Short型, UByte型, UInt32型, UShort型

参照

[And 演算子](#), [LShift関数](#), [LShift64関数](#), [Not 演算子](#), [Or 演算子](#), [RShift64関数](#), [Xor 演算子](#)

RShift使用例

下記は、"0"から始まるInteger型数値データに対する、すべてのRshift値を示すプログラムの例です。

```
Function rshiftst
  Integer num, snum, i
  num = 32767
  For i = 1 to 16
    Print "i =", i
    snum = RShift(num, i)
    Print "RShift(32767, ", i, ") = ", snum
  Next i
Fend
```

コマンドウィンドウでRshift命令を操作した例です。

```
> Print RShift(10,1)
5
> Print RShift(8,3)
1
> Print RShift(16,2)
4
```

3.21.37 RShift64関数

数値データの右論理シフト

書式

RShift64(数値データ, シフトビット数)

パラメーター

数値データ

論理シフトする数値を、式または直接数値で指定します。

シフトビット数

右論理シフトするビット数値 (0～63の整数値)を指定します。

戻り値

指定した数値データを、右論理シフトした値を返します。

解説

RShift64は、指定した数値データを右 (下位方向)に指定ビット数シフトします。シフトした上位のビットは、常に0がセットされます。

最も簡単な説明としては、Rshift64は数値データを2のシフトビット乗で割った数を返します。

注意

- 数値データ型

数値型は数種類あります。RShift64は、Int64型, UInt64型の数値に使うことができます。

参照

[And 演算子](#), [LShift関数](#), [LShift64関数](#), [Not 演算子](#), [Or 演算子](#), [RShift関数](#), [Xor 演算子](#)

RShift 64使用例

下記は、"0"から始まるUInt64型数値データに対する、すべてのRshift64値を示すプログラムの例です。

```
Function rshif64tst
  UInt64 num, snum, i
  num = 18446744073709551615
  For i = 1 to 63
    Print "i =", i
    snum = RShift64(num, i)
    Print "RShift64(18446744073709551615, ", i, ") = ", snum
  Next i
Fend
```

コマンドウィンドウでRshift64命令を操作した例です。

```
> Print RShift64(10,1)
5
> Print RShift64(8,3)
1
> Print RShift64(16,2)
4
```

3.21.38 RTrim\$関数

文字列の右側半角スペースを削除して返します。

書式

RTrim\$(文字列)

パラメーター

文字列

文字列式または直接文字列で指定します。

戻り値

右側の半角スペースを削除した文字列

参照

[LTrim\\$関数](#), [Trim\\$関数](#)

RTrim\$関数使用例

```
str$ = " data "  
str$ = RTrim$(str$) ' str$ = " data"
```

3.21.39 RunDialog

SPEL+ のプログラムからEpson RC+画面を起動します。

書式

- (1) RunDialog ダイアログID
- (2) RunDialog DLG_ROBOTMNG [, ロボット選択ビット]

パラメーター

ダイアログID

- 有効なダイアログIDを含んだ整数値を指定します。これらの値は、あらかじめ下記の定数で定義されています。
- DLG_ROBOTMNG: 100: ロボットマネージャダイアログを起動します
 - DLG_IOMON: 102: I/Oモニターを起動します
 - DLG_VGUIDE: 110: Vision Guideダイアログを起動します

ロボット選択ビット

ダイアログIDにDLG_ROBOTMNGを指定したときのみ有効です。ロボットマネージャで選択可能なロボットをビット値で指定します。

設定例	設定値	bit15	bit14	...	bit2	bit1	bit0
ロボット1	&H0001	Off	Off		Off	Off	On
ロボット2	&H0002	Off	Off		Off	On	Off
ロボット1と2	&H0003	Off	Off		Off	On	On
:							
ロボット16	&H1000	On	Off		Off	Off	Off

解説

SPEL+ のタスクからEpson RC+の画面を起動/表示するときに、RunDialog を使います。タスクは、画面が閉じられるまで一時停止します。

Epson RC+の画面からロボットコマンドを実行するときは、他に、その画面表示中にロボットを制御するタスクがないことを確認してください。もし他にロボットを制御するタスクが起動している場合は、エラーになります。

参照

[InputBox](#), [MsgBox](#)

RunDialog使用例

```
If Motor = Off Then
  RunDialog DLG_ROBOTMNG
  If Motor = Off Then
    Print "Motors are off, aborting program"
    Quit All
  EndIf
EndIf
```

3.22 S

3.22.1 SafetyOn関数

安全扉の状態を返します。

書式

SafetyOn

戻り値

安全扉開状態のときは"True"、それ以外のときは"False"を返します。

解説

本関数は、NoPauseタスク、NoEmgAbortタスク (Xqt時に、NoPauseやNoEmgAbortを指定して開始された特別なタスク)とバックグラウンドタスクでのみ使用します。

参照

[ErrorOn関数](#), [EStopOn関数](#), [PauseOn関数](#), [Wait](#), [Xqt](#)

SafetyOn関数使用例

下記例は、コントローラーが安全扉開になるのを監視し、安全扉開が発生した場合I/Oをオンオフするプログラムです。

注意

- Forcedフラグ

本プログラム例では、On/Off命令にForcedフラグを指定しています。

エラー発生中、および、非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

```
Function main
    Xqt SafetyOnOffMonitor, NoPause
    :
    :
Fend

Function SafetyOnOffMonitor
    Do
        Wait SafetyOn = True
        Print "Saftey Open"
        Off 10, Forced
        On 12, Forced

        Wait SafetyOn = False
        Print "Saftey Close"
        On 10, Forced
        Off 12, Forced
    Loop
Fend
```

3.22.2 SavePoints

メインメモリーにあるポイントデータを、カレントロボットのディスクファイルに保存します。

書式

SavePoints ファイル名

パラメーター

ファイル名

ポイントデータを保存する先のファイル名を、文字列式で指定します。拡張子は、".pts"固定です。パスを指定することはできません。また、ChDiskなどの影響を受けません。詳細は、ChDiskを参照してください。

解説

SavePointsは、指定したファイルにポイントを保存します。拡張子は、".pts"固定です。拡張子を省略した場合、".pts"が追加されます。

また、SavePointsコマンドは、すでにポイントファイルが存在していなければ、カレントロボットのプロジェクトへ追加します。

起こりやすいエラー

- ディスクの容量を超えている場合

ディスク容量が残っていない場合、エラーが発生します。

- 指定したファイルが現在のロボットのファイルでない場合

ファイル名に別のロボットのポイントファイルを指定するとエラーとなります。

- 指定ファイルが見つからない場合

ファイル名にパスが含まれているとエラーとなります。カレントプロジェクトのファイル名のみ指定できます。

- ファイル名のエラー

ファイル名内にスペースや、無効な文字が含まれていた場合、エラーが発生します。

参照

[ImportPoints](#), [LoadPoints](#)

SavePoints使用例

```
Integer i
ClearPoints
For i = 1 To 10
    P(i) = XY( i, 100, 0, 0 )
Next i
SavePoints "TEST.PTS"
```


3.22.3 Seek

指定したファイルのファイルポインター位置を変更します。

書式

Seek #ファイル番号, ポインター

パラメーター

ファイル番号

30～63の整数値または式で指定します。

ポインター

0からファイルサイズまでのファイルポインターを整数値または式で指定します。

参照

[BOpen](#), [Read](#), [ROpen](#), [UOpen](#), [Write](#), [WOpen](#)

Seek使用例

```
Integer fileNum
String data$

fileNumber = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 20
Read #fileNum, data$, 2
Close #fileNum
```

3.22.4 Select...Send

式の値にしたがって、いくつかの文のうちのいずれかに制御を移します。

書式

Select式 Case項目 ステートメント [Case 項目 ステートメント] [Default ステートメント] Send

パラメーター

式

数値式あるいは文字列式を指定します。

項目

式と一致する型の数値または文字列式を指定します。

ステートメント

1つまたは複数の有効なSPEL+ ステートメント、あるいはマルチステートメントを指定します。

解説

Case文の項目の中に、Select文の式の結果と一致するものがあれば、最初に一致したCaseステートメントの後のステートメント群が実行されます。実行後、プログラム制御は、Sendステートメントに続くステートメントに移ります。

Case文の項目の中にSelect文の式の結果と一致するものがなければ、Defaultステートメントが実行され、プログラム制御はSendステートメントに続くステートメントに移ります。

Case文の項目の中にSelect文の式の結果と一致するものがなく、Defaultが省略されると、何も実行されず、プログラム制御はSendステートメントの次のステートメントに移ります。

Select 文の式には、定数, 変数, そしてAnd, Or, Xor などの演算子を指定できます。Case 文の項目にも 定数, 変数, そしてAnd, Or, Xor などの演算子を指定できます。この場合 Case文の項目の演算結果が Select文の式と比較されます。また、動作が複雑になることから Case文の項目に 変数を指定しないでください。

参照

[If...Then...Else...EndIf](#)

Select...Send使用例

下記に簡単なSelect...Sendの例を示します。

```
Function Main
  Integer I
  For i = 0 To 10
    Select I
      Case 0
        Off 1;On 2;Jump P1
      Case 3
        On 1;Off 2
        Jump P2;Move P3;On 3
      Case 7
        On 4
      Default
        On 7
    Send
  Next
End
```

3.22.5 SelectDB関数

オープンしているデータベース内のテーブルからデータを検索する関数です。

書式

SelectDB(#データベース番号, テーブル名, Select条件, Sort方法)

パラメーター

データベース番号

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

テーブル名

データ検索を行うテーブル名を指定します。ファイル番号で指定されたデータベースの種別がExcelブックの場合、Excelシートまたは名前をつけたテーブルを指定します。Excelシートを指定する場合、シート名の後ろに\$をつけ、[]で囲みます。Excelシート内の名前指定した領域を指定する場合、名前を[]で囲みます。

Select条件

検索条件を指定します。AND, ORを使用して複合条件を指定できます。検索条件を指定しない場合、テーブルの全データが検索されます。

Sort方法

検索したデータを抽出する順番を指定します。SortキーとSort順番(昇順[ASC]/降順[DESC])を指定します。Sort順番を省略した場合、Sortキーの昇順を指定したことになります。Sort方法を省略した場合、オープンされているデータベースにより、順番は決定されます。

戻り値

検索された行の総数を返します。

解説

オープンされたデータベースの指定されたテーブルからSelect条件に一致するデータをSort条件によりSortします。

Input#, Print#によりデータの読み込み、書き込みを行う前に必ず実行する必要があります。

オープンされたデータベースがExcelブックの場合、シートおよび名前前で定義された領域の1行目に検索に使用する列名を記述してください。

また、Excel 2007 ブックの場合、シート名を指定してください。名前前で定義された領域へのアクセスは行えません。

注意

RC+ がインストールされているPCの接続が必要です

参照

[OpenDB](#), [CloseDB](#), [UpdateDB](#), [DeleteDB](#), [Input #](#), [Print #](#)

SelectDB関数使用例

SQLデータベースを使用した例

下記に、SQL サーバー2000のサンプルデータベースNorthwindのテーブルEmployeesからTitleOfCourtesyがMs.のデータをEmployeeIDの降順に読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, Title$
```

```
OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees", "TitleOfCourtesy = 'Ms.'", "EmployeeID DESC")
For i = 0 To count - 1
    Input #501, eid, Lastname$, Firstname$, Title$
    Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
Next
CloseDB #501
```

Accessデータベースを使用した例

下記に、Microsoft Access 2007のサンプルデータベース生徒名簿のテーブル生徒から役職が組長のデータをIDの昇順に読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, dummy$

OpenDB #502, Access, "c:\MyDataBase\生徒名簿.accdb"
count = SelectDB(#502, "生徒", "役職 = '組長'", "ID")
For i = 0 To count - 1
    Input #502, eid, dummy$, dummy$, Lastname$, dummy$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #502
```

Excelブックを使用した例

下記に、Microsoft Excelブック生徒名簿の生徒シートからAgeが25未満のデータをIDの昇順に読み込む簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$

OpenDB #503, Excel, "c:\MyDataBase\生徒名簿.xls"
count = SelectDB(#503, "[生徒$]", "Age < 25", "ID ASC")
For i = 0 To count - 1
    Input #503, eid, Lastname$, Firstname$
    Print eid, ",", Lastname$, ",", Firstname$
Next
CloseDB #503
```

3.22.6 Sense

Jump, Jump3, Jump3CPでSenseを指定したときに、目標座標上空で停止させるための条件の設定と表示をします。

書式

Sense [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子(=, <>, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

- 関数: Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
- 変数: Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

- 演算子: And, Or, Xor

[例]

```
Sense Sw(5) = On
Sense Sw(5) = On And Sw(6) = Off
```

解説

Senseは、Jumpコマンド実行中に第3関節の下降開始直前のタイミングで入力条件をチェックします。

また、Jump3, Jump3CPコマンド実行中に接近動作開始直前のタイミングで入力条件をチェックします。

Sense条件式には、上記の関数を1つ以上含んでいなくてはなりません。

Sense条件式に変数が含まれた場合、その値は、Sense条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。Senseステートメントは複数使うこともできます。次のSenseステートメントに移るまで最後に実行された入力条件が有効です。

■ JumpとSense修飾子

現在のSense条件が成立するかチェックします。成立すれば、Jumpコマンドは、ロボットが目標座標の上空で停止した状態で完了したことになります。つまり、Sense条件が"True"のときは、ロボットは目標座標の上空で第3関節が下降開始直前の状態で止まります。Sense条件が"False"のときは、ロボットは目標座標でJump命令の動作を完了します。

■ Jump3, Jump3CPとSense修飾子

現在のSense条件が成立するかチェックします。成立すれば、Jump3, Jump3CPコマンドは、ロボットが接近開始位置で停止した状態で完了したことになります。

パラメーターを省略すると、現在のSense設定を表示します。

注意

- コントローラー電源オン時のSense 設定

コントローラー電源オン時のSense条件は未設定です。電源オン後は、条件を設定してから使用してください。

■ Sense 条件成立をチェックするJS関数とState関数

Sense修飾子を用いた動作コマンドを実行した後で、Sense条件が成立したかどうかを、JS関数やState関数を用いてチェックすることができます。

■ イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が0.01秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1システム内で使用できる変数待ちの個数は最大64個です (Waitなどのイベント条件式で使われた変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数をByrefで参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。

参照

[In関数](#), [JS関数](#), [Jump](#), [Jump3](#), [Jump3CP](#), [MemIn関数](#), [MemSw関数](#), [Stat関数](#), [Sw関数](#), [SF_GetStatus関数](#)

Sense使用例

Sense命令の簡単な使用例です。

```
Function test
.
.
TrySense:
    Sense Sw(1) = Off      '入力ビット1がOffのとき目標座標上空で停止するように設定
    Jump P1 C2 Sense
    If JS = True Then
        GoSub ERRPRC      'アームが目標座標上空で停止している場合は
        GoTo TrySense     'ERRPRCを実行しTrySenseに移動
    EndIf
    On 1; Wait 0.2; Off 1
.
.
Fend
```

[その他の書式例]

```
> Sense Sw(1)=1 And MemSw(1)=1
> Sense Sw(0) Or (Sw(1) And MemSw(1))
```

3.22.7 SetCom

RS-232Cポートのパラメーターを設定または表示します。

書式

SetCom #通信ポート番号 [, 通信速度] [, データビット長] [, ストップビット長] [, パリティ] [, 送受信行末] [, H/Wフロー制御] [, S/Wフロー制御] [, タイムアウト時間]

パラメーター

通信ポート番号

RS-232Cのポート番号を整数値を指定します。

- SPEL+制御部: 1～4
- PC部: 1001～1008

通信速度

ボーレートを指定します。有効値は下記のとおりです。省略可能です。(デフォルト値: 9600)

- 110
- 300
- 600
- 1200
- 2400
- 4800
- 9600
- 14400
- 19200
- 38400
- 57600
- 115200

PC部のポートを使用する場合、19200以上の通信速度ではデータを取りこぼすケースがあります。

データビット長

1文字ずつのデータビット長を7または8の数値で指定します。省略可能です。

ストップビット長

1文字ずつのストップビット長を1または2の数値で指定します。省略可能です。

パリティ

パリティを指定します。奇数の場合はO、偶数の場合はE、なしの場合はNを指定します。省略可能です。

送受信行末

CR, LF, CRLFのいずれかの送受信行末を指定します。省略可能です。

H/Wフロー制御

ハードウェア制御が有効な場合はRTS、無効な場合はNONEを指定します。省略可能です。

S/Wフロー制御

ソフトウェア制御が有効な場合はXON、無効な場合はNONEを指定します。省略可能です。

タイムアウト時間

タイムアウト時間(正の実数値、単位: 秒)を式または数値で指定します。0を指定するとタイムアウトは無限になります。省略可能です。

解説

すべてのパラメーターを省略した場合、通信ポートの設定を表示します。

複数のポートを19200以上の通信速度に設定し同時に通信を行うと、エラー2929または2922が発生する場合があります。この場合、より遅い転送速度を選択するか、同時に通信を行わないようにしてください。

PC部のポートを使用する場合、19200以上の通信速度ではデータを取りこぼすケースがあります。

取りこぼしが発生した場合は、より遅い転送速度を選択するか、SPEL+制御部のポートを使用してください。

参照

[OpenCom](#), [CloseCom](#), [SetNet](#)

SetCom使用例

```
SetCom #1, 9600, 8, 1, N, CRLF, NONE, NONE, 0
```

```
SetCom #2, 4800
```


3.22.8 SetLatch

R-I/O入力によるロボット位置のラッチ機能を設定します。

書式

SetLatch { ポート番号, 入力論理, 連続ラッチ回数 }

パラメーター

ポート番号

トリガー入力信号を接続するR-I/O入力ポートのポート番号を指定します。

指定できるポート番号は以下のとおりです。対象とするロボットが接続されているユニットのポート番号を指定します。

		ポイント	ポート番号
コントロールユニット	入力	4点	24,25, 26, 27
	出力	-	-

ポート番号として以下の定数が定義されています。

定数	ポート番号
SETLATCH_PORT_CU_0	24
SETLATCH_PORT_CU_1	25
SETLATCH_PORT_CU_2	26
SETLATCH_PORT_CU_3	27

入力論理

R-I/Oに接続するトリガー入力信号の論理を指定します。論理は以下の定数で指定することができます。

定数	値	意味
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	負論理
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	正論理

負論理では入力信号のHighからLowへの切り替わりエッジで、ロボット位置をラッチします。正論理では入力信号のLowからHighへの切り替わりエッジで、ロボット位置をラッチします。

連続ラッチ回数

R-I/O入力信号によるロボット位置のラッチの連続回数を指定します。1,2,3,4が指定可能です。

LatchEnable On以降、指定した連続ラッチ回数分のポイントデータのラッチが可能となります。最大4回までのラッチが可能です。パラメーターは省略可能です。省略した場合、連続ラッチ回数は1回となります。

解説

R-I/O入力信号によるロボット位置のラッチ条件を設定します。1台のロボットが複数のポートのトリガー信号を同時に待つことはできません。SetLatchの実行には、約40 msecの処理時間がかかります。

注意

選択されているロボットと関係ない別ユニットのポート番号を指定するとパラメーター範囲外エラーが発生します。

参照

[LatchEnable](#), [LatchState関数](#), [LatchPos関数](#)

SetLatch使用例

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE, 4
  '正論理 連続ラッチ4回設定
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True    'トリガー待ち
  Print LatchPos(WithoutToolArm, 1)    'ラッチ位置1表示
  Print LatchPos(WithoutToolArm, 2)    'ラッチ位置2表示
  Print LatchPos(WithoutToolArm, 3)    'ラッチ位置3表示
  Print LatchPos(WithoutToolArm, 4)    'ラッチ位置4表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

パラメーターを省略する場合の例:

```
Function main
  SetLatch 24, SETLATCH_TRIGGERMODE_LEADINGEDGE    '正論理
  LatchEnable On      'ラッチ機能有効
  Go P1
  Wait LatchState = True    'トリガー待ち
  Print LatchPos      'ラッチ位置表示
  LatchEnable Off      'ラッチ機能無効
Fend
```

3.22.9 SetIn

仮想I/Oの入力ポート (8ビット)を設定します。

書式

SetIn ポート番号, 設定値

パラメーター

ポート番号

I/O入力バイトを指定します。

設定値

範囲が0～255の整数でポート番号を指定します。

解説

仮想I/Oが有効な場合、8個の入力ビットを同時に設定します。

仮想I/Oが無効な場合、本コマンドはエラーになります。

参照

[SetInReal](#), [SetSw](#), [SetInW](#)

SetIn使用例

```
> setin 0, 1   'ポート0の最初のビットをオンに設定
```

3.22.10 SetInReal

仮想I/Oの入力ポート (32ビット)を設定します。

書式

SetInReal ポート番号, 設定値

パラメーター

ポート番号

I/O入力ワードを指定します。

設定値

Real型の値を指定します。

解説

仮想I/Oが有効な場合、32個の入力ビットを同時に設定します。

仮想I/Oが無効な場合、本コマンドはエラーになります。

参照

[SetSw](#), [SetIn](#), [SetInW](#)

SetIn使用例

```
> setinReal 32, 2.543 '入力ポート32に設定
```

3.22.11 SetInW

仮想I/Oの入力ポート (16ビット)を設定します。

書式

SetInW ポート番号, 設定値

パラメーター

ポート番号

I/O入力ワードを指定します。

設定値

範囲が0～65535の整数でワードを指定します。

注意

- リアルタイムI/Oの入力ビットを含むワードポートに関する規則

リアルタイムI/Oの入力ビットに関しては、反映することはできません。

リアルタイムI/Oの入力ビットを含むワードポート=1は、設定値の範囲は0から255の整数で指定してください。255より大きい値は、エラーになります。

解説

仮想I/Oが有効な場合、16個の入力ビットを同時に設定します。

仮想I/Oが無効な場合、本コマンドはエラーになります。

参照

[SetInReal](#), [SetSw](#), [SetIn](#)

SetInW使用例

```
> setinw 0, 1   'ワード0の最初のビットをオンに設定
```

3.22.12 SetIODef

入力, 出力, メモリーI/Oのビット, バイト, ワードのI/Oラベルとコメントを設定します。

書式

SetIODef 種別, インデックス, ラベル, コメント

パラメーター

- 種別: I/Oタイプを表す整数値
 - 1: InputBit
 - 2: InputByte
 - 3: InputWord
 - 4: OutputBit
 - 5: OutputByte
 - 6: OutputWord
 - 7: MemoryBit
 - 8: MemoryByte
 - 9: MemoryWord
 - 10: InputReal
 - 11: OutputReal
- インデックス: ビット、あるいはポート番号を表す整数値
- ラベル: 新規ラベルを表す文字列 (最大32文字)
- コメント: 新規コメントを表す文字列

解説

本コマンドは、各I/Oポイントのラベルやコメントを定義します。

参照

[GetIODef](#), [IONumber関数](#), [IODef関数](#), [IOLabel\\$関数](#)

SetIODef使用例

```
SetIODef 1, 0, "MyInputBit0", "The best input I have ever configured"
```

3.22.13 SetNet

TCP/IPポートのパラメーターの設定をします。

書式

(1) SetNet #通信ポート番号, ホストアドレス [, TCP/IP ポート番号 [, ターミネーター [, フロー制御 [, タイムアウト時間 [, プロトコル [, CloseNetタイムアウト時間]]]]]]]

(2) SetNet

パラメーター

通信ポート番号

パラメーターを設定するTCP/IPのポート番号を指定します。範囲は201～216です。

ホストアドレス

ホストのIPアドレスを指定します。

TCP/IPポート番号

TCP/IPポート番号を指定します。

ターミネーター

CR, LF, CRLFのいずれかの行末文字を指定します。

フロー制御

ソフトウェア制御フローです。NONEを指定します。

タイムアウト時間

送受信の最大時間を秒で指定します。0の場合、タイムアウトは無限となります。

プロトコル

通信のプロトコルを指定します。

- TCP: TCP通信
- UDP: UDP通信
- UDP_SEND: UDP送信
- UDP_RECV: UDP受信

CloseNetタイムアウト時間

CloseNetでソケットを閉じるまでの時間を、秒単位で指定します。(整数: 0 - 5)

0を指定すると、シャットダウン要求に対する応答を待たずに、ソケットを閉じます。

解説

他のIPアドレス・ポート番号と重複した場合、エラーとなります。

参照

[OpenNet](#), [CloseNet](#), [SetCom](#)

SetNet使用例

```
SetNet #201, "192.168.0.1", 2001, CRLF, NONE, 0
```

3.22.14 SetSw

仮想I/Oの入力を設定します。

書式

SetSw ビット番号, 設定値

パラメーター

ビット番号

I/O入力ビットを指定します。

設定値

0 (オフ)か1 (オン)の整数を指定します。

解説

仮想I/Oが有効な場合、入力ビットを設定します。

仮想I/Oが無効な場合、本コマンドはエラーになります。

参照

[SetInReal](#), [SetIn](#), [SetInW](#)

SetSw使用例

```
> setsw 2, on  '2つ目の入力ビットをオンに設定
```


3.22.15 SF_GetParam関数

安全機能パラメーターの情報を返します。

書式

SF_GetParam (インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値、または定数で指定します。定数末尾が、"_EN"のインデックスを指定した場合、有効な場合に1、無効な場合に0を返します。

戻り値

指定した情報の整数値を返します。

解説

指定した安全機能パラメーター値を返します。

インデックス	定数	解説
1	DRYRUNOFF	ドライラン無効状態
2	SLS_1_HAND_EN	SLS_1のハンド速度監視状態
3	SLS_1_SPEED	SLS_1の監視速度設定値
4	SLS_1_ELBOW_EN	SLS_1の肘 (水平多関節型: J2, 垂直6軸型: J3) 速度監視状態 *1
5	SLS_1_JOINT_EN	SLS_1の関節速度監視状態
6	SLS_1_JOINTSPEED	SLS_1の監視関節速度設定値
7	SLS_1_WRIST_EN	SLS_1の手首(垂直6軸型: J5)速度監視状態 *1
8	SLS_1_SHOULDER_EN	SLS_1の肩(垂直6軸型: J2)速度監視状態 *1
9	SLS_2_HAND_EN	SLS_2のハンド速度監視状態
10	SLS_2_SPEED	SLS_2の監視速度設定値
11	SLS_2_ELBOW_EN	SLS_2の肘 (水平多関節型: J2, 垂直6軸型: J3) 速度監視状態 *1
12	SLS_2_JOINT_EN	SLS_2の関節速度監視状態
13	SLS_2_JOINTSPEED	SLS_2の監視関節速度設定値
14	SLS_2_WRIST_EN	SLS_2の手首(垂直6軸型: J5)速度監視状態 *1
15	SLS_2_SHOULDER_EN	SLS_2の肩(垂直6軸型: J2)速度監視状態 *1
16	SLS_3_HAND_EN	SLS_3のハンド速度監視状態

インデックス	定数	解説
17	SLS_3_SPEED	SLS_3の監視速度設定値
18	SLS_3_ELBOW_EN	SLS_3の肘 (水平多関節型: J2, 垂直6軸型: J3) 速度監視状態 *1
19	SLS_3_JOINT_EN	SLS_3の関節速度監視状態
20	SLS_3_JOINTSPEED	SLS_3の監視関節速度設定値
21	SLS_3_WRIST_EN	SLS_3の手首(垂直6軸型: J5)速度監視状態 *1
22	SLS_3_SHOULDER_EN	SLS_3の肩(垂直6軸型: J2)速度監視状態 *1
23	SLS_T2_HAND_EN	SLS_T2のハンド速度監視状態
24	SLS_T2_SPEED	SLS_T2の監視速度設定値
25	SLS_T2_ELBOW_EN	SLS_T2の肘 (水平多関節型: J2, 垂直6軸型: J3) 速度監視状態 *1
26	SLS_T2_JOINT_EN	SLS_T2の関節速度監視状態
27	SLS_T2_JOINTSPEED	SLS_T2の監視関節速度設定値
28	SLS_T2_WRIST_EN	SLS_T2の手首(垂直6軸型: J5)速度監視状態 *1
29	SLS_T2_SHOULDER_EN	SLS_T2の肩(垂直6軸型: J2)速度監視状態 *1
30	SLS_T_SPEED	SLS_Tの監視速度設定値
31	SLS_T_JOINT_EN	SLS_Tの関節速度監視状態
32	SLS_T_JOINTSPEED	SLS_Tの監視関節速度設定値
33	SLS_HAND_OFS_X	SLSのX軸方向 TCPオフセット位置
34	SLS_HAND_OFS_Y	SLSのY軸方向 TCPオフセット位置
35	SLS_HAND_OFS_Z	SLSのZ軸方向 TCPオフセット位置
36	SLS_1_DELAY	SLS_1の遅延時間設定値
37	SLS_2_DELAY	SLS_2の遅延時間設定値
38	SLS_3_DELAY	SLS_3の遅延時間設定値
39	SLS_JOINT_POS_EN	関節角度監視の監視状態
40	SLS_JOINT_POS_ANGLE	監視関節角度設定値
41	SLP_A_XU_EN	SLP_AのXU (壁: X2, 制限領域: X1)位置監視状態 *2
42	SLP_A_XU_POS	SLP_AのXU (壁: X2, 制限領域: X1)監視位置 設定値 *2

インデックス	定数	解説
43	SLP_A_XL_EN	SLP_AのXL (壁: X1, 制限領域: X2)位置監視状態 *2
44	SLP_A_XL_POS	SLP_AのXL (壁: X1, 制限領域: X2)監視位置 設定値 *2
45	SLP_A_YU_EN	SLP_AのYU (壁: Y2, 制限領域: Y1)位置監視状態 *2
46	SLP_A_YU_POS	SLP_AのYU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
47	SLP_A_YL_EN	SLP_AのYL (壁: Y1, 制限領域: Y2)位置監視状態 *2
48	SLP_A_YL_POS	SLP_AのYL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
49	SLP_A_ZU_EN	SLP_AのZ2 位置監視状態 *2
50	SLP_A_ZU_POS	SLP_AのZ2 監視位置 設定値 *2
51	SLP_A_ZL_EN	SLP_AのZ1 位置監視状態 *2
52	SLP_A_ZL_POS	SLP_AのZ1 監視位置 設定値 *2
53	SLP_B_XU_EN	SLP_BのXU (壁: X2, 制限領域: X1)位置監視状態 *2
54	SLP_B_XU_POS	SLP_BのXU (壁: X2, 制限領域: X1)監視位置 設定値 *2
55	SLP_B_XL_EN	SLP_BのXL (壁: X1, 制限領域: X2)位置監視状態 *2
56	SLP_B_XL_POS	SLP_BのXL (壁: X1, 制限領域: X2)監視位置 設定値 *2
57	SLP_B_YU_EN	SLP_BのYU (壁: Y2, 制限領域: Y1)位置監視状態 *2
58	SLP_B_YU_POS	SLP_BのYU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
59	SLP_B_YL_EN	SLP_BのYL (壁: Y1, 制限領域: Y2)位置監視状態 *2
60	SLP_B_YL_POS	SLP_BのYL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
61	SLP_B_ZU_EN	SLP_BのZ2 位置監視状態 *2
62	SLP_B_ZU_POS	SLP_BのZ2 監視位置 設定値 *2
63	SLP_B_ZL_EN	SLP_BのZ1 位置監視状態 *2
64	SLP_B_ZL_POS	SLP_BのZ1 監視位置 設定値 *2
65	SLP_C_XU_EN	SLP_CのXU (壁: X2, 制限領域: X1)位置監視状態 *2
66	SLP_C_XU_POS	SLP_CのXU (壁: X2, 制限領域: X1)監視位置 設定値 *2
67	SLP_C_XL_EN	SLP_CのXL (壁: X1, 制限領域: X2)位置監視状態 *2
68	SLP_C_XL_POS	SLP_CのXL (壁: X1, 制限領域: X2)監視位置 設定値 *2
69	SLP_C_YU_EN	SLP_CのYU (壁: Y2, 制限領域: Y1)位置監視状態 *2

インデックス	定数	解説
70	SLP_C_YU_POS	SLP_CのYU (壁: Y2, 制限領域: Y1)監視位置 設定値 *2
71	SLP_C_YL_EN	SLP_CのYL (壁: Y1, 制限領域: Y2)位置監視状態 *2
72	SLP_C_YL_POS	SLP_CのYL (壁: Y1, 制限領域: Y2)監視位置 設定値 *2
73	SLP_C_ZU_EN	SLP_CのZ2 位置監視状態 *2
74	SLP_C_ZU_POS	SLP_CのZ2 監視位置 設定値 *2
75	SLP_C_ZL_EN	SLP_CのZ1 位置監視状態 *2
76	SLP_C_ZL_POS	SLP_CのZ1 監視位置 設定値 *2
77	SLP_J2_MON_RAD	SLPのJ2軸監視範囲半径設定値
78	SLP_J3_MON_RAD	SLPのJ3軸監視範囲半径設定値
79	SLP_J5_MON_RAD	SLPのJ5軸監視範囲半径設定値
80	SLP_J6_MON_RAD	SLPのJ6軸監視範囲半径設定値
81	SLP_J1_RANGE_MAX	ソフト軸制限のJ1軸制限範囲最大設定値
82	SLP_J1_RANGE_MIN	ソフト軸制限のJ1軸制限範囲最小設定値
83	SLP_J2_RANGE_MAX	ソフト軸制限のJ2軸制限範囲最大設定値
84	SLP_J2_RANGE_MIN	ソフト軸制限のJ2軸制限範囲最小設定値
85	SLP_J3_RANGE_MAX	ソフト軸制限のJ3軸制限範囲最大設定値
86	SLP_J3_RANGE_MIN	ソフト軸制限のJ3軸制限範囲最小設定値
87	SLP_J4_RANGE_MAX	ソフト軸制限のJ4軸制限範囲最大設定値
88	SLP_J4_RANGE_MIN	ソフト軸制限のJ4軸制限範囲最小設定値
89	SLP_J5_RANGE_MAX	ソフト軸制限のJ5軸制限範囲最大設定値
90	SLP_J5_RANGE_MIN	ソフト軸制限のJ5軸制限範囲最小設定値
91	SLP_J6_RANGE_MAX	ソフト軸制限のJ6軸制限範囲最大設定値
92	SLP_J6_RANGE_MIN	ソフト軸制限のJ6軸制限範囲最小設定値
93	SIN_1_SLS_1_EN	SAFETY_IN1に対するSLS_1機能割り当て状態
94	SIN_1_SLS_2_EN	SAFETY_IN1に対するSLS_2機能割り当て状態
95	SIN_1_SLS_3_EN	SAFETY_IN1に対するSLS_3機能割り当て状態
96	SIN_1_SLP_A_EN	SAFETY_IN1に対するSLP_A機能割り当て状態

インデックス	定数	解説
97	SIN_1_SLP_B_EN	SAFETY_IN1に対するSLP_B機能割り当て状態
98	SIN_1_SLP_C_EN	SAFETY_IN1に対するSLP_C機能割り当て状態
99	SIN_1_SG_EN	SAFETY_IN1に対する保護停止機能割り当て状態
100	SIN_1_ESTOP_EN	SAFETY_IN1に対する非常停止機能割り当て状態
101	SIN_2_SLS_1_EN	SAFETY_IN2に対するSLS_1機能割り当て状態
102	SIN_2_SLS_2_EN	SAFETY_IN2に対するSLS_2機能割り当て状態
103	SIN_2_SLS_3_EN	SAFETY_IN2に対するSLS_3機能割り当て状態
104	SIN_2_SLP_A_EN	SAFETY_IN2に対するSLP_A機能割り当て状態
105	SIN_2_SLP_B_EN	SAFETY_IN2に対するSLP_B機能割り当て状態
106	SIN_2_SLP_C_EN	SAFETY_IN2に対するSLP_C機能割り当て状態
107	SIN_2_SG_EN	SAFETY_IN2に対する保護停止機能割り当て状態
108	SIN_2_ESTOP_EN	SAFETY_IN2に対する非常停止機能割り当て状態
109	SIN_3_SLS_1_EN	SAFETY_IN3に対するSLS_1機能割り当て状態
110	SIN_3_SLS_2_EN	SAFETY_IN3に対するSLS_2機能割り当て状態
111	SIN_3_SLS_3_EN	SAFETY_IN3に対するSLS_3機能割り当て状態
112	SIN_3_SLP_A_EN	SAFETY_IN3に対するSLP_A機能割り当て状態
113	SIN_3_SLP_B_EN	SAFETY_IN3に対するSLP_B機能割り当て状態
114	SIN_3_SLP_C_EN	SAFETY_IN3に対するSLP_C機能割り当て状態
115	SIN_3_SG_EN	SAFETY_IN3に対する保護停止機能割り当て状態
116	SIN_3_ESTOP_EN	SAFETY_IN3に対する非常停止機能割り当て状態
117	SIN_4_SLS_1_EN	SAFETY_IN4に対するSLS_1機能割り当て状態
118	SIN_4_SLS_2_EN	SAFETY_IN4に対するSLS_2機能割り当て状態
119	SIN_4_SLS_3_EN	SAFETY_IN4に対するSLS_3機能割り当て状態
120	SIN_4_SLP_A_EN	SAFETY_IN4に対するSLP_A機能割り当て状態
121	SIN_4_SLP_B_EN	SAFETY_IN4に対するSLP_B機能割り当て状態
122	SIN_4_SLP_C_EN	SAFETY_IN4に対するSLP_C機能割り当て状態
123	SIN_4_SG_EN	SAFETY_IN4に対する保護停止機能割り当て状態

インデックス	定数	解説
124	SIN_4_ESTOP_EN	SAFETY_IN4に対する非常停止機能割り当て状態
125	SIN_5_SLS_1_EN	SAFETY_IN5に対するSLS_1機能割り当て状態
126	SIN_5_SLS_2_EN	SAFETY_IN5に対するSLS_2機能割り当て状態
127	SIN_5_SLS_3_EN	SAFETY_IN5に対するSLS_3機能割り当て状態
128	SIN_5_SLP_A_EN	SAFETY_IN5に対するSLP_A機能割り当て状態
129	SIN_5_SLP_B_EN	SAFETY_IN5に対するSLP_B機能割り当て状態
130	SIN_5_SLP_C_EN	SAFETY_IN5に対するSLP_C機能割り当て状態
131	SIN_5_SG_EN	SAFETY_IN5に対する保護停止機能割り当て状態
132	SIN_5_ESTOP_EN	SAFETY_IN5に対する非常停止機能割り当て状態
133	SOUT_1_STO	SAFETY_OUT1に対するSTO機能割り当て状態
134	SOUT_1_SLS_1	SAFETY_OUT1に対するSLS_1機能割り当て状態
135	SOUT_1_SLS_2	SAFETY_OUT1に対するSLS_2機能割り当て状態
136	SOUT_1_SLS_3	SAFETY_OUT1に対するSLS_3機能割り当て状態
137	SOUT_1_SLS_T2	SAFETY_OUT1に対するSLS_T2機能割り当て状態
138	SOUT_1_SLS_T	SAFETY_OUT1に対するSLS_T機能割り当て状態
139	SOUT_1_SLP_A	SAFETY_OUT1に対するSLP_A機能割り当て状態
140	SOUT_1_SLP_B	SAFETY_OUT1に対するSLP_B機能割り当て状態
141	SOUT_1_SLP_C	SAFETY_OUT1に対するSLP_C機能割り当て状態
142	SOUT_1_EP_RC	SAFETY_OUT1に対する非常停止 (コントローラー)機能割り当て状態
143	SOUT_1_EP_TP	SAFETY_OUT1に対する非常停止 (ティーチペンダント)機能割り当て状態
144	SOUT_1_EN_SW	SAFETY_OUT1に対するイネーブルスイッチ機能割り当て状態
145	SOUT_2_STO	SAFETY_OUT2に対するSTO機能割り当て状態
146	SOUT_2_SLS_1	SAFETY_OUT2に対するSLS_1機能割り当て状態
147	SOUT_2_SLS_2	SAFETY_OUT2に対するSLS_2機能割り当て状態
148	SOUT_2_SLS_3	SAFETY_OUT2に対するSLS_3機能割り当て状態
149	SOUT_2_SLS_T2	SAFETY_OUT2に対するSLS_T2機能割り当て状態

インデックス	定数	解説
150	SOUT_2_SLS_T	SAFETY_OUT2に対するSLS_T機能割り当て状態
151	SOUT_2_SLP_A	SAFETY_OUT2に対するSLP_A機能割り当て状態
152	SOUT_2_SLP_B	SAFETY_OUT2に対するSLP_B機能割り当て状態
153	SOUT_2_SLP_C	SAFETY_OUT2に対するSLP_C機能割り当て状態
154	SOUT_2_EP_RC	SAFETY_OUT2に対する非常停止 (コントローラー)機能割り当て状態
155	SOUT_2_EP_TP	SAFETY_OUT2に対する非常停止 (ティーチペンダント)機能割り当て状態
156	SOUT_2_EN_SW	SAFETY_OUT2に対するイネーブルスイッチ機能割り当て状態
157	SOUT_3_STO	SAFETY_OUT3に対するSTO機能割り当て状態
158	SOUT_3_SLS_1	SAFETY_OUT3に対するSLS_1機能割り当て状態
159	SOUT_3_SLS_2	SAFETY_OUT3に対するSLS_2機能割り当て状態
160	SOUT_3_SLS_3	SAFETY_OUT3に対するSLS_3機能割り当て状態
161	SOUT_3_SLS_T2	SAFETY_OUT3に対するSLS_T2機能割り当て状態
162	SOUT_3_SLS_T	SAFETY_OUT3に対するSLS_T機能割り当て状態
163	SOUT_3_SLP_A	SAFETY_OUT3に対するSLP_A機能割り当て状態
164	SOUT_3_SLP_B	SAFETY_OUT3に対するSLP_B機能割り当て状態
165	SOUT_3_SLP_C	SAFETY_OUT3に対するSLP_C機能割り当て状態
166	SOUT_3_EP_RC	SAFETY_OUT3に対する非常停止 (コントローラー)機能割り当て状態
167	SOUT_3_EP_TP	SAFETY_OUT3に対する非常停止 (ティーチペンダント)機能割り当て状態
168	SOUT_3_EN_SW	SAFETY_OUT3に対するイネーブルスイッチ機能割り当て状態
169	POS_ROT_U	設定平面回転 U_ROT 設定値
170	POS_ROT_V	設定平面回転 V_ROT 設定値
171	POS_ROT_W	設定平面回転 W_ROT 設定値
172	POS_OFS_X	設置位置オフセット X_OFS 設定値
173	POS_OFS_Y	設置位置オフセット Y_OFS 設定値

インデックス	定数	解説
174	POS_OFS_Z	設置位置オフセット Z_OFS 設定値

*1 安全機能マネージャーの安全速度監視の監視部位J2, J3, J5と、本マニュアルで扱う速度超過部位ハンド, 手首, 肘, 肩との対応は以下のとおりです。

- 水平多関節型
 - J2: 肘
 - J3: 該当なし
- 垂直6軸型
 - J2: 肩
 - J3: 肘
 - J5: 手首
 - Hand: ハンド

*2 安全機能マネージャーの安全位置監視の監視位置X1, X2, Y1, Y2, Z1, Z2と、本マニュアルで扱う監視位置XL, XU, YL, YU, ZL, ZUとの対応は以下のとおりです。

- 監視位置で「壁」選択時
 - X1 = XL, X2 = XU
 - Y1 = YL, Y2 = YU
 - Z1 = ZL, Z2 = ZU (垂直6軸型のみ)
- 監視位置で「制限領域」選択時
 - X1 = XU, X2 = XL
 - Y1 = YU, Y2 = YL
 - Z1 = ZL, Z2 = ZU (垂直6軸型のみ)

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_GetParam関数使用例

```
If SF_GetParam (SLS_1_HAND_EN) = 1 Then
Print "SLS_1 hand speed monitoring is enabled."
EndIf
```


3.22.16 SF_GetParam\$関数

安全機能パラメーターのテキスト情報を返します。

書式

SF_GetParam\$(インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値、または定数で指定します。

戻り値

指定した情報の文字列値を返します。

解説

指定した安全機能パラメーター値を返します。

インデックス	定数	解説
1	SF_TOOLVERSION	設定ツールバージョン
2	SF_CHECKSUM	安全機能パラメーターチェックサム
3	SF_LAST_MODIFIED	安全機能パラメーター更新日
4	SF_ROBOT_MODEL_NAME	ロボットモデル名
5	SF_ROBOT_CHECKSUM	ロボットパラメーターチェックサム
6	SF_HOFS	物理的なロボットの原点に対するエンコーダーオフセット値 (Hofs)
7	SF_HOFS_LAST_MODIFIED	Hofs更新日

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_GetParam\$関数使用例

```
String Checksum$
Checksum$ = SF_GetParam$(SF_CHECKSUM)
Print "Safety function parameter Checksum is ", Checksum$

> Print SF_GetParam$(SF_LAST_MODIFIED)
2022/01/01 00:00:00
```

3.22.17 SF_GetStatus関数

安全機能の状態ビットを返します。

書式

SF_GetStatus (インデックス)

パラメーター

インデックス

検索する情報のインデックスを整数値で指定します。

戻り値

指定したインデックスの情報を整数値で返します。

解説

戻り値のビット情報は次の表のとおりです。

インデックス	ビット	値	解説
0	0-6	-	予約済
	7	&H80	Safety基板の故障検知
1	0	&H1	SLS_1機能有効
	1	&H2	SLS_2機能有効
	2	&H4	SLS_3機能有効
	3-7	-	予約済
2	0	&H1	SLP_A機能有効
	1	&H2	SLP_B機能有効
	2	&H4	SLP_C機能有効
	3-6	-	予約済
	7	&H80	ソフト軸制限機能有効 (常時有効)
3	0	&H1	SAFETY_IN1信号High (機能オフ) *1
	1	&H2	SAFETY_IN2信号High (機能オフ) *1
	2	&H4	SAFETY_IN3信号High (機能オフ) *1
	3	&H8	SAFETY_IN4信号High (機能オフ) *1
	4	&H10	SAFETY_IN5信号High (機能オフ) *1
	5	&H20	非常停止スイッチ (ティーチペンダント) High (機能オフ) *1
	6	&H40	非常停止スイッチ (コントローラー接続) High (機能オフ) *1

インデックス	ビット	値	解説
	7	&H80	Enable信号High (機能オフ) *1
4	0	&H1	SAFETY_OUT1信号High (機能オフ) *2
	1	&H2	SAFETY_OUT2信号High (機能オフ) *2
	2	&H4	SAFETY_OUT3信号High (機能オフ) *2
	3-7	-	予約済
5	0	&H1	SLP_A *3
	1	&H2	SLP_B *3
	2	&H4	SLP_C *3
	3-6	-	予約済
	7	&H80	SLP_J *3
6～11	0-7	-	予約済
12	0-7	-	STF_ID
13	0-7	-	STF_DET_L
14	0-7	-	STF_DET_U
15	0-7	-	予約済

*1. 安全入力信号は負論理 (Active Low) です。

安全入力に信号レベルHighが入力されると、本関数は1を返し、安全入力に割り当てた機能は動作しません。

安全入力に信号レベルLowが入力されると、本関数は0を返し、安全入力に割り当てた機能が動作します。

安全入力に機器を接続しないと、この状態になります。

*2 安全出力信号は負論理 (Active Low) です。

安全出力に割り当てたいずれかの機能が動作状態の場合、安全出力が有効になり、安全出力の信号レベルはLowになり、本関数は0を返します。

安全出力に割り当てたいずれの機能も動作状態ではない場合、安全出力が無効になり、安全出力の信号レベルはHighになり、本関数は1を返します。

安全出力にいずれの機能も割り当てない場合、安全出力の信号レベルはLowになり、本関数は0を返します。

*3 安全入力にSLP機能が割り当てられているか割り当てられていないかにかかわらず、また入力信号が有効か無効にかかわらず、監視位置に入った場合は1となります。

STF_ID, およびSTF_DET_L, STF_DET_Hを使用するとエラー発生原因を確認できます。SPEL+を使った確認方法は後述します。STF_ID, およびSTF_DET_L, STF_DET_Hの情報は次の通りです。

STF_ID	エラー名	STF_DET_L	STF_DET_U
100	停止通知 安全入力	安全入力ポート SAFETY_IN1 &H01 SAFETY_IN2 &H02 SAFETY_IN3 &H04 SAFETY_IN4 &H08 SAFETY_IN5 &H10	不使用
101	停止通知 SLS_1 速度超過 関節	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
102	停止通知 SLS_1 速度超過 部位	部位 ハンド &H01 / 肘 &H02*1 手首 &H04 / 肩 &H08 *1	不使用
103	停止通知 SLS_2 速度超過 関節	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
104	停止通知 SLS_2 速度超過 部位	部位 ハンド &H01 / 肘 &H02*1 手首 &H04 / 肩 &H08 *1	不使用
105	停止通知 SLS_3 速度超過 関節	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
106	停止通知 SLS_3 速度超過 部位	部位 ハンド &H01 / 肘 &H02*1 手首 &H04 / 肩 &H08 *1	不使用
107	停止通知 SLS_T 速度超過 関節	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
108	停止通知 SLS_T 速度超過 部位	部位 ハンド &H01 / 肘 &H02*1 手首 &H04 / 肩 &H08 *1	不使用
109	停止通知 SLS_T2 速度超過 関節	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
110	停止通知 SLS_T2 速度超過 部位	部位 ハンド &H01 / 肘 &H02*1 手首 &H04 / 肩 &H08 *1	不使用

STF_ID	エラー名	STF_DET_L	STF_DET_U
115	停止通知 SLP_A位置違反 監視位置	監視位置 YL &H01 / YU &H02 XL &H04 / XU &H08 ZL &H10 / ZU &H20 *2	関節番号 J6 &H08 J5 &H04 J3 &H02 J2 &H01
116	停止通知 SLP_B位置違反 監視位置	監視位置 YL &H01 / YU &H02 XL &H04/ XU &H08 ZL &H10 / ZU &H20 *2	関節番号 J6 &H08 J5 &H04 J3 &H02 J2 &H01
117	停止通知 SLP_C 位置違反 監視位置	監視位置 YL &H01 / YU &H02 XL &H04 / XU &H08 ZL &H10 / ZU &H20 *2	関節番号 J6 &H08 J5 &H04 J3 &H02 J2 &H01
118	停止通知 ソフト軸制限	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
121	停止通知 スイッチ入力	スイッチ番号 イネーブルスイッチ &H01 非常停止スイッチ(ティーチペンダント) &H02 非常停止スイッチ(コントローラー接続) &H04	不使用
122	停止通知 モード制御	モード パラメーター通信許可 &H08 安全機能 (Safety基板) 無効化 &H04 操作モード切替 &H02 パラメーター設定 認証済み &H01	不使用
123	停止通知 減速監視	検出異常 減速異常&H08, &H04 減速完了 &H02 監視時間経過 &H01	不使用
124	停止通知 関節角度監視	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
131	故障停止通知 エンコーダー通信異常	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
132	故障停止通知 位置異常	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用

STF_ID	エラー名	STF_DET_L	STF_DET_U
133	故障停止通知 入力二重化異常	異常検出部位 安全入力ポート SAFETY_IN 1 &H01 SAFETY_IN 2 &H02 SAFETY_IN 3 &H04 SAFETY_IN 4 &H08 SAFETY_IN 5 &H10 イネーブルスイッチ &H20 非常停止スイッチ(ティーチペンダント) &H40 非常停止スイッチ(コントローラー接続) &H80	不使用
134	故障停止通知 出力二重化異常	異常検出部位 安全出力ポート SAFETY_OUT 1 &H01 SAFETY_OUT 2 &H02 SAFETY_OUT 3 &H04 STO &H80	不使用
135	故障停止通知 Safety基板異常	異常検出部位 通信バス &H20 電源(3.3V) &H08 電源(5V) &H04 ウォッチドックタイマー検知 &H02 リレー溶着 &H01	不使用
136	故障停止通知 Safety基板 MCU異常	異常検出部位 シーケンスモニタ &H10 CPU &H08 RAM &H04 プログラムROM &H02 データROM &H01	DET_L = &H01のとき (データROM) &H00 - &HFE データ故障箇所 &HFF パラメーターの故障
137	故障停止通知 Safety基板 二重化内部異常	異常検出部位 TCP位置不一致 &H02 状態不一致 &H01	不使用
138	故障停止通知 エンコーダー 内部異常	関節番号 J1 &H01 / J2 &H02 J3 &H04 / J4 &H08 J5 &H10 / J6 &H20	不使用
139	故障停止通知 コントローラー 内部異常	異常検出部位 操作モード受信エラー &H01	不使用

*1 安全機能マネージャーの安全速度監視の監視部位J2, J3, J5と、本マニュアルで扱う速度超過部位ハンド, 手首, 肘, 肩との対応は以下のとおりです。

- 水平多関節型
 - J2: 肘
 - J3: 該当なし
 - J5: 該当なし

- Hand: ハンド
- 垂直6軸型
 - J2: 肩
 - J3: 肘
 - J5: 手首
 - Hand: ハンド

*2 安全機能マネージャーの安全位置監視の監視位置X1, X2, Y1, Y2, Z1, Z2と、本マニュアルで扱う監視位置XL, XU, YL, YU, ZL, ZUとの対応は以下のとおりです。

- 監視位置で「壁」選択時
 - X1 = XL, X2 = XU
 - Y1 = YL, Y2 = YU
 - Z1 = ZL, Z2 = ZU (垂直6軸型のみ)
- 監視位置で「制限領域」選択時
 - X1 = XU, X2 = XL
 - Y1 = YU, Y2 = YL
 - Z1 = ZL, Z2 = ZU (垂直6軸型のみ)

STF_ID、およびSTF_DET_L、STF_DET_Hを使用したエラー発生原因の確認方法

コマンドウィンドウ等で以下の順番でコマンドを入力、確認します。

```
> SF_GetStatus (12)
115      '「停止通知 SLP_A 位置違反」によるエラーを表します。
> SF_GetStatus (13)
1      '「YL」方向の監視位置を越えたことを表します。(STF_ID: 115の内容を確認します。)
> SF_GetStatus (14)
1      '「J2軸」が越えたことを表します。(STF_ID: 115の内容を確認します。)
```

以上の情報をまとめると、「J2軸がSLP_AのYLの監視位置を越えた」ためエラーが発生したことがわかります。

エラー情報はEpson RC+のシステムヒストリーにも記録されます。エラー発生原因は「追加情報」に記録されます。SPEL+ランゲージリファレンスのSPEL+ エラーメッセージを参照してください。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_GetStatus関数使用例

```
If (SF_GetStatus(3) And &H1) = &H1 Then
Print "SAFETY_IN1 is High"
Else
Print "SAFETY_IN1 is Low"
EndIf
```

3.22.18 SF_LimitSpeedS

安全速度監視(SLS)有効時に、速度を調整する機能について速度調整値を設定、表示します。

書式

SF_LimitSpeedS [SLS番号 [, 速度調整値]]

パラメーター

SLS番号

SLS番号を整数値、または以下に示す定数で指定します。省略可能です。

定数	値	内容
SLS_1	1	SLS_1有効時の速度調整値を設定、表示します
SLS_2	2	SLS_2有効時の速度調整値を設定、表示します
SLS_3	3	SLS_3有効時の速度調整値を設定、表示します
SLS_T	9	SLS_T有効時の速度調整値を表示します *1
SLS_T2	10	SLS_T2有効時の速度調整値を表示します *1

*1 SLS_TとSLS_T2については、本コマンドで速度調整値を設定することはできません。安全機能マネージャーで設定した監視速度で速度調整します。詳細は以下のマニュアルを参照してください。

"ロボットコントローラー 安全機能マニュアル"

速度調整値

速度を整数値 (0～10000、単位: mm/s)で指定します。省略可能です。0を指定した場合、自動で速度調整値が設定されます。初期値は0です。

解説

指定したSLS番号の速度調整値[mm/s]を設定します。第2パラメーターが省略されているときは、指定したSLS番号の速度調整値を表示します。

全パラメーターが省略されているときは、全てのSLS番号の速度調整値を表示します。

安全機能オプションが有効のときのみ使用できます。

速度調整対象

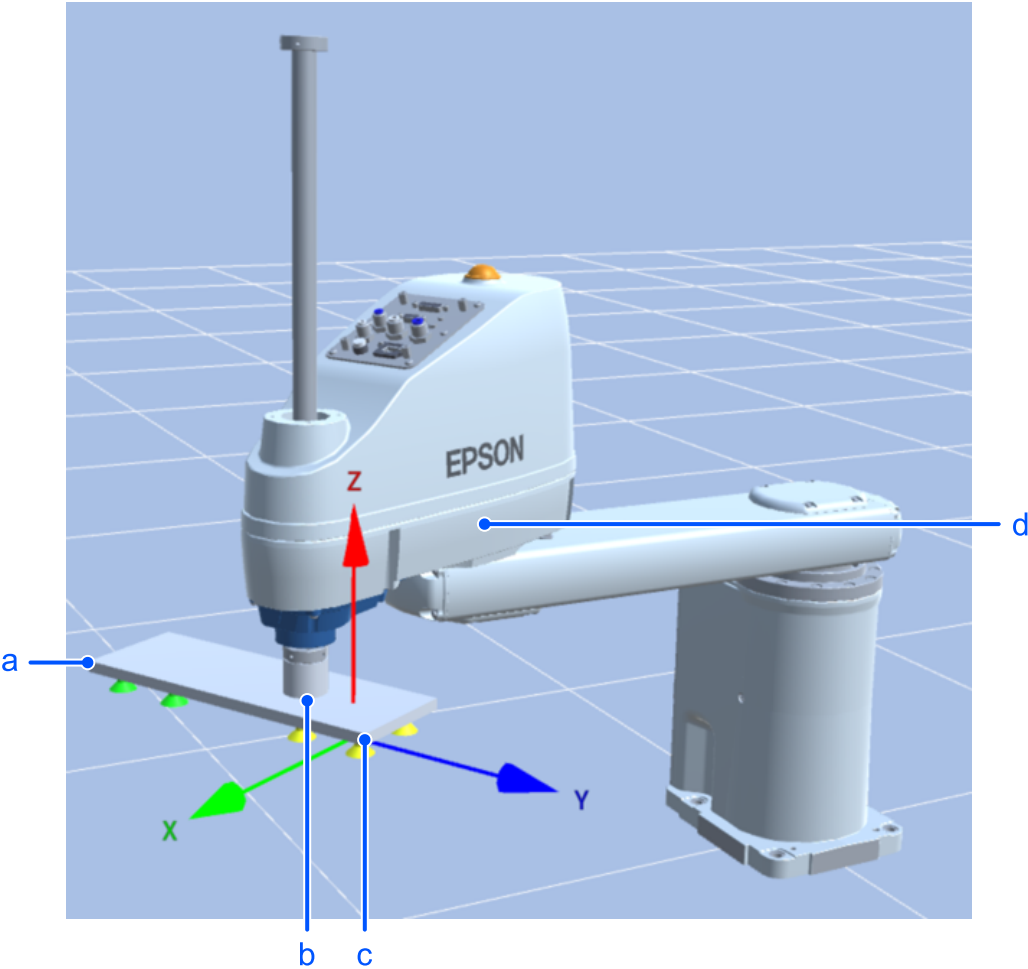
安全速度監視(SLS)有効時に✓で示した部位を速度調整します。

安全機能マネージャーでSLS速度監視する部位を選択することができますが、安全機能マネージャーの設定によらず、下記の表に従い速度調整します。

■ スカラロボット

	SLS監視可能部位 *1		SLS監視対象外	
	ハンド	肘	シャフト先端	ツール位置 *2
PTP動作	✓	✓	✓	✓
CP動作				✓

- *1 詳細は以下を参照してください。
"ロボットコントローラー安全機能マニュアル"
- *2 Toolコマンドで現在選択されているツール位置です。

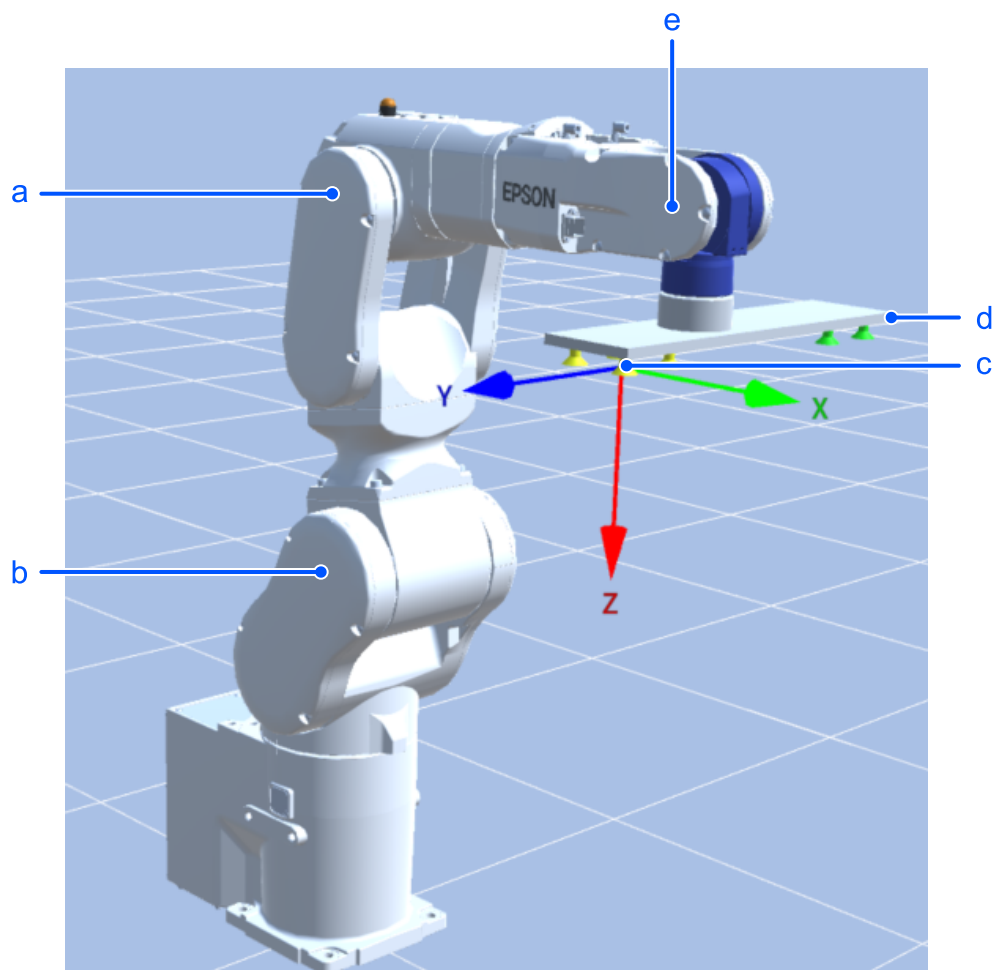


記号	説明
a	ハンド(SLS監視可能部位)
b	シャフト先端
c	ツール位置
d	肘(SLS監視可能部位)

■ 6軸ロボット

	SLS監視可能部位 *1				SLS監視対象外
	ハンド	肘	手首	肩	ツール位置 *2
PTP動作	✓	✓	✓	✓	✓
CP動作					✓

- *1 詳細は以下を参照してください。
「ロボットコントローラー安全機能マニュアル」
- *2 Toolコマンドで現在選択されているツール位置です。



記号	説明
a	肘(SLS監視可能部位)
b	肩(SLS監視可能部位)
c	ツール位置
d	ハンド(SLS監視可能部位)
e	手首(SLS監視可能部位)

注意

- SF_LimitSpeedSEnableは特異点回避, コンベヤトラッキング, 力制御機能と同時に使えません
SF_LimitSpeedSEnableのデフォルト設定はOnです。特異点回避, コンベヤトラッキング, 力制御機能の動作中に本機能で速度調整を行うと動作エラー (それぞれ4093, 4403, 5830) になります。また、SF_LimitSpeedSEnableがOnの場合、コンベヤ動作開始時に4500エラーが発生します。
これらの動作でSLSを使う場合、動作開始前にSF_LimitSpeedSEnableをOffにしてください。また、SF_PeakSpeedS, SF_RealSpeedSを使ってロボットアーム動作速度を計測し、SLS監視速度 (安全機能パラメーター) を超えないようにロボットアーム動作速度をSpeed, SpeedFactor, SpeedSなどを使用して制御してください。
- CP動作の場合、SF_LimitSpeedSで速度調整される部位は現在選択されているツール位置です
安全機能マネージャのTCPオフセットで指定した位置は、Toolコマンドに自動設定されません。Toolコマンドを利用し、適切なTCP位置を設定してください。

- 本機能は安全速度監視(SLS)の監視速度を超えないことを保証するものではありません
SLSの監視速度を超過する場合は、SF_PeakSpeedS, SF_RealSpeedSを使ってロボットアーム動作速度を計測し、SLSの監視速度を超えないようにロボットアーム動作速度をSpeed, SpeedFactor, SpeedSなどを使用して制御してください。

ヒント

速度調整値はコントローラー起動時には0（自動設定）です。通常は0のまま利用することを推奨します。

Go命令などのPTP動作の場合、SLS有効時は各部位が監視速度を超えないように動作します。Move命令などのCP動作の場合、SLS有効時はツール位置のみが監視速度を超えないように動作します。その他の部位がSLS監視速度を超過してエラーとなる場合、SpeedS、SpeedFactor等を利用して速度を制限してください。

SF_LimitSpeedS使用例

SLS_1の速度調整値を1500mm/sに設定する方法

```
SF_LimitSpeedS SLS_1, 1500
```

SLS_2の速度調整値を表示する方法（コマンドウィンドウを使用）

```
> SF_LimitSpeedS SLS_2  
SLS_2: 400
```

全てのSLS番号の速度調整値を表示する方法（コマンドウィンドウを使用）

```
> SF_LimitSpeedS  
SLS_1: 1500  
SLS_2: 400  
SLS_3: 750  
SLS_T: 250  
SLS_T2: 3000
```

3.22.19 SF_LimitSpeedS関数

安全速度監視(SLS)有効時に、速度を調整する機能の速度調整値を返します。

書式

SF_LimitSpeedS (SLS番号)

パラメーター

SLS番号

SLS番号を整数値、または以下に示す定数で指定します。

定数	値	内容
SLS_1	1	SLS_1有効時の速度調整値を返します
SLS_2	2	SLS_2有効時の速度調整値を返します
SLS_3	3	SLS_3有効時の速度調整値を返します
SLS_T	9	SLS_T有効時の速度調整値を返します
SLS_T2	10	SLS_T2有効時の速度調整値を返します

戻り値

指定したSLS番号の速度調整値[mm/s]を返します。

解説

SLS有効時に速度調整する機能の指定したSLS番号の速度調整値[mm/s]を返します。

SF_LimitSpeedS関数使用例

```
Integer i
i = SF_LimitSpeedS(SLS_1)
Print "SLS_1 limit speed is ", i
```

3.22.20 SF_LimitSpeedSEnable

安全速度監視(SLS)有効時に、速度を調整する機能についてOn/Offの設定、設定状態を表示します。

書式

SF_LimitSpeedSEnable [SLS番号 [, {On | Off}]]

パラメーター

SLS番号

SLS番号を整数値、または以下に示す定数で指定します。省略可能です。

定数	値	内容
SLS_1	1	SLS_1有効時の速度調整機能
SLS_2	2	SLS_2有効時の速度調整機能
SLS_3	3	SLS_3有効時の速度調整機能
SLS_T	9	SLS_T有効時の速度調整機能 *1
SLS_T2	10	SLS_T2有効時の速度調整機能 *1

*1 SLS_TとSLS_T2については、本コマンドで速度調整機能を設定することはできません。SLS_Tは操作モードがTEACHおよびTEST T1の時に速度調整機能がOnになります。SLS_T2は操作モードがTEST T2の時に速度調整機能がOnになります。詳細は以下のマニュアルを参照してください。

"ロボットコントローラー 安全機能マニュアル"

On | Off

速度調整機能のOn/Offを指定します。省略可能です。デフォルト設定は、Onです。

戻り値

なし

解説

指定したSLS番号が監視状態のときに速度を調整する機能のOn/Offを設定、表示します。Onにすると、SLSが有効な場合に、速度調整対象(*1)の速度を調整します。Offにすると、SLSが有効な場合でも速度を調整しません。On/Off設定に関わらず、SLSが無効な場合は、速度を調整しません。

第2パラメーターが省略されているときは、指定したSLS番号の速度調整状態(On/Off)を表示します。

全パラメーターが省略されているときは、全てのSLS番号の速度調整状態(On/Off)を表示します。

安全機能オプションが有効のときのみ使用できます。

*1 以下を参照してください。

[SF_LimitSpeedS](#)

SF_LimitSpeedSEnable使用例

SLS_1の速度調整機能を有効に設定する方法

```
SF_LimitSpeedSEnable SLS_1, On
```

SLS_2の速度調整機能の状態を表示する方法 (コマンドウィンドウを使用)

```
> SF_LimitSpeedSEnable SLS_2  
SLS_2: Off
```

全てのSLS番号の速度調整機能の状態を表示する方法 (コマンドウィンドウを使用)

```
> SF_LimitSpeedSEnable  
SLS_1: On  
SLS_2: Off  
SLS_3: Off  
SLS_T: On  
SLS_T2: On
```

注意

- SF_LimitSpeedSEnableは特異点回避, コンベヤートラッキング, 力制御機能と同時に使えません

SF_LimitSpeedSEnableのデフォルト設定はOnです。特異点回避, コンベヤートラッキング, 力制御機能の動作中に本機能で速度調整を行うと動作エラー (それぞれ4093, 4403, 5830) になります。また、SF_LimitSpeedSEnableがOnの場合、コンベヤ動作開始時に4500エラーが発生します。

これらの動作でSLSを使う場合、動作開始前にSF_LimitSpeedSEnableをOffにしてください。

また、SF_PeakSpeedS, SF_RealSpeedSを使ってロボットアーム動作速度を計測し、SLS監視速度 (安全機能パラメーター) を超えないようにロボットアーム動作速度をSpeed, SpeedFactor, SpeedSなどを使用して制御してください。

- CP動作の場合、SF_LimitSpeedSで速度調整される部位は現在選択されているツール位置です

安全機能マネージャーのTCPオフセットで指定した位置は、Toolコマンドに自動設定されません。Toolコマンドを利用し、適切なTCP位置を設定してください。

- 本機能は安全速度監視(SLS)の監視速度を超えないことを保証するものではありません

SLS監視速度を超過する場合は、SF_PeakSpeedS, SF_RealSpeedSを使ってロボットアーム動作速度を計測し、SLS監視速度を超えないようにロボットアーム動作速度をSpeed, SpeedFactor, SpeedSなどを使用して制御してください。

3.22.21 SF_LimitSpeedSEnable関数

安全速度監視(SLS)有効時に速度を調整する機能の状態を返します。

書式

SF_LimitSpeedSEnable(SLS番号)

パラメーター

SLS番号

SLS番号を整数値、または以下に示す定数で指定します。

定数	値	内容
SLS_1	1	SLS_1有効時の速度調整機能
SLS_2	2	SLS_2有効時の速度調整機能
SLS_3	3	SLS_3有効時の速度調整機能
SLS_T	9	SLS_T有効時の速度調整機能
SLS_T2	10	SLS_T2有効時の速度調整機能

戻り値

- 指定したSLS番号の速度調整機能がOnの場合、1を返します。
- 指定したSLS番号の速度調整機能がOffの場合、0を返します。

解説

指定したSLS番号が監視状態のときに速度を調整する機能の状態を返します。

SF_LimitSpeedSEnable関数使用例

```
If SF_LimitSpeedSEnable(SLS_1) = 0 Then
Print "SLS_1 linked speed adjustment function is disabled."
Endif
```

3.22.22 SF_PeakSpeedS

速度監視点のピーク速度値を表示します。

書式

SF_PeakSpeedS [速度監視点番号]

パラメーター

速度監視点番号

速度監視点番号を整数値 (1～4) で指定します。省略可能です。

- 1: ハンド
- 2: 肘
- 3: 手首
- 4: 肩

戻り値

なし

解説

指定した速度監視点のピーク速度値[mm/s]を表示します。

パラメーターが省略されているときは、すべての速度監視点のピーク速度値を表示します。

ハンドの速度は安全機能マネージャーで設定したTCPオフセット位置での速度です。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_PeakSpeedS使用例

ハンドのピーク速度値を表示する方法 (コマンドウィンドウを使用)

```
> SF_PeakSpeedS 1
250
```

すべての速度監視点のピーク速度値を表示する方法 (コマンドウィンドウを使用)

```
> SF_PeakSpeedS
250      150
100      50
```

表示順番は以下の通りです。

ハンド	肘
手首	肩

3.22.23 SF_PeakSpeedS関数

速度監視点のピーク速度を返します。

書式

SF_PeakSpeedS (速度監視点番号)

パラメーター

速度監視点番号

速度監視点番号を整数値 (1～4) で指定します。省略可能です。

- 1: ハンド
- 2: 肘
- 3: 手首
- 4: 肩

戻り値

ピーク速度[mm/s]を返します。

解説

指定した速度監視点のピーク速度を返します。

ハンドの速度は安全機能マネージャーで設定したTCPオフセット位置での速度です。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_PeakSpeedS関数使用例

```
Print "Hand peak speed is ", SF_PeakSpeedS (1)
```

3.22.24 SF_PeakSpeedSClear

速度監視点のピーク速度値をクリアし、初期化します。

書式

SF_PeakSpeedSClear [速度監視点番号1 [, 速度監視点番号2]]

パラメーター

速度監視点番号1

1つ目の速度監視点番号を整数値 (1～4) で指定します。省略可能です。

速度監視点番号2

2つ目の速度監視点番号を整数値 (1～4) で指定します。省略可能です。

- 1: ハンド
- 2: 肘
- 3: 手首
- 4: 肩

戻り値

なし

解説

指定した速度監視点のピーク速度値をクリア(初期化)します。

パラメーターを指定しない場合は、全ての速度監視点のピーク速度値がクリアされます。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_PeakSpeedSClear使用例

ハンドのピーク速度値をクリアする方法

```
SF_PeakSpeedSClear 1
```

全ての速度監視点のピーク速度値をクリアする方法

```
SF_PeakSpeedSClear
```

3.22.25 SF_RealSpeedS

速度監視点の現在速度を表示します。

書式

SF_RealSpeedS [速度監視点番号]

パラメーター

速度監視点番号

速度監視点番号を整数値 (1～4) で指定します。省略可能です。

- 1: ハンド
- 2: 肘
- 3: 手首
- 4: 肩

戻り値

なし

解説

指定した速度監視点の現在速度[mm/s]を表示します。

ハンドの速度は安全機能マネージャーで設定したTCPオフセット位置での速度です。

パラメーターが省略されているときは、全ての速度監視点の現在速度を表示します。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_RealSpeedS使用例

ハンドの現在速度を表示する方法 (コマンドウィンドウを使用)

```
> SF_RealSpeedS 1
250
```

全ての速度監視点の現在速度を表示する方法 (コマンドウィンドウを使用)

```
> SF_RealSpeedS
250      150
100      50
```

表示順番は以下の通りです。

ハンド	肘
手首	肩

3.22.26 SF_RealSpeedS関数

速度監視点の現在速度を返します。

書式

SF_RealSpeedS(速度監視点番号)

パラメーター

速度監視点番号

速度監視点番号を整数値 (1～4) で指定します。省略可能です。

- 1: ハンド
- 2: 肘
- 3: 手首
- 4: 肩

戻り値

現在速度[mm/s]を返します。

解説

指定した速度監視点の現在速度[mm/s]を返します。

ハンドの速度は安全機能マネージャーで設定したTCPオフセット位置での速度です。

本コマンドは、Safety基板を搭載したコントローラーで使用できます。

SF_RealSpeedS関数使用例

```
Print "Hand real speed is ", SF_RealSpeedS (1)
```

3.22.27 SF_SetIn

仮想安全I/Oの入力ポート(8ビット)を設定します。

書式

SF_SetIn ポート番号, 設定値

パラメーター

ポート番号

仮想安全I/Oの入力ポート番号を指定します。0を指定してください。

設定値

整数で仮想安全I/Oの入力ポートの設定値を指定します。

解説

仮想安全I/Oの入力ビットを同時に設定します。

仮想安全機能が有効時のみ動作します。

安全入力信号は負論理 (Active Low)です。

安全入力に0(オフ)が入力されると、安全入力に割り当てた機能が動作します。

現在の入力ポートの状態を取得するには EStopOn関数とSF_GetStatus関数を使用してください。

ポート番号0のビット割当

ビット	値	安全入力
0	&H1	SAFETY_IN1
1	&H2	SAFETY_IN2
2	&H4	SAFETY_IN3
3	&H8	SAFETY_IN4
4	&H10	SAFETY_IN5
5	-	1を指定してください
6	&H40	ESTOP_IN
7	-	1を指定してください

参照

[SF_SetSw](#), [SF_GetStatus関数](#), [SetIn](#), [SetSw](#), [EStopOn関数](#)

SF_SetIn使用例

SF_SetIn 0, &HFE ・ 安全入力の0ビット目 (SAFETY_IN1) をOFFに設定

3.22.28 SF_SetSw

仮想安全I/Oの入力を設定します。

書式

SF_SetSw ビット番号, 設定値

パラメーター

ビット番号

仮想安全I/Oの入力ビットを指定します。

設定値

0(オフ)か1(オン)の整数を指定します。

解説

仮想安全I/Oの入力ビットを設定します。

仮想安全機能が有効時のみ動作します。

安全入力信号は負論理 (Active Low)です。

安全入力に0(オフ)が入力されると、安全入力に割り当てた機能が動作します。

現在の入力ポートの状態を取得するには EStopOn関数とSF_GetStatus関数を使用してください。

ポート番号0のビット割当

ビット番号	安全入力
0	SAFETY_IN1
1	SAFETY_IN2
2	SAFETY_IN3
3	SAFETY_IN4
4	SAFETY_IN5
5	設定禁止
6	ESTOP_IN
7	設定禁止

参照

[SF_SetIn](#), [SF_GetStatus関数](#), [SetIn](#), [SetSw](#), [EStopOn関数](#)

SF_SetIn使用例

SF_SetSw 0, 0 '安全入力の0ビット目 (SAFETY_IN1) をOFFに設定

3.22.29 SFree

指定した関節をフリー状態にします。

書式

SFree [関節番号[, 関節番号,...]]

パラメーター

関節番号

関節の番号 (1～9の整数)を、式または数値で指定します。付加軸のS軸は8、T軸は9になります。

解説

SFreeは、指定された関節のモーターパワーを遮断します。このときの状態をフリージョイント状態とよびます。この命令は、ダイレクトティーチングや、指定された関節のみフリージョイント状態にして組み込みを行う時に使います。その関節のフリージョイント状態を解除するにはSLock命令、Motor On、またはMotor Offを実行します。

SFree命令を実行すると、ロボット制御パラメーターを初期化します。

詳細は、Motor Onを参照してください。

注意

- SFree で、システム設定の一部が初期化されます

SFreeは安全のため、ロボットの動作速度や加減速度にかかわるパラメーター (Speed, SpeedS, Accel, AccelSなど)およびLimZパラメーターを初期化します。詳細は、Motor Onを参照してください。

ファームウェアバージョン7.5.1.0より前では、Motor ON状態のみでSFree可能です。

SFreeは、ファームウェアバージョンによって、以下のように動作が変わります。

ファームウェア	SFreeの可否
7.5.1.0より前	モーターオン時のみ
7.5.1.0以降	モーターオン時、およびモーターオフ時

重要事項

- SFreeを水平多関節型ロボット (RSシリーズを含む)の第3関節、および第4関節に用いるとき

水平多関節型ロボット (RSシリーズを含む)の第3関節には電磁ブレーキがかかるので、SFreeを設定しても第3関節はすぐに動くわけではありません。手動で第3関節を動かすには、ロボットアーム上部にあるブレーキ解除スイッチを押し続ける必要があります。

また、第4関節にブレーキがある機種があります。第4関節にブレーキがある機種の場合、第4関節に電磁ブレーキがかかるので、SFreeを設定しても第4関節はすぐに動きません。手動で第4関節を動かすには、ロボットアーム上部にあるブレーキ解除スイッチを押しながら動かします。

- SFreeを垂直6軸型ロボット (Nシリーズを含む)に用いることはできません

垂直6軸型ロボット (Nシリーズを含む)でSFreeを行った場合、エラーが発生します。

手動でアームを動かす場合、Motor Off後にBrake Offで電磁ブレーキを解除してください。

- 関節がフリージョイント状態にあるときに動作コマンドを実行する

関節がフリージョイント状態にあるときに動作コマンドを実行すると、コントローラーがデフォルト状態ではエラーになります。1関節でもフリージョイント状態にあるときに動作命令を行うには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスをチェックします。

- コンベヤートラッキング中にSFreeを使用しないでください

コンベヤートラッキング中にSFreeを使用した場合、エラー5057,5058などが発生します。コマンド Cnv_AbortTrackなどでコンベヤートラッキングを終了させた後、SFreeを使用してください。

参照

[Brake](#), [LimZ](#), [Motor](#), [SFree関数](#), [SLock](#)

SFree使用例

SFree命令の簡単な使用例です。この例で作動するには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスがチェックされていなければいけません。

```
Function GoPick
  Speed pickSpeed
  SFree 1, 2    'J1とJ2をフリージョイント状態にして
パーツ取り付けのためZとU関節を動かす
  Go pick
  SLock 1, 2    'J1とJ2のフリージョイント状態を解除する
End
```


3.22.30 SFree関数

指定した関節のフリーズジョイント状態を返します。

書式

SFree(関節番号)

パラメーター

関節番号

フリーズジョイント状態をチェックする関節の番号 (整数)を、数値または式で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

- 1: フリーズジョイント状態
- 0: 非フリーズジョイント状態

参照

[SFree](#)

SFree関数使用例

```
If SFree(1) = 1 Then
  Print "Joint 1 is free"
EndIf
```

3.22.31 Sgn関数

数値の符号を返します。

書式

Sgn(演算数)

パラメーター

演算数

数値を指定します。

戻り値

- 1: 演算数は正の値
- 0: 演算数は0
- -1: 演算数は負の値

解説

Sgn関数は、数値の符号を返します。

参照

[Abs関数](#), [And 演算子](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Mod 演算子](#), [Or 演算子](#), [Not 演算子](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#), [Xor 演算子](#)

Sgn関数使用例

次の例はコマンドウィンドウでSgnを使った例です。

```
>print sgn(123)
1
>print sgn(-123)
-1
>
```

3.22.32 Short

Short型の変数を宣言します。(2バイト整数型変数)

書式

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

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

Shortは整数型変数の宣言をします。整数型変数の範囲は-32768～32767です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

Short使用例

次はShortを使って整数型の変数を宣言するプログラム例です。

```
Function shorttest
  Short A(10)           'Short型の1次元配列
  Short B(10, 10)       'Short型の2次元配列
  Short C(5, 5, 5)      'Short型の3次元配列
  Short var1, arrayvar(10)
  Integer i
  Print "Please enter an Integer Number"
  Input var1
  Print "The Integer variable var1 = ", var1
  For i = 1 To 5
    Print "Please enter an Integer Number"
    Input arrayvar(i)
    Print "Value Entered was ", arrayvar(i)
  Next i
End
```

3.22.33 Signal

WaitSig命令を実行中のタスクに信号を送ります。

書式

Signal シグナル番号

パラメーター

シグナル番号

0～63の、送信するシグナル番号を指定します。

解説

Signalはマルチタスクの同期をとります。WaitSig実行前のSignalは無視されます。

参照

[WaitSig](#)

Signal使用例

```
Function Main
  Xqt 2, SubTask
  Call InitSys
  Signal 1

Fend

Function SubTask
  WaitSig 1

Fend
```

3.22.34 SimGet

シミュレーターの各種オブジェクトのプロパティの設定値を取得します。

書式

SimGet Object.Property, Var

SimGet Robot.Hand.Propoerty, Var

パラメーター

Object

プロパティの値を取得するオブジェクト名を示す文字列変数

Robot

"Hand"で指定されたハンドが取り付けられているロボット名を示す文字列変数

Hand

プロパティの値を取得するハンド名を示す文字列変数

Property

値を取得するプロパティの名前。プロパティについては後述します。

Var

返される値を示す変数

解説

この命令は、シミュレーターにおける各種オブジェクトのプロパティの設定値を取得するときに使用します。

以下に示すプロパティを指定することで、オブジェクトの設定値を取得することができます。

プロパティ	説明	単位	データ タイプ	戻り値
PositionX	X座標位置を取得します	ミリメートル(mm)	Double	
PositionY	Y座標位置を取得します	ミリメートル(mm)	Double	
PositionZ	Z座標位置を取得します	ミリメートル(mm)	Double	
RotationX	X軸回転角度を取得します	度 (degree)	Double	
RotationY	Y軸回転角度を取得します	度 (degree)	Double	
RotationZ	Z軸回転角度を取得します	度 (degree)	Double	
CollisionCheck	衝突検出の有効・無効を取得します	-	Boolean	TrueもしくはFalse
CollisionCheckSelf	ロボットの自己衝突検出の有効/無効を取得します	-	Boolean	TrueもしくはFalse

プロパティ	説明	単位	データ タイプ	戻り値
Visible	表示/非表示の状態を取得します	-	Boolean	TrueもしくはFalse
Type	オブジェクトのタイプを取得します	-	Integer	Layout: 0 Part: 1 Mounted Device: 3
HalfSizeX	BoxオブジェクトのX方向の長さを取得します	ミリメートル(mm)	Double	
HalfSizeY	BoxオブジェクトのY方向の長さを取得します	ミリメートル(mm)	Double	
HalfSizeZ	BoxオブジェクトのZ方向の長さを取得します	ミリメートル(mm)	Double	
HalfSizeHeight	Planeオブジェクトの長さを取得します	ミリメートル(mm)	Double	
HalfSizeWidth	Planeオブジェクトの幅を取得します	ミリメートル(mm)	Double	
PlaneType	Planeオブジェクトのタイプを取得します	-	Integer	Horizontal: 0 Vertical: 1
Radius	SphereオブジェクトまたはCylinderオブジェクトの半径を取得します	ミリメートル(mm)	Double	
Height	Cylinderオブジェクトの高さを取得します	ミリメートル(mm)	Double	
Name	オブジェクト名を取得します		String	
Color	オブジェクトの表示色を取得します		String	色名もしくは16進カラーコード(ARGB)

以下の表に示す組み合わせでプロパティを取得することができます。

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	○	○	○	○	○	○	○	○
PositionY	○	○	○	○	○	○	○	○
PositionZ	○	○	○	○	○	○	○	○
RotationX	○	○	○	○	○	○	○	○
RotationY	○	○	○	○	○	○	○	○

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
RotationZ	○	○	○	○	○	○	○	○
CollisionCheck	○	○	○	○	○	○	○	○
CollisionCheckSelf	○	-	-	-	-	-	-	-
Visible	-	○	○	○	○	○	○	○
Type	-	-	○	○	○	○	○	-
HalfSizeX	-	-	○	-	-	-	-	-
HalfSizeY	-	-	○	-	-	-	-	-
HalfSizeZ	-	-	○	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	○	-	-
HalfSizeWidth	-	-	-	-	-	○	-	-
PlaneType	-	-	-	-	-	○	-	-
Radius	-	-	-	○	○	-	-	-
Height	-	-	-	-	○	-	-	-
Name	○	○	○	○	○	○	○	○
Color	-	-	○	○	○	○	-	-

参照

[SimSet](#)

SimGet使用例

'SBox_1オブジェクトのx座標値を取得する

```
Double boxPosX
```

```
SimGet SBox_1.PositionX, boxPosX
```

'SBox_1オブジェクトの表示・非表示の状態を取得する

```
Boolean boxVisible
```

```
SimGet SBox_1.Visible, boxVisible
```

'SBox_1オブジェクトのタイプを取得する

```
Integer boxType
```

```
SimGet SBox_1.Type, boxType
```

3.22.35 SimSet

シミュレーターの各種オブジェクトのプロパティの設定をします。また、ロボットの動作とオブジェクトの操作、シミュレーターの設定に関する操作をします。

書式

(1) オブジェクトのプロパティ設定

```
SimSet Object.Property, Value  
SimSet Robot.Hand.Property, Value
```

(2) ロボットの動作設定 (PickおよびPlace)

```
SimSet Robot.Pick, Object [,Tool]  
SimSet Robot.Place, Object
```

(3) オブジェクトの操作設定 (親オブジェクトの指定)

```
SimSet Object.SetParent [, ParentObject]
```

(4) シミュレーターの設定 (衝突検出のリセット)

```
SimSet ResetCollision
```

パラメーター

(1) オブジェクトのプロパティ設定

Object

プロパティの値を設定するオブジェクト名を示す文字列変数

Robot

"Hand"で指定されたハンドが取り付けられているロボット名を示す文字列変数

Hand

プロパティの値を設定するハンド名を示す文字列変数

Property

新たに値を設定するプロパティの名前。プロパティについては後述します。

Value

新しい値の式。データタイプはプロパティによります。

(2) ロボット動作設定 (PickおよびPlace)

Robot

PickもしくはPlaceするロボット名を示す文字列変数

Object

PickもしくはPlaceされるオブジェクト名を示す文字列変数

Tool

Pick時に利用するTool番号を示す式

(3) オブジェクトの操作設定 (親オブジェクトの指定)

Object

親オブジェクトを設定するオブジェクト名を示す文字列変数

ParentObject

親オブジェクト名を示す文字列変数

解説

この命令は、シミュレーターにおける各種オブジェクトのプロパティ設定や操作、ロボット動作、シミュレーター設定の変更をするときに使用します。

(1) オブジェクトのプロパティ設定

以下に示すプロパティを指定することで、オブジェクトの設定をすることができます。

プロパティ	説明	単位	データ タイプ	設定値
PositionX	X座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000
PositionY	Y座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000
PositionZ	Z座標位置を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: -100000
RotationX	X軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
RotationY	Y軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
RotationZ	Z軸回転角度を設定します	度 (degree)	Double	最大値: 360 最小値: -360
CollisionCheck	衝突検出の有効・無効を設定します	-	Boolean	TrueもしくはFalse
CollisionCheckSelf	ロボットの自己衝突検出の有効・無効を設定します	-	Boolean	TrueもしくはFalse
Visible	表示・非表示を設定します	-	Boolean	TrueもしくはFalse
HalfSizeX	BoxオブジェクトのX方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeY	BoxオブジェクトのY方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeZ	BoxオブジェクトのZ方向の長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeHeight	Planeオブジェクトの長さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
HalfSizeWidth	Planeオブジェクトの幅を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
PlaneType	Planeオブジェクトのタイプを設定します	-	Integer	Horizontal: 0 Vertical: 1

プロパティ	説明	単位	データ タイプ	設定値
Radius	SphereオブジェクトまたはCylinderオブジェクトの半径を設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
Height	Cylinderオブジェクトの高さを設定します	ミリメートル(mm)	Double	最大値: 100000 最小値: 0.001
Name	オブジェクト名を設定します		String	
Color	オブジェクトの表示色を設定します		String	色名もしくは16進カラーコード(ARGB)

以下の表に示す組み合わせでプロパティを設定することができます。

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
PositionX	○	○	○	○	○	○	○	○
PositionY	○	○	○	○	○	○	○	○
PositionZ	○	○	○	○	○	○	○	○
RotationX	○	○	○	○	○	○	○	○
RotationY	○	○	○	○	○	○	○	○
RotationZ	○	○	○	○	○	○	○	○
CollisionCheck	○	○	○	○	○	○	○	○
CollisionCheckSelf	○	-	-	-	-	-	-	-
Visible	-	○	○	○	○	○	○	○
HalfSizeX	-	-	○	-	-	-	-	-
HalfSizeY	-	-	○	-	-	-	-	-
HalfSizeZ	-	-	○	-	-	-	-	-
HalfSizeHeight	-	-	-	-	-	○	-	-
HalfSizeWidth	-	-	-	-	-	○	-	-
PlaneType	-	-	-	-	-	○	-	-
Radius	-	-	-	○	○	-	-	-
Height	-	-	-	-	○	-	-	-
Name	○	○	○	○	○	○	○	○

プロパティ	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Color	-	-	○	○	○	○	-	-

(2) ロボットの動作設定 (PickおよびPlace)

以下に示すロボットの動作を設定することができます。

■ Pick

"Robot"で指定されたロボットが、"Object"で指定されたオブジェクトを把持する動作を行います。

把持されたオブジェクトは、そのロボットのパーツとして登録されることになります。また、"Tool"に任意のツール番号を指定することで、指定されたツールを利用して把持動作を行います。"Tool"の指定が省略された場合は、Tool0を利用して把持動作を行います。

すでにパーツとして登録されているオブジェクト、もしくはアーム取付機器として設定されているオブジェクトを把持することはできません。また、カメラについても把持することはできません。

■ Place

"Robot"で指定されたロボットが、"Object"で指定されたオブジェクトを配置する動作を行います。配置されたオブジェクトは、そのロボットのパーツとしての登録が解除されます。

すでにパーツとしての登録が解除されているオブジェクトは配置することはできません。

以下の表に示す組み合わせで、オブジェクトの把持もしくは配置をすることができます。

動作	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Pick	-	-	○	○	○	○	○	-
Place	-	-	○	○	○	○	○	-

(3) オブジェクトの操作設定 (親オブジェクトの指定)

以下に示すオブジェクトの操作を設定することができます。

■ SetParent

"Object"で指定されたオブジェクトに対して、"ParentObject"で指定されたオブジェクトを親オブジェクトとして設定します。"ParentObject"は省略することができます。この場合、"Object"で指定されたオブジェクトが親オブジェクトとなります。例えば、"Object"で指定されたオブジェクトが、何らかのオブジェクトの子オブジェクトであった場合には、子オブジェクトとしての設定が解除されることになります。

また、"Object"で指定されたオブジェクトがパーツもしくはアーム取付機器として設定されているオブジェクトの場合は、親オブジェクトの指定をすることはできません。

SetParentを指定可能なオブジェクトは以下の表になります。カメラオブジェクトについては、固定カメラとして設定されているオブジェクトのみSetParentを利用することができます。

操作	オブジェクト							
	Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
SetParent	-	-	○	○	○	○	○	○

また、以下に示す表の組み合わせでSetParentを利用することができます。

		ParentObject (親オブジェクト)							
		Robot	Hand	Box	Sphere	Cylinder	Plane	CAD	Camera
Object (子オブジェクト)	Robot	-	-	-	-	-	-	-	-
	Hand	-	-	-	-	-	-	-	-
	Box	-	-	○	○	○	○	○	○
	Sphere	-	-	○	○	○	○	○	○
	Cylinder	-	-	○	○	○	○	○	○
	Plane	-	-	○	○	○	○	○	○
	CAD	-	-	○	○	○	○	○	○
	Camera	-	-	○	○	○	○	○	○

(4)シミュレータの設定 (衝突検出のリセット)以下に示すシミュレータ設定の変更ができます。

■ ResetCollision

衝突検出のリセットを行います。ResetCollisionを実行後に、ロボットとオブジェクトの間に衝突が発生していなければ、衝突状態が解除され、シミュレータの3D表示が更新されます。ロボットとオブジェクトの間に衝突が発生している場合は、衝突状態は解除されず、シミュレータの3D表示も更新されません。

参照

[SimGet](#)

SimSet使用例

```
'SBox_1オブジェクトのX座標値を100.0mmに設定する
```

```
SimSet SBox_1.PositionX, 100.0
```

```
'Robot1でSBox_1をTool1で把持する
```

```
SimSet Robot1.Pick, SBox_1, 1
```

```
'Robot1で把持しているSBox_1を配置する
```

```
SimSet Robot1.Place, SBox_1
```

```
'SBox_1の親オブジェクトにCAD_1を設定する
```

```
SimSet SBox_1.SetParent, CAD_1
```

```
'SBox_1を親オブジェクトにする
```

```
SimSet SBox_1.SetParent
```

```
'衝突検出をリセットする
```

```
SimSet ResetCollision
```

3.22.36 Sin関数

指定した数値のサイン値を返します。

書式

Sin(角度)

パラメーター

角度

角度を実数値で指定します。

戻り値

指定した角度を表わすサイン値を数値で返します。

解説

指定した角度 (ラジアン) のサイン値を返します。パラメーターの単位はラジアンでなければなりません。

戻り値の範囲は-1～1です。

角度をラジアンに変換するためにはDegToRad関数を使います。

参照

[Abs関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Mod 演算子](#), [Not 演算子](#), [Sgn関数](#), [Sqr関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#)

Sin関数使用例

Sin関数を使った例です。

```
Function sintest
  Real x
  Print "Please enter a value in radians:"
  Input x
  Print "Sin of ", x, " is ", Sin(x)
Fend
```

3.22.37 SingularityAngle

特異姿勢通過機能に必要な特異点近傍角度を設定します。

書式

SingularityAngle {角度設定値}

パラメーター

角度設定値

垂直6軸型ロボット (Nシリーズを含む)の手首特異姿勢近傍を判断するための第5関節角度 (0.1以上の実数、単位はdeg)を式または数値で指定します。

結果

パラメーターを省略すると、現在のSingularityAngle設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。

デフォルト値は10 degが設定されています。特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より小さい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを変更する必要はありませんが、特異姿勢通過時にエラーが発生するようなとき、エラーを抑制できることがあります。

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

参照

[AvoidSingularity](#), [SingularityAngle関数](#), [SingularitySpeed](#)

SingularityAngle使用例

```
SingularityAngle 7.0 '特異姿勢近傍角度設定を7度にする
```

3.22.38 SingularityAngle関数

SingularityAngleの設定値を返します。

書式

SingularityAngle

戻り値

設定されている特異姿勢近傍角度 (単位: deg) を返します。

参照

[AvoidSingularity](#), [SingularityAngle](#), [SingularitySpeed](#), [SingularitySpeed](#)関数

SingularityAngle関数使用例

```
Real currSingularityAngle  
currSingularityAngle = SingularityAngle
```

3.22.39 SingularityDist

特異姿勢通過機能に必要な特異点近傍距離を設定します。

書式

SingularityDist {距離設定値}

パラメーター

距離設定値

垂直6軸型ロボット (Nシリーズを含む) およびRSシリーズの肩特異姿勢近傍を判断するためのP点と第1関節回転軸距離 (0以上の実数、単位はmm) を式または数値で指定します。

結果

パラメーターを省略すると、現在のSingularityDist設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。

デフォルト値は30 mmが設定されています。肩特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より小さい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを変更する必要はありませんが、特異姿勢通過時にエラーが発生するようなとき、エラーを抑制できることがあります。

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

参照

[AvoidSingularity](#), [SingularityAngle](#), [SingularityAngle関数](#), [SingularityDist関数](#), [SingularitySpeed](#), [SingularitySpeed関数](#)

SingularityDist使用例

```
SingularityDist 10.0 '特異姿勢近傍距離設定を10 mmにする
```


3.22.40 SingularityDist関数

SingularityDistの設定値を返します。

書式

SingularityDist

戻り値

設定されている特異姿勢近傍距離 (単位: mm)を返します。

参照

[SingularityDist](#), [AvoidSingularity](#), [SingularityAngle](#), [SingularityAngle関数](#), [SingularitySpeed](#), [SingularitySpeed関数](#)

SingularityDist関数使用例

```
Real currSingularityDist  
currSingularityDist = SingularityDist
```

3.22.41 SingularitySpeed

特異姿勢通過機能に必要な特異点近傍角速度を設定します。

書式

SingularitySpeed {角速度設定値}

パラメーター

角速度設定値

垂直6軸ロボット (Nシリーズを含む)の手首特異姿勢近傍を判断するための第4関節角速度の最大角速度に対する割合 (1以上100以下の整数、単位は%)を式または数値で指定します。

結果

パラメーターを省略すると、現在のSingularitySpeed設定値を表示します。

解説

この命令は、特異姿勢通過機能を使用するときのみ有効となります。

デフォルト値は10 %が設定されています。特異姿勢に接近したときに行う回避動作の開始位置を調整したい場合に使用します。デフォルト値より大きい値を設定すると、特異姿勢により近づいてから回避動作をはじめます。通常は本パラメーターを調整し、変更する必要はありませんが、特異姿勢通過時にエラーが発生するようとき、小さい値を設定してエラーを抑制できることがあります。

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

参照

[AvoidSingularity関数](#), [SingularityAngle](#), [SingularitySpeed](#)

SingularitySpeed使用例

```
SingularitySpeed 30 '特異姿勢近傍角速度の割合を30 %に設定する
```

3.22.42 SingularitySpeed関数

SingularitySpeedの設定値を返します。

書式

SingularitySpeed

戻り値

設定されている特異姿勢近傍角速度の最大角速度に対する割合 (単位: %)を返します。

参照

[SingularitySpeed](#), [SingularityAngle](#), [AvoidSingularity](#)

SingularitySpeed関数使用例

```
Real currSingularitySpeed  
currSingularitySpeed = SingularitySpeed
```

3.22.43 SLock

指定された関節のフリージョイント状態を解除します。

書式

SLock [関節番号[, 関節番号,...]]

パラメーター

関節番号

関節の番号 (1～9の整数)を式または数値で指定します。付加軸のS軸は8、T軸は9になります。

解説

SLock は、ダイレクトティーチングやワークの装着などのため、SFree命令でフリージョイント状態になっていた関節のフリージョイント状態を解除します。

関節番号を省略すると、全関節のフリージョイント状態を解除します。

第3関節のSLockを行うと、電磁ブレーキが解除されます。

SLock命令を実行すると、ロボット制御パラメーターを初期化します。詳細は、Motor Onを参照してください。

垂直6軸型ロボット (Nシリーズを含む)はSFree命令でフリージョイント状態にできません。また、SLockを実行した場合、エラーが発生します。

参照

[Brake](#), [LimZ](#), [Reset](#), [SFree](#)

SLock使用例

SLock 命令を使った例です。この例で作動するには、[セットアップ]メニューの[システム設定]-[コントローラー]-[環境設定]パネルの、[関節が非励磁状態での移動命令を許可]チェックボックスが有効にされていなければなりません。

```
Function test
.
.
.
SFree 1, 2      ' J1とJ2をフリージョイント状態にしてパール取りつけのためZとU関節を動かす
Go P1
SLock 1, 2      ' J1とJ2のフリージョイント状態を解除する
.
.
.
Fend
```

3.22.44 SoftCP

SoftCP動作モードを設定します。

書式

SoftCP { On | Off }

パラメーター

On | Off

- On: SoftCP動作モードを有効にします。
- Off: SoftCP動作モードを解除します。

解説

SoftCP動作モードは、高加減速度でCP動作を実行したときの振動的な動作を抑制します。

通常のCP動作は、軌跡追従性能や等速動作性能を重視する設定がされています。そのため、高加減速度で動作したとき振動的になることがあります。この振動を抑制するためにはSpeedSやAccelSといった加減速度設定を低く制限することが有効でした。しかし、アプリケーションによっては、高い軌跡追従性や等速性よりも、高加減速度でより振動が少ないCP動作を実行することが要求されます。SoftCP動作モードでは、CP動作時の軌跡追従性や等速性を通常のCP動作より抑えることで、高加減速度が設定されたCP動作の振動をより少なくします。

SoftCP動作モードは、以下のCP動作命令に有効です。

Move, BMove, TMove, Arc, Arc3, CVMove, Jump3CP

通常のCP動作によって振動が問題にならない場合や、軌跡追従性や等速性が重視されるアプリケーションではSoftCP動作モードを使用しないでください。

重要事項

- CP On時にCP動作とPTP動作を接続する場合

下記のようにCP動作とPTP動作を接続する場合、SoftCPを有効にしてください。有効にしない場合、動作によってはロボットから異音が発生することがあります。接続終了後、SoftCPを無効にしてください。

```
SoftCP On
Go P1 CP
Move P2
SoftCP Off
```

参照

[SoftCP関数](#)

SoftCP使用例

```
SoftCP On
Move P1
Move P2
SoftCP Off
```

3.22.45 SoftCP関数

SoftCP動作モードの状態を返します。

書式

SoftCP

戻り値

- 0 = SoftCP動作モードオフ
- 1 = SoftCP動作モードオン

参照

[SoftCP](#)

SoftCP関数使用例

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

3.22.46 Space\$関数

指定した数だけの半角スペース文字列を返します。

書式

Space\$(指定数)

パラメーター

指定数

戻り値として返すスペースの数を指定します。

戻り値

指定した数の半角スペース文字列

解説

Space\$は指定した数の半角スペース文字列を返します。Space\$は最大255文字の半角スペース（文字列変数の許容範囲）を返します。

Space\$ は、別の文字列の前, 後, 中間に半角スペースを加えるために使います。

参照

[Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Left\\$関数](#), [Len関数](#), [LSet\\$関数](#), [Mid\\$関数](#), [Right\\$関数](#), [RSet\\$関数](#), [Str\\$関数](#), [Val関数](#)

Space\$関数使用例

```
> Print "XYZ" + Space$(1) + "ABC"
XYZ ABC

> Print Space$(3) + "ABC"
   ABC

>
```

3.22.47 Speed

Go, Jump, Pulse命令などによるPTP動作の、速度設定と表示をします。

書式

(1) Speed 速度設定値 [, 退避速度, 接近速度]

(2) Speed

パラメーター

速度設定値

最大動作速度 (PTP動作)に対する割合 (1~100の整数、単位: %)を式または数値で指定します。

退避速度

Jump命令時の退避動作の速度 (1~100の整数、単位: %)を、式または数値で指定します。省略可能です。

Jumpコマンド時のみ設定可能です。

接近速度

Jump命令時の接近動作の速度 (1~100の整数、単位: %)を、式または数値で指定します。省略可能です。

Jumpコマンド時のみ設定可能です。

結果

パラメーターを省略すると、現在のSpeed設定値を表示します。

解説

Speed は、すべてのPTP動作命令について、速度を指定します。これにはGo, Jump, Pulseなどの動作命令に関する速度設定が含まれます。速度設定は、最大速度に対する割合 (%)を1~100の整数で指定します。"100"を指定すると、最大速度で動作します。

退避速度および接近速度は、Jumpコマンドのみに対して適用されます。省略すると、速度設定値が、退避速度と接近速度の設定値となります。

下記のいずれかの場合、Speed値は、初期化されます。

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

ローパワーモード時には、Speed設定値は、デフォルト値よりも低い値になります。コマンドウィンドウから、またはプログラム中で、それよりも高い値が入れられても、デフォルト値が設定されます。ハイパワーモード時には、動作速度は、Speedで設定された値になります。

より速い動作速度が必要な場合は、Power Highでハイパワーモードを設定し、安全扉を閉めます。安全扉が開いていると、Speed値は、デフォルト値に変更されます。

ローパワーモード時にSpeedが実行されると、メッセージが表示されます。下記の例では、ロボットがローパワーモードにあるので、Speedの設定値が "80"であっても、ロボットはデフォルト値 "5"で動作します。

```
> speed 80
> speed
Low Power Mode
    80
    80      80
>
```


参照[Accel](#), [Go](#), [Jump](#), [Power](#), [Pass](#), [Pulse](#), [SpeedS](#)**Speed使用例**

Speed は、コマンドウィンドウでも、プログラム中でも、用いることができます。下記に両方の場合の使用例を示します。

```
Function speedtst
  Integer slow, fast, i
  slow = 10
  fast = 100
  For i = 1 To 10
    Speed slow
    Go P0
    Go P1
    Speed fast
    Go P0
    Go P1
  Next i
Fend
```

コマンドウィンドウから Speed値を設定する例です。

```
> Speed 100,100,50      'z関節の降下速度を50%に設定
> Speed 50
> Speed
  Low Power State: Speed is limited to 5
    50
    50          50
>
```

3.22.48 Speed関数

現在設定されているPTP動作速度を返します。

書式

Speed [(パラメーター番号)]

パラメーター

パラメーター番号

下記に示す番号から、設定値番号を1つ指定します。省略した場合は1が指定されたとみなします。

- 1: PTP 動作速度
- 2: JUMP 退避速度
- 3: JUMP 接近速度

戻り値

1～100の整数値を返します。

参照

[Speed](#)

Speed関数使用例

```
Function SpeedEx
  Integer savSpeed

  savSpeed = Speed(1)
  Speed 50
  Go pick
  Speed savSpeed
Fend
```

3.22.49 SpeedFactor

ロボット動作の速度係数を設定、および設定値を返す。

書式

(1) SpeedFactor 速度比率

(2) SpeedFactor

パラメーター

速度比率

ロボット動作の速度比率を (1～100の整数、単位: %)を式または数値で指定します。

結果

パラメーターが省略されているときは、現在のSpeedFactor値を表示します。

解説

SpeedFactorは、コントローラーに設定されているすべてのロボット、すべての動作に掛かる速度係数です。通常SpeedFactorは100%が設定されており、各ロボット、各動作命令の速度はSpeed, SpeedS等で設定します。SpeedFactorはすべてのロボットのすべての動作に対し一定の速度比率で速度を変更したい場合に有効です。例えば、Speed 80%の動作は、速度比率50%を設定したとき、動作速度40%となります。

SpeedFactorはロボット動作の加減速度のバランスを考慮するため加速度についても同じ割合で変更します。

SpeedFactorは、オペレーターウィンドウの速度比率の設定と等価であり、連動して変化します。

SpeedFactorは、コントローラー起動時にデフォルト値100%に初期化されます。

参照

[SpeedFactor関数](#)

SpeedFactor使用例

```
Function main
  Motor On
  Power High
  SpeedFactor 80

  Speed 100; Accel 100,100
  Go P1          'Speed 80; Accel 80,80で動作する

  Speed 50; Accel 50,50
  Go P2          'Speed 40; Accel 40,40で動作する
Fend
```

3.22.50 SpeedFactor関数

SpeedFactor命令の設定値を返します。

書式

SpeedFactor

戻り値

SpeedFactor命令の設定値を整数値で返します。

参照

[SpeedFactor](#)

SpeedFactor関数使用例

```
Real savSpeedFactor

savSpeedFactor = SpeedFactor
SpeedFactor 80
Go P1
Go P2
SpeedFactor savSpeedFactor
```

3.22.51 SpeedR

ROTを使っているときのCP動作のツール姿勢変化速度の設定と表示をします。

書式

(1) SpeedR 速度設定値

(2) SpeedR

パラメーター

速度設定値

CP動作時のツール姿勢変化速度 (0.1以上の実数、単位はdeg/sec)を式または数値で指定します。

パラメーターの有効設定値: 0.1～1000

結果

パラメーターを省略すると、現在のSpeedR設定値を表示します。

解説

この命令は、Move, Arc, Arc3, BMove, TMove, Jump3CPの動作命令で、ROT修飾パラメーターを指定したとき有効です。また、SpeedRLimitation有効時は、ツール姿勢変化速度の上限値として機能します。

下記のいずれかの場合、SpeedR値は、デフォルト値 (低速)に初期化されます。

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

参照

[AccelR](#), [Arc](#), [Arc3](#), [BMove](#), [Jump3](#), [Jump3CP](#), [Power](#), [SpeedR関数](#), [TMove](#), [SpeedRLimitation](#)

SpeedR使用例

```
SpeedR 200
```

3.22.52 SpeedR関数

ツール姿勢変化速度を返します。

書式

SpeedR

戻り値

設定された速度を実数値 (単位: deg/sec)で返します。

参照

[AccelR](#), [SpeedR](#)

SpeedR関数使用例

```
Real currSpeedR  
currSpeedR = SpeedR
```

3.22.53 SpeedRLimitation

ツール姿勢変化速度による、動作速度制限の設定と表示をします。

書式

(1) SpeedRLimitation {On | Off}

(2) SpeedRLimitation

パラメーター

On | Off

On: ツール姿勢変化速度による動作速度制限を有効にします。

Off: ツール姿勢変化速度による動作速度制限を無効にします。

結果

パラメーターを省略すると、現在のSpeedR設定値を表示します。

解説

SpeedRLimitationは下記の命令で有効です。

Move, TMove, BMove, Arc, Arc3, Jump3CP

SpeedRLimitationを有効にすると、CP動作時のツール姿勢変化速度は設定したSpeedRを超えないように動作速度が制限されます。ツール姿勢変化速度がSpeedRを超えない場合は動作速度はSpeedSで設定されたままの速度で動作します。

SpeedRLimitationが無効にされている場合でも、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働いています。ツール姿勢変化速度を調整したい場合は、SpeedRLimitation機能を有効にして、適切なSpeedRを調整してください。

下記のいずれかの場合、SpeedRLimitationは、コントローラー設定で指定した設定状態(工場出荷時：Off)になります。

- コントローラー起動時
- Motor On実行
- Reset実行
- Reset, Reset Error実行
- 停止ボタン、Quit All実行などによるタスクの終了

注意

- SpeedRLimitation使用上の注意
コントローラーの環境設定により、コントローラー起動時にツール姿勢変化速度による動作速度制限をOnにできません。
- SpeedRの設定
SpeedRのデフォルト値は低速に設定されているため、SpeedRを適切に設定しない場合、姿勢変化をともなうCP動作が遅くなることになります。SpeedRLimitationを有効にする場合は、SpeedSとともにツール姿勢変化速度の上限となるSpeedRを適切に設定してください。

参照

[SpeedR](#), [SpeedS](#), [SpeedRLimitation関数](#), [Move](#), [TMove](#), [BMove](#), [Arc](#), [Arc3](#), [Jump3](#), [Jump3CP](#)

SpeedRLimitation使用例

```
SpeedR 50 'deg/s  
SpeedRLimitation On  
Move P1
```


3.22.54 SpeedRLimitation関数

ツール姿勢変化速度による動作速度制限の有効無効を返します。

書式

SpeedRLimitation

戻り値

0 = 無効

1 = 有効

SpeedRLimitation関数使用例

```
If SpeedRLimitation = On Then
    Print "SpeedRLimitation is On"
EndIf
```

3.22.55 SpeedS

Move, Arc, Arc3, Jump3, Jump3CPなどCP動作時のアームの速度の設定と表示を行います。

書式

(1) SpeedS 速度設定値 [, 退避速度, 接近速度]

(2) SpeedS

パラメーター

速度設定値

速度を式または数値 (整数、単位: mm/s)で指定します。

退避速度

Jump3の退避速度を示す実数または式 (単位: mm/s)を指定します。省略可能です。

接近速度

Jump3の接近速度を示す実数または式 (単位: mm/s)を指定します。省略可能です。

パラメーターの有効設定値:

- N2シリーズ以外: 0.1～2000
- N2シリーズ: 0.1～1120

結果

パラメーターを省略すると、現在のSpeedS値を表示します。

解説

SpeedSは、CP動作 (MoveおよびArc)命令時の速度を指定します。

SpeedS値は、ロボットの速度を指定した値です。指定する単位はmm/sです。デフォルト値は、ロボットの機種によって異なります。SpeedSのデフォルト値については、以下のマニュアルを参照してください。

"マニピュレーターマニュアル"

コントローラーの電源オン時は、この値が常に自動的に設定されます。

下記のいずれかの場合、SpeedS値は、初期化されます。

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

ローパワーモード時には、SpeedS値は、デフォルト値か設定値のどちらか低い方の速度が有効になります。より高い速度設定を、コマンドウィンドウから、またはプログラム中で指定すると、速度はデフォルト値に設定されます。ハイパワーモード時には、動作速度は、SpeedS設定値になります。

より速い動作速度が必要な場合は、Power Highでハイパワーモードを設定し、安全扉を閉めます。安全扉が開いていると、SpeedS値は、デフォルト値に変更されます。

SpeedSを変更しても動作速度が変わらない場合、SpeedRLimitationが有効になっている場合があります。Epson RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[姿勢回転速度による動作速度を抑制する]チェックボックスがチェックされていないか確認してください。チェックされている場合、姿勢回転速度がSpeedRに制限されている可能性があります。

参照

[AccelS](#), [Arc](#), [Arc3](#), [Jump3](#), [Jump3CP](#), [Move](#), [Speed](#), [SpeedRLimitation](#), [SpeedR](#)

SpeedS使用例

SpeedSは、コマンドウィンドウでも、プログラム中でも、いずれでも用いることができます。下記に両方の場合の使用例を示します。

```
Function speedtst
  Integer slow, fast, i
  slow = 50
  fast = 500
  For i = 1 To 10
    SpeedS slow
    Move P0
    Move P1
    SpeedS fast
    Move P0
    Move P1
  Next i
Fend
```

SpeedS は、コマンドウィンドウからも設定できます。

```
> speeds 1000
> speeds 500
> speed 30      'PTP動作速度を設定
> go p0         'PTP動作で移動
> speeds 100    '直線速度をmm/sで設定
> move P1      '直線で移動
```

3.22.56 SpeedS関数

SpeedS設定を返します。

書式

SpeedS [(パラメーター番号)]

パラメーター

パラメーター番号

返されるSpeedS値を、下記の中から整数または式で指定します。省略可能です。

- 1: CP速度
- 2: Jump3退避速度
- 3: Jump3接近速度

戻り値

実数値を返します。単位はmm/sです。

参照

[SpeedS](#)

SpeedS関数使用例

```
Real savSpeeds  
  
savSpeeds = SpeedS  
  
Print "Jump3 depart speed = ", SpeedS(2)
```

3.22.57 Sqr関数

平方根を返します。

書式

Sqr(数値)

パラメーター

数値

実数値または式で指定します。

戻り値

平方根を返します。

解説

平方根を返します。

起こりやすいエラー

- 負数演算子

数値が負数の場合、エラーが発生します。

参照

[Abs関数](#), [And 演算子](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Mod 演算子](#), [Not 演算子](#), [Or 演算子](#), [Sgn関数](#), [Sin関数](#), [Str\\$関数](#), [Tan関数](#), [Val関数](#), [Xor 演算子](#)

Sqr関数使用例

コマンドウィンドウからSqr関数を使用した例です。

```
>print sqr(2)
1.414214
>
```

Sqrを使用した簡単なプログラム例です。

```
Function sqrtest
  Real x
  Print "Please enter a numeric value:"
  Input x
  Print "The Square Root of ", x, " is ", Sqr(x)
Fend
```

3.22.58 ST関数

指定された付加軸の座標値をポイントデータとして返します。

書式

ST(s座標要素, t座標要素)

パラメーター

s座標要素

s軸の座標を実数値で指定します。

t座標要素

t軸の座標を実数値で指定します。

戻り値

指定された付加軸の座標値をポイントデータとして返します。

解説

付加軸(S軸, T軸)を使用する場合のみ本関数は使用します。

Go ST(10,20) というような使い方をした場合、付加軸は指定の座標値へ移動しますが、ロボットは動きません。ロボットも同時に動作させるようなケースでは Go XY(60,30,-50,45) : ST(10,20) のような指定をしてください。

詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - 付加軸"

参照

[XY関数](#)

ST関数使用例

```
P10 = ST(10, 20)
```

3.22.59 STM関数

参照ポイントを開始点としたときに動作時間が最も短くなるように指定ポイントの姿勢フラグを変換したポイントデータを返します。

書式

STM(ポイント指定 [, 参照ポイント指定 [, モード指定]])

パラメーター

ポイント指定

対象とするポイントデータを指定します。

参照ポイント指定

基準とするポイントデータを指定します。省略可能です。参照ポイント指定を省略した場合は現在位置(Here)を参照ポイントとします。

モード指定

省略可能です。

- RSシリーズ

STM_H_MODE_FASTEST: 参照ポイントから指定ポイントにGo動作した場合に、動作時間が最小となるように姿勢フラグを変更します。J1Flag、J2Flagおよび腕姿勢 (Handフラグ)が変更されます。X,Y,ZおよびU軸の値は変更されません。

動作時間が最小となる姿勢フラグが複数存在する場合は、姿勢フラグの変化が少ないものが選択されます。

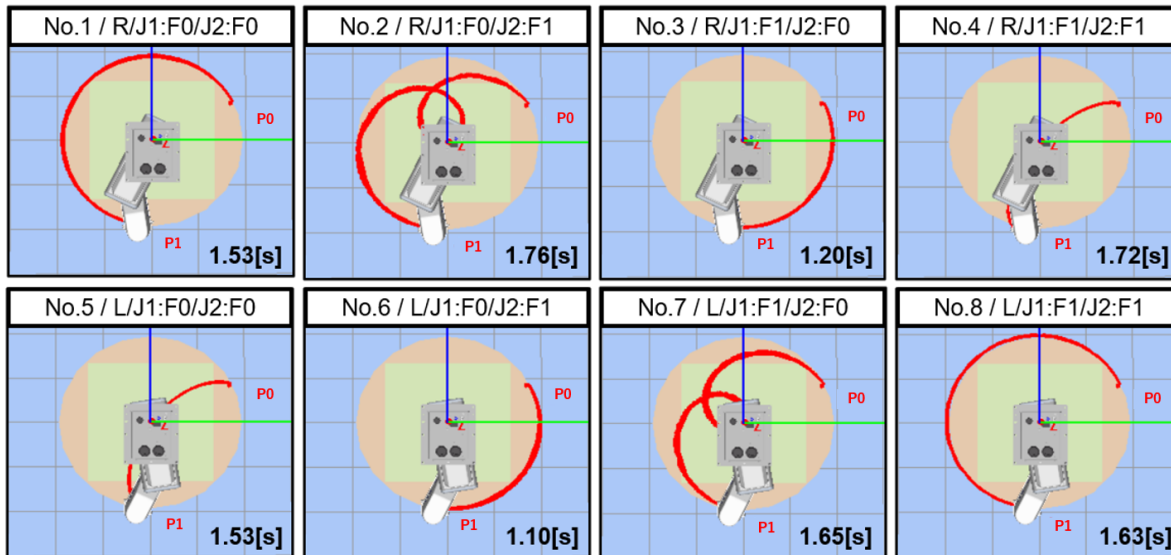
- その他の機種

STM関数はサポートしません。

解説

RSシリーズでは、パレットや相対オフセットなどのポイント演算で得られたポイントに動作するとき、姿勢フラグを正しく指定しないと、第1アーム、第2アームが意図しない方向へ回転することがあります。また、どの姿勢フラグを選択すれば動作時間が最小になるのか、実行しなければわかりません。STM関数はアームの意図しない回転を防止するとともに、動作時間が最小となるようにポイントデータの姿勢フラグを適切に変換します。これによりタクトタイムの短縮が期待できます。

下の図のポイントP0からポイントP1にPTP動作することを考えます。RSシリーズでは、J1Flag、J2Flagおよび腕姿勢の組み合わせがあるため、姿勢フラグの選び方は合計で8パターンあります。図中の赤線はポイントP0からそれぞれの姿勢フラグに対応するポイントP1に動作したときの動作軌跡を表しています。また、図の左下の数値は動作時間を表します。STM関数は複数ある姿勢フラグのうち動作時間が最小となるものにむかって動作します。下の図の例ではNo.6の左腕系、J1F0、J2F1の姿勢フラグが選ばれます。

P0:R/J1:F0/J2:F0

動作時間は、関数実行時に設定されているWeight、Inertia設定値およびSpeed,Accel設定をもとに計算されます。これらの設定値を変更した場合、STM関数による姿勢フラグの変換結果が変わる可能性があります。STM関数のテストを行う場合は、実環境での速度加速度設定を用いてください。

パワーモードの設定値、SpeedFactorの設定値、およびSLS速度調整値に関わらず、STM関数による姿勢フラグの変換結果は変わりません。パワーモードがLowの場合、パワー出力が制限される前のSpeedおよびAccelをもとに動作時間の計算を行い姿勢フラグを変換します。テストモードかAUTOモードかによって、姿勢フラグの変換結果は変わりません。

パルス動作範囲外の姿勢フラグは選択されません。

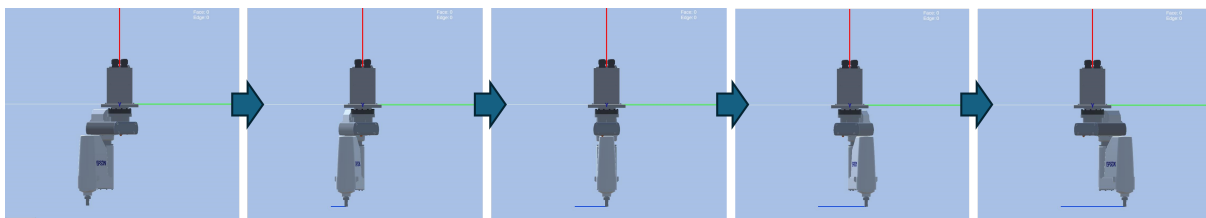
STM関数で返されるポイントのローカル番号は、"ポイント指定"と同じローカル番号になります。

注意

■ 配線、装置レイアウトについて

STM関数を使用した場合、動作前後で腕姿勢が切り替わることがあります。腕姿勢が切り替わる場合、アームを畳みこむような動作になるため、配線に注意してください。

また、右腕左腕系が切り替わっても周辺機器との干渉がないようにレイアウトしてください。下図は腕姿勢が切り替わった時の動作例です。


■ 姿勢フラグの違いによる誤差について

本コマンドは粗位置決めでの使用を想定しています。同一ポイントを右腕系と左腕系で位置決めした場合、ツール設定の誤差やキャリブレーションの誤差により、位置ずれが生じる場合があります。精密な位置合わせが要求される作業では本コマンドを使用せずティーチしたポイントを使用してください。

参照

[Pallet](#), [AutoSTM](#), [Power](#), [SpeedFactor](#), [Speed](#), [Accel](#), [Weight](#), [Inertia](#), [SF_LimitSpeedS](#)

STM関数使用例

以下のプログラムはRSシリーズでのSTM関数の使用例です。2点間をJump動作で往復します。

P2の腕姿勢は右腕系で定義されていますが、STM関数によって、動作時間が最短となる左腕系に変換されています。

```
Function STM_Example1
    P1 = XY(150, 0, 0, 0)
    P2 = XY(-150, 0, 0, 0)
    Motor On; Power High;
    Speed 20; Accel 20, 20
    Weight 1.0; Inertia 0.005;
    Do
        Jump P1
        Print Here
        Jump P2
        Print Here
        Jump P1
        Print Here
        Jump STM(P2)
        Print Here
    Loop
Fend
```

出力結果は以下のようになります。

```
X: 150.000 Y: 0.000 Z: 0.000 U: 0.000 V: 0.000 W: 0.000 /R /J2F0
/J1F0 /J1A(0.000) /0
X: -150.000 Y: 0.000 Z: 0.000 U: 0.000 V: 0.000 W: 0.000 /R /J2F0
/J1F0 /J1A(0.000) /0
X: 150.000 Y: 0.000 Z: 0.000 U: 0.000 V: 0.000 W: 0.000 /R /J2F0
/J1F0 /J1A(0.000) /0
X: -150.000 Y: 0.000 Z: 0.000 U: 0.000 V: 0.000 W: 0.000 /L /J2F0
/J1F1 /J1A(0.000) /0
```

3.22.60 StartMain

バックグラウンドタスクからメインファンクションを実行します。

本命令は、上級者向けの命令です。命令の仕様を十分理解したうえで使用してください。

書式

StartMain メインファンクション名

パラメーター

メインファンクション名

実行するメインファンクション名 (main～main63) を指定します。

解説

本命令をプログラムから実行するには、Epson RC+ の[セットアップ]-[システム設定]-[コントローラー]-[環境設定]の[アドバンスタスク制御コマンドを有効]チェックボックスがチェックされている必要があります。

バックグラウンドタスクからXqt命令でタスクを実行した場合、実行されたタスクもバックグラウンドタスクになります。本命令を使用すると、バックグラウンドタスクから、メインファンクションを通常タスクとして実行することができます。

すでにメインファンクションを実行済みの場合や、通常タスクから本命令を実行した場合はエラーが発生します。

注意

プログラムからStartMainコマンドを実行する場合、コマンドの仕様を理解して、システムとしてメインファンクション実行可能な条件が整っていることを確認してください。ループでコマンドを実行し続けるなど、使い方をあやまるとシステムの安全性を低下させる可能性があります。十分に注意してください。

参照

[Xqt](#)

StartMain使用例

```
Function bgmain
:
If Sw(StartMainSwitch) = On And Sw(ErrSwitch) = Off Then
  StartMain main
EndIf
:
Fend
```

3.22.61 Stat関数

コントローラーの状態を返します。

書式

Stat(アドレス)

パラメーター

アドレス

コントローラーの状態を示すアドレス (0～2の整数)を指定します。

戻り値

コントローラーの状態を示す4バイトの値を返します。(下記表を参照してください。)

解説

Stat 命令は、下記表に示されたデータを返します。各ビットの内容を参照してください。

アドレ ス	Bit		ビットがオンのとき示されるコントローラーの状態
0	0-15	&H1- &H8000	タスク1から16が 実行中 (Xqt)またはHalt状態
	16	&H10000	タスク実行中
	17	&H20000	ポーズ (一時停止)状態
	18	&H40000	エラー状態
	19	&H80000	TEACHモード
	20	&H100000	非常停止状態
	21	&H200000	ローパワーモード (Power Low)
	22	&H400000	安全扉入力が開
	23	&H800000	Enableスイッチがオン
	24	&H1000000	未定義
	25	&H2000000	未定義
	26	&H4000000	テストモード
	27	&H8000000	T2モード状態
	28-31		未定義

アドレス	Bit		ビットがオンのとき示されるコントローラーの状態
1	0	&H1	Jump...Senseステートメントの条件成立で目標座標上空停止の履歴。(この履歴は次のJumpステートメントが実行されると消去されます。)
	1	&H2	Go/Jump/Move...Till ステートメントの条件成立で動作途中停止した履歴。(この履歴は次のGo/Jump/Move...Tillステートメントが実行されると消去されます。)
	2	&H4	未定義
	3	&H8	Trap ステートメントの条件成立で動作途中停止した履歴
	4	&H10	Motor On 状態
	5	&H20	現在は、ホーム位置
	6	&H40	ローパワー状態
	7	&H80	未定義
	8	&H100	第4関節の励磁中
	9	&H200	第3関節の励磁中
	10	&H400	第2関節の励磁中
	11	&H800	第1関節の励磁中
	12	&H1000	第6関節の励磁中
	13	&H2000	第5関節の励磁中
	14	&H4000	第T関節の励磁中
	15	&H8000	第S関節の励磁中
	16	&H10000	第7関節の励磁中
	17-31		未定義
2	0-15	&H1- &H8000	タスク17から32が 実行中(Xqt)またはHalt状態

参照

[EStopOn関数](#), [TillOn関数](#), [PauseOn関数](#), [SafetyOn関数](#)

Stat関数使用例

```
Function StatDemo
```

```
Integer rbt1_sts
rbt1_sts = RShift((Stat(0) And &H070000), 16)
Select TRUE
    Case (rbt1_sts And &H01) = 1
```

```
        Print "Tasks are running"
    Case (rbt1_sts And &H02) = 2
        Print "Pause Output is ON"
    Case (rbt1_sts And &H04) = 4
        Print "Error Output is ON"
Send
Fend
```

3.22.62 Str\$関数

数値を文字列化して返す関数です。

書式

Str\$(数値)

パラメーター

数値

文字列に変換する数値を、式または直接数値で指定します。

戻り値

数値を文字列化して返します。

解説

Str\$ は、数値 (正負とも可能) を文字列化します。

参照

[Abs関数](#), [Asc関数](#), [Chr\\$関数](#), [InStr関数](#), [Int関数](#), [Left\\$関数](#), [Len関数](#), [Mid\\$関数](#), [Mod 演算子](#), [Right\\$関数](#), [Sgn関数](#), [Space\\$関数](#), [Val関数](#)

Str\$関数使用例

下記例は、いくつかの異なる数値を文字列化してスクリーンに表示します。

```
Function strttest
  Integer intvar
  Real realvar
  '
  intvar = -32767
  Print "intvar = ", Str$(intvar)
  '
  realvar = 567.9987
  Print "realvar = ", Str$(realvar)
  '
Fend
```

コマンドウィンドウから Str\$ 命令を行った結果です。

```
> Print Str$(999999999999999)
1.000000E+014

> Print Str$(25.999)
25.999
```

3.22.63 String

変数を文字列変数に宣言します。

書式

String 変数名\$[(配列の最大要素番号)] [,変数名\$[(配列変数の最大要素番号)]...]

パラメーター

変数名\$

文字列型と宣言する変数の名前を指定します。配列変数の最大要素番号配列変数の最大要素番号で3次元まで宣言できます。次の書式を使用します。省略可能です。

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

- ローカル変数: 200
- バックアップ変数(Global Preserve): 400
- グローバル変数とモジュール変数: 10,000

解説

String は、変数の文字列型宣言のために用います。文字列型変数は、最大255文字までです。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

- String 演算子

文字列変数には、下記の演算子を使うことができます。

- +: 複数の文字列変数をマージします。文字列変数への割当ステートメントやPrint命令に使えます。

```
Example: name$ = fname$ + " " + lname$
```

- =: 複数の文字列変数を比較します。大文字/小文字の違いを含め2つの文字列が完全に一致するとき、Trueが返されます。

```
Example: If temp1$ = "A" Then GoSub test
```

- <>: 複数の文字列変数を比較します。2つの文字列中、1つ以上の文字が異なるときにTrueを返します。

```
Example: If temp1$ <> "A" Then GoSub test
```

注意

- 変数名の最後には"\$" をつけてください

String 型の変数名は、最後に"\$" をつけてください。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [UByte](#), [UInt32](#), [UInt64](#), [UShort](#)

String使用例

```
String password$  
String A$(10)           'String型の1次元配列  
String B$(10, 10)       'String型の2次元配列
```

```
String C$(5, 5, 5)      'String型の3次元配列
```

```
Print "Enter password:"
```

```
Input password$
```

```
If UCase$(password$) = "EPSON" Then
```

```
    Call RunMaintenance
```

```
Else
```

```
    Print "Password invalid!"
```

```
EndIf
```


3.22.64 Sw関数

指定された入力ビットの状態を返す関数です。

書式

Sw(ビット番号)

パラメーター

ビット番号

I/Oの入力ビットを、整数または式で指定します。

戻り値

指定された入力が入オンのとき"1"を、オフのとき"0"を返します。

解説

Swは、I/O入力の状態チェックを行います。Swは、最も一般的には、I/Oで動くローダー、コンベヤー、グリッパ、その他の周辺装置に接続するセンサーを、チェックするために使われます。Sw命令でチェックされる入力状態には、"1"か"0"があります。それぞれ、デバイスがオン (1)かオフ (0)かを示します。

参照

[In関数](#), [InBCD関数](#), [MemOn](#), [MemOff](#), [MemSw関数](#), [Off](#), [On](#), [OpBCD](#), [Oport関数](#), [Out](#), [Wait](#)

Sw関数使用例

下記例は、入力ビット5をチェックして、その結果に基づいてプログラムを枝分かれさせる例です。より明確にするため、"1"の代わりに"On"が使われています。

```
Function main
  Integer i, feed5Ready
  feed5Ready = Sw(5)
  'Feederの準備ができているか確認
  If feed5Ready = On Then
    Call mkpart1
  Else
    Print "Feeder #5 is not ready. Please reset and"
    Print "then restart program"
  EndIf
Fend
```

下記は、コマンドウィンドウからの簡単な操作例です。

```
> print sw(5)
1
>
```

3.22.65 SyncLock

相互排他ロックを用いて複数のタスクを同期させます。

書式

SyncLock シグナル番号 [, タイムアウト]

パラメーター

シグナル番号

受信する信号番号 (0～63の整数)を、式または数値で指定します。

タイムアウト

ロックまで待つ最長待機時間を、式または数値 (実数)で指定します。省略可能です。

解説

SyncLock は、共通のリソースを1度に1タスクだけが使うよう、同期ロックするために使います。あるタスクが共通リソースを使い終わったら、SyncUnlockを呼び出してロック解除し、他のタスクがそのリソースを使えるようにします。

1つのタスクがロック解除することができるのは、あらかじめ自分がロックしたsyncIDだけです。

タスクは、ロック解除するためにSyncUnlockを実行しなくてはなりません。タスクが終了された場合は、そのタスクが行ったロックは解除されます。

同じシグナル番号に対して、2回連続でSyncLockを実行するとエラーになります。

タイムアウトパラメーターが指定された場合は、Tw関数を使って、ロックが成功したかどうかをチェックできるようにしてください。

注意

EPSON RC+ 5.0ではタスクが終了しても自動的にロックは解除されませんが、EPSON RC+ 6.0, RC+ 7.0, Epson RC+ 8.0では自動的に解除されます。

参照

[Signal](#), [SyncLock](#), [TW関数](#), [Wait](#), [WaitPos](#)

SyncLock使用例

下記例では、SyncLockとSyncUnlockを使って、1度に1タスクだけが、ログファイルにメッセージを書き込めるように設定しています。

```
Function Main
    Xqt Func1
    Xqt Func2
Fend

Function Func1
    Long count
    Do
        Wait .5
        count = count + 1
        LogMsg "Msg from Func1, " + Str$(count)
    Loop
Fend

Function Func2
```

```
Long count
Do
    Wait .5
    count = count + 1
    LogMsg "Msg from Func2, " + Str$(count)
Loop
Fend

Function LogMsg(msg$ As String)
    SyncLock 1
    OpenCom #1
    Print #1, msg$
    CloseCom #1
    SyncUnlock 1
Fend
```

下記は、オプションのタイムアウトを使ったSyncLockの例です。Twを使ってロックが成功したかどうかをチェックしています。タイムアウトを使うことで、リソースをロックするのを待っている間、他のプログラムを実行することができます。

```
Function MySyncLock(syncID As Integer)
    Do
        SyncLock syncID, .5
        If Tw = 0 Then
            Exit Function
        EndIf
        If Sw(1) = On Then
            Off 1
        EndIf
    Loop
Fend
```

3.22.66 SyncUnlock

あらかじめSyncLockでロックされた、シグナル番号のロックを解除します。

書式

SyncUnlock シグナル番号

パラメーター

シグナル番号

受信する信号番号 (1～63の整数)を、式または数値で指定します。

解説

SyncUnlock は、あらかじめSyncLock でロックされたシグナル番号のロックを解除します。

あらかじめロックされたシグナル番号でなければ、ロック解除することはできません。

参照

[Signal](#), [SyncLock](#), [Wait](#), [WaitPos](#)

SyncUnlock使用例

```
Function Main
    Xqt task
    Xqt task
    Xqt task
    Xqt task
Fend

Function task
    Do
        SyncLock 1
        Print "resource 1 is locked by task", MyTask
        Wait .5
        SyncUnlock 1
    Loop
Fend
```

3.22.67 SyncRobots

動作予約中のロボットの動作を開始します。

書式

SyncRobots ロボット番号 [, ロボット番号] [...]

SyncRobots All

パラメーター

ロボット番号

動作を開始するロボット番号を式または数値 (整数) で指定します。

All

動作予約中のすべてのロボットを指定します。

解説

SyncRobots は、各動作コマンドにサポートされているSYNCパラメーターを使って動作予約されたロボットの動作を開始するために使います。SyncRobotsで指定されたロボットは同じタイミングで動作が開始します。通常のマルチタスクプログラムのI/O信号のイベント待ちを使った同期よりも、タスクスイッチングの影響がないため、より正確にロボットの動作開始を同期できます。

動作予約されていないロボット番号を指定した場合はエラーになります。

参照

[SyncRobots関数](#)

SyncRobots使用例

下記例では、動作コマンドのSYNCパラメーターとSyncRobotsを使って、2台のロボットを同時に動作開始します。

```
Function Main
  Xqt Func1
  Xqt Func2
  Do
    Wait 0.1
    If (SyncRobots And &H03) = &H03 Then
      Exit Do
    EndIf
  Loop
  SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

3.22.68 SyncRobots関数

動作予約中のロボットの状態を返す関数です。

書式

SyncRobots

戻り値

ロボットが動作予約中のとき、そのロボット番号に対応するビットを"1"、予約中でないとき"0"とする整数値を返します。

- ビット0: ロボット番号1
- ビット 1: ロボット番号2
- :
- ビット15: ロボット番号16

解説

SyncRobots関数は、ロボット動作コマンドのSYNCパラメーターによるロボットの動作予約の状態チェックを行います。SyncRobotsでチェックされる状態は、ロボット番号に対応するビットの状態で表されます。それぞれ、各ビットはロボットが動作予約中のとき (1)か予約中でないとき (0)かを示します。予約中のロボットはSyncRobots命令で動作を開始することができます。

参照

[SyncRobots](#)

SyncRobots関数使用例

下記例では、動作コマンドのSYNCパラメーターとSyncRobotsを使って、2台のロボットを同時に動作開始します。

```
Function Main
  Xqt Func1
  Xqt Func2
  Do
    Wait 0.1
    If (SyncRobots And &H03) = &H03 Then
      Exit Do
    EndIf
  Loop
  SyncRobots 1,2
Fend

Function Func1
  Robot 1
  Motor On
  Go P1 SYNC
Fend

Function Func2
  Robot 2
  Motor On
  Go P1 SYNC
Fend
```

3.22.69 SysConfig

システム設定パラメーターを表示します。

書式

SysConfig

結果

システム設定パラメーターを表示します。

解説

システム制御データとして現在設定されている値を表示します。ロボットの納入時やデータを変更したときは、このデータを保存してください。保存するには、Epson RC+ の[ツール]-[メンテナンス]メニュー-[コントローラー設定バックアップ]で行います。

表示されるのは、次のデータです。(次の数値は例で、実際の数値はコントローラーの機種によって異なります。)

```
' Version:
'   Firmware 1, 0, 0, 0

' Options:
'   External Control Point
'   RC+ API

' HOUR: 414.634

' Controller:
'   Serial #: 0001

' ROBOT 1:
'   Name: Mnp01
'   Model: PS3-AS10
'   Serial #: 0001
'   Motor On Time: 32.738
'   Motor 1: Enabled, Power = 400
'   Motor 2: Enabled, Power = 400
'   Motor 3: Enabled, Power = 200
'   Motor 4: Enabled, Power = 50
'   Motor 5: Enabled, Power = 50
'   Motor 6: Enabled, Power = 50

ARCH 0, 30, 30
ARCH 1, 40, 40
ARCH 2, 50, 50
ARCH 3, 60, 60
ARCH 4, 70, 70
ARCH 5, 80, 80
ARCH 6, 90, 90
ARMSET 0, 0, 0, 0, 0, 0
HOF5 0, 0, 0, 0, 0, 0
HORDR 63, 0, 0, 0, 0, 0
RANGE -7427414, 7427414, -8738134, 2621440, -3145728, 8301227, -5534152, 5534152,
-3640889, 3640889, -6553600, 6553600
BASE 0, 0, 0, 0, 0, 0
WEIGHT 2, 0
INERTIA 0.1, 0
XYLIM 0, 0, 0, 0, 0, 0

' Extended I/O Boards:
'   1: Installed
'   2: Installed
```

```
' 3: None installed
' 4: None installed

' Fieldbus I/O Slave Board:
'   Installed
'   Type: PROFIBUS

' Fieldbus I/O Master Board:
'   None installed

' RS232C Boards:
'   1: Installed
'   2: None installed

' PG Boards:
'   1: None installed
'   2: None installed
'   3: None installed
'   4: None installed
```

SysConfig使用例

```
> SysConfig
```


3.22.70 SysErr関数

最新のエラーステータス、またはワーニングステータスを返します。

書式

SysErr [(情報番号)]

パラメーター

情報番号

エラーコード、またはワーニングコードのどちらを取得するかを整数値で指定します。(省略可)0: エラーコード (省略時)1: ワーニングコード

戻り値

コントローラーのエラーコード、またはワーニングコードを整数値で返します。

解説

本関数は、NoEmgAbortタスク(Xqt時に、NoEmgAbortを指定して開始された特別なタスク)とバックグラウンドタスクでのみ使用します。

コントローラーのエラーコード、ワーニングコードとはLCDに表示されているエラーコード、ワーニングコードです。

エラーやワーニングが発生していない場合は "0" を返します。

参照

[ErrMsg\\$関数](#), [ErrorOn関数](#), [Trap \(システム状態トリガー\)](#), [Xqt](#)

SysErr関数使用例

下記例は、コントローラーがエラー状態になるのを監視し、エラーが発生した場合、エラー番号に応じてI/Oをオンオフするプログラムです。

注意

■ Forcedフラグ

本プログラム例では、On/Off命令にForcedフラグを指定しています。

エラー発生中、および、非常停止中や安全扉開時にI/O出力が変化するのでシステム設計上の注意が必要です。

■ エラー発生後の処理

本例のように、エラーが発生し、必要な処理をした後、速やかにタスクは終了するようにしてください。

```
Function main
Xqt ErrorMonitor, NoEmgAbort
:
:
Fend

Function ErrorMonitor
Wait ErrorOn
If 4000 < SysErr Then
Print "Mortion Error = ", SysErr
Off 10, Forced
On 12, Forced
```

```
Else
  Print "Other Error = ", SysErr
  Off 11, Forced
  On 13, Forced
EndIf

Fend
```

3.23 T

3.23.1 Tab\$関数

指定した数だけの半角タブ文字列を返します。

書式

Tab\$(指定数)

パラメーター

指定数

半角タブの数を整数値で指定します。

戻り値

半角タブ文字を含む文字列を返します。

解説

指定した数だけの半角タブ文字列を返します。

参照

[Left\\$関数](#), [Mid\\$関数](#), [Right\\$関数](#), [Space\\$関数](#)

Tab\$関数使用例

```
Integer i
Print "X", Tab$(1), "Y"
Print
For i = 1 To 10
    Print "x", Tab$(i), "y"
Next i
```

3.23.2 Tan関数

指定した数値のタンジェント値を返します。

書式

Tan(角度)

パラメーター

角度

角度を表わす実数値を指定します。

戻り値

指定した角度を表わす数値のタンジェント値を、数値で返します。

解説

Tan はタンジェント値を返します。パラメーターの単位はラジアンでなければなりません。

角度をラジアンに変換するためにはDegToRad関数を使います。

参照

[Abs関数](#), [Atan関数](#), [Atan2関数](#), [Cos関数](#), [Int関数](#), [Mod 演算子](#), [Not 演算子](#), [Sgn関数](#), [Sin関数](#), [Sqr関数](#), [Str\\$関数](#), [Val関数](#)

Tan関数使用例

```
Function tantest
  Real num
  Print "Enter number in radians to calculate tangent for:"
  Input num
  Print "The tangent of ", num, "is ", Tan(num)
Fend
```

次はコマンドウィンドウからTan関数を使う例です。

```
> print tan(0)
0.00
> print tan(degtorad(45))
1
>
```

3.23.3 TargetOK関数

現在位置から目標座標へのPTP (point to point) 動作が可能かどうかを返します。

書式

TargetOK(目標座標)

パラメーター

目標座標

動作可能かどうかを調べる目標座標をポイントデータで指定します。

戻り値

現在位置から目標座標までのPTP動作が可能であれば"True"を、そうでない場合は"False"を返します。

解説

目標座標や姿勢に到達できるかを、実際に (ロボットを)動かす前に確認します。ただし、目標座標にたどり着くまでの軌道は考慮されていません。

参照

[CurPos関数](#), [FindPos関数](#), [InPos関数](#), [WaitPos](#)

TargetOK関数使用例

```
If TargetOK(P1) Then
  Go P1
EndIf

If TargetOK(P10 /L /F) Then
  Go P10 /L /F
EndIf
```

3.23.4 TaskDone関数

タスクの終了を確認する関数です。

書式

TaskDone (タスク識別子)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

戻り値

タスクが終了している場合は"True"、それ以外は"False"を返します。

解説

TaskDone はタスクが終了しているか確認するために使います。

参照

[TaskState関数](#), [TaskWait](#)

TaskDone関数使用例

```
Xqt 2, conveyor
Do
  .
  .
Loop Until TaskDone(conveyor)
```

3.23.5 TaskInfo関数

タスクの状態情報を返します。

書式

TaskInfo(タスク識別子, インデックス)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

インデックス

検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報を整数値で返します。

解説

インデックス	解説
0	タスク番号
1	0 - バックグラウンドタスク 1 - 通常のタスクかNoPauseタスクかNoEmgAbortタスク
2	タスクタイプ 0 - 通常のタスクXqtで何も指定しないか、Normalを指定して開始したタスク 1 - NoPauseタスクXqtでNoPauseを指定して開始したタスク 2 - NoEmgAbortタスクXqtでNoEmgAbortを指定して開始したタスク 3 - Trapタスク 4 - バックグラウンドタスク
3	1 - 指定したタスク実行中 2 - 指定したタスクはイベント待ち 3 - 指定したタスクは一時停止またはHalt中 4 - 指定したタスクはクイックポーズ状態 5 - 指定したタスクはエラー状態
4	イベント待機中にタイムアウト発生 (TWと同じ)
5	イベント待ち時間 (msec)
6	タスクが選択したロボット番号
7	タスクが使用中のロボット番号

タスクが実行していないとき、インデックス0～7は全て-1 になります。

参照

[CtrlInfo関数](#), [RobotInfo関数](#), [TaskInfo\\$関数](#)

TaskInfo関数使用例

```
If TaskInfo(1, 3) <> 0 Then
    Print "Task 1 is runnning"
Else
    Print "Task 1 is not running"
EndIf
```


3.23.6 TaskInfo\$関数

タスクのテキスト情報を返します。

書式

TaskInfo\$(タスク識別子, インデックス)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

インデックス

検索する情報のインデックスを整数値で指定します。

戻り値

指定した情報を文字列で返します。

解説

TaskInfo\$で検索できる情報は次の表のとおりです。

インデックス	解説
0	タスク名
1	開始日時
2	現在実行しているファンクション名
3	ファンクションを含んでいる、プログラムの行番号

参照

[CtrlInfo関数](#), [RobotInfo関数](#), [TaskInfo関数](#)

TaskInfo\$関数使用例

```
Print "Task 1 started: "TaskInfo$(1, 1)
```

3.23.7 TaskState関数

タスクの現在の状態を取得する関数です。

書式

TaskState(タスク識別子)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

戻り値

- 0: 指定タスク未実行
- 1: 指定タスク実行中
- 2: 指定タスクイベント待機中
- 3: 指定タスク一時停止中
- 4: 指定タスククイックポーズ中
- 5: 指定タスクエラー状態

解説

TaskState はタスク番号、またはタスク名で指定したタスクの現在状態を取得します。

参照

[TaskDone関数](#), [TaskWait](#)

TaskState関数使用例

```
If TaskState(conveyor) = 0 Then
    Xqt 2, conveyor
EndIf
```

3.23.8 TaskWait

指定のタスクの終了を待ちます。

書式

TaskWait (タスク識別子)

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32
- バックグラウンドタスク: 65～80
- トラップタスク: 257～267

参照

[TaskDone関数](#), [TaskState関数](#)

TaskWait使用例

```
Xqt 2, conveyor  
TaskWait conveyor
```

3.23.9 TC

トルク制御モードの設定および現在のモードを表示します。

書式

(1) TC { On | Off }

(2) TC

パラメーター

On | Off

- On: トルク制御モードを有効にします。
- Off: トルク制御モードを無効にします。

戻り値

パラメーター省略時、現在のトルク制御モードを表示します。

解説

TC On/Offでトルク制御モードを有効、無効にします。

トルク制御モードは、モーター出力の制限値を設定することで一定の力を発生させるためのモードで、ハンドを対象物に一定の力で押しつける、ハンドと対象物の接触を保ちながら対象物の動きに合わせてならい動作をさせる場合などに利用します。

トルク制御を有効にする前に、TCLimでトルク制限値を設定してください。

トルク制御モード中も、動作コマンド実行時には、目標位置に位置決めするようにロボットが動作します。ロボットが対象物に接触して、モーター出力がトルク制限値に達したところで、ロボットは停止し、一定トルクを保持します。

下記のいずれかの場合、トルク制御モードを無効にします。

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

参照

[TCLim](#), [TCSpeed](#)

TC使用例1

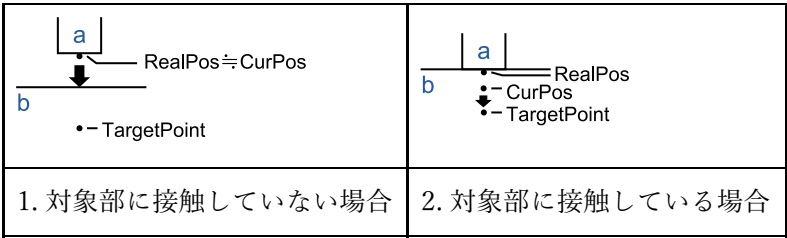
```
Speed 5
Go ApproachPoint

'z軸トルク制限値を20に設定します。
TCLim -1, -1, 20, -1

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

注意

- Safety基板を搭載したコントローラーで位置異常が検出される場合
- 「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。
- 大きく離れてしまうと、Safety基板が故障と判断し、「エラーNo.9801 Safety基板が位置異常を検出した」エラーが出ます。(Safety基板を搭載したコントローラー)
- ロボットの現在位置はRealPos関数で取得できます。
- ロボットの現在の動作目標位置はCurPos関数で取得できます。
- スカラの場合、RealPosの差とCurPosの差はJ1:10度, J2:10度, J3:40mm, J4:90度程度以下で使用することを奨励します。
- エラーが発生する値は機種ごとに異なります。エラーにならない範囲で機種ごとに大きく設定が可能です。
- TCSpeedで速度制限すると、対象物に接触する前にRealPosとCurPosに差が生じます。TCSpeedを利用しないか、利用する場合はSpeedと同じ値にしてください。



記号	説明
a	ハンド
b	対象物

1. ロボットの現在位置 (RealPos)と、ロボットの現在の動作目標位置 (CurPos)は近い位置です。
2. ロボットの現在位置 (RealPos)は、対象物に接触した位置を維持します。
ロボットの現在の動作目標位置 (CurPos)は、TagetPointに向かって進行します。
この距離が一定以上開くとエラーNo.9801が発生します。
- Safety基板の位置異常を発生させずに利用するサンプルプログラムを以下に示します。
- Tillコマンドを使い、「ロボットの現在位置」と、「ロボットの現在の動作目標位置」の差が一定以上で次の処理に進むことによりエラーを回避できます。
- 差をなるべく小さい値にすることで、早期に次の処理に移るため、タクトタイム向上につながります。

TC使用例2

```
Speed 5
Go ApproachPoint

'z軸トルク制限値を20%に設定します。
TCLim -1, -1, 20, -1

Xqt posDiffChk(10.0) '位置差分チェックタスク起動。J3軸10.0mm以上の差で、フラグを立てる
```

```
TC On
Go TargetPoint Till MemSw(0)
Wait 3
Go ApproachPoint
TC Off

Function posDiffChk(Zth As Double)
    Do
        If (Abs(CZ(RealPos) - CZ(CurPos)) > Zth) Then
            ' 「ロボットの現在位置」と、「ロボットの現在の目標位置」の差が閾値を超過したか?
            MemOn (0) ' 位置偏差大フラグ ON
        Else
            MemOff (0) ' 位置偏差大フラグ クリア
        EndIf
        Wait 0.01
    Loop
Fend
```

3.23.10 TCLim

トルク制御モードのための各関節のトルク制限値を設定します。

書式

TCLim [第1関節トルク制限値, 第2関節トルク制限値, 第3関節トルク制限値, 第4関節トルク制限値 [, 第5関節トルク制限値] [, 第6関節トルク制限値] [, 第7関節トルク制限値] [, 第8関節トルク制限値] [, 第9関節トルク制限値]]

パラメーター

第1関節トルク制限値

瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第2関節トルク制限値

瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第3関節トルク制限値

瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第4関節トルク制限値

瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第5関節トルク制限値

省略可。瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第6関節トルク制限値

省略可。瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第7関節トルク制限値

省略可。瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第8関節トルク制限値

省略可。S軸の瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

第9関節トルク制限値

省略可。T軸の瞬時最大トルクに対する割合 (1～100の整数、単位: %)を式または数値で指定します。-1のときトルク制限値を無効とし、通常的位置制御モードとなります。

戻り値

パラメーター省略時、現在のトルク制限値を表示します。

解説

トルク制限値の設定は、TC Offの時に行ってください。TC Onにしたときに、設定した値が有効になります。

設定値が低すぎるとロボットが動作せず、目標位置に到達する前に動作コマンドが終了します。

下記のいずれかの場合、TCLim設定値は、初期化されます。

- コントローラー起動時
- Motor On実行

- SFree, SLock, Brake 実行
- Reset, Reset Error実行
- 停止ボタン, Quit All実行などによるタスクの終了

参照

[TC](#), [TCLim関数](#), [TCSpeed](#)

TCLim使用例

```
Speed 5
Go ApproachPoint

' z軸トルク制限値を20%に設定します。
TCLim -1, -1, 20, -1

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

注意

- Safety基板を搭載したコントローラーで位置異常が検出される場合

「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。

大きく離れてしまうと、Safety基板が故障と判断し、「エラーNo.9801 Safety基板が位置異常を検出した」エラーが出ます。(Safety基板を搭載したコントローラー)

詳細は、TC コマンドの説明を参照してください。

3.23.11 TCLim関数

指定した関節のトルク制限値を返します。

書式

TCLim(関節番号)

パラメーター

関節番号

トルク制限値を取得したい関節の関節番号を式または数値で指定します。付加軸のS軸は8、T軸は9になります。

戻り値

現在のトルク制限値をあらわす1～100の整数を返します。トルク制限値が無効の場合、(1を返します。

参照

[TC](#), [TCLim](#), [TCSpeed](#)

TCLim関数使用例

```
Print "現在のZ軸トルク制限値:", TCLim(3)
```

3.23.12 TCPSpeed関数

算出された現在のツール先端 (TCP)速度を返します。

書式

TCPSpeed

戻り値

算出された現在のツール先端速度を実数で返します。(単位: mm/秒)

解説

CP動作コマンドを実行したときの、算出されたツール先端速度をmm/秒で返します。

CP動作コマンドとは、Move, TMove, Arc, Arc3, CVMove, およびJump3CPです。この速度は実際の速度と異なります。この速度は、ファンクションを呼び出したときにシステムが動作計画したツール先端速度です。

モーターの追従遅れが考慮されません。ロボットがPTP動作コマンドを実行している場合は、"0"を返します。

付加軸を使用している場合でも、ロボットの移動速度のみを返します。例えば、付加軸を走行軸に使用するケースでも付加軸移動速度を考慮しません。

参照

[AccelS](#), [CurPos関数](#), [InPos関数](#), [SpeedS](#)

TCPSpeed関数使用例

```
Function MoveTest
AccelS 4000, 4000
SpeedS 200
Xqt ShowTCPSpeed
Do
    Move P1
    Move P2
Loop
Fend

Function ShowTCPSpeed
Do
    Print "Current TCP speed is: ", TCPSpeed
    Wait .1
Loop
Fend
```

3.23.13 TCSpeed

トルク制御中の速度制限値を設定します。

書式

TCSpeed [速度]

パラメーター

速度

最大速度に対する割合 (1~100の整数、単位: %)を式または数値で指定します。

解説

トルク制御中は、Speedコマンドなどの速度設定に関わらず、TCSpeedで設定された速度に制限されます。

トルク制御中に制限値以上の速度を検出するとエラーが発生します。

下記のいずれかの場合、TCSpeed設定値は、100%に初期化されます。

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

参照

[TC](#), [TCLim](#), [TCSpeed関数](#)

TCSpeed使用例

```
Speed 5
Go ApproachPoint

' z軸トルク制限値を20(に設定します。
TCLim -1, -1, 20, -1
' トルク制御中の速度をSpeed設定と同じ5%に設定します。
TcSpeed 5

TC On
Go TargetPoint
Wait 3
Go ApproachPoint
TC Off
```

注意

- Safety基板を搭載したコントローラーで位置異常が検出される場合

「ロボットの現在位置」と、「ロボットの現在の動作目標位置」が大きく離れないよう、プログラムを修正してください。

TCSpeedで速度制限すると、「ロボットの現在位置」と「ロボットの現在の目標位置」に差が生じます。TCSpeedを利用しないか、利用する場合はSpeedと同じ値にしてください。

大きく離れてしまうと、Safety基板が故障と判断し、「エラーNo.9801 Safety基板が位置異常を検出した」エラーが出ます。(Safety基板を搭載したコントローラー)

詳細は、TC コマンドの説明を参照してください。

3.23.14 TCSpeed関数

トルク制御中の速度制限値を返します。

書式

TCSpeed

戻り値

現在の速度制限値を表す1～100の整数を返します。

参照

[TC](#), [TCSpeed](#), [TCLim](#)

TCSpeed使用例

```
Integer var  
var = TCSpeed
```

3.23.15 TeachOn関数

ティーチモードの状態を返します。

書式

TeachOn

戻り値

ティーチモードのときは"True"、それ以外のときは"False"を返します。

解説

本関数は、バックグラウンドスクでのみ使用します。

参照

[ErrorOn関数](#), [EStopOn関数](#), [SafetyOn関数](#), [Xqt](#)

TeachOn関数使用例

下記例は、コントローラーがティーチモードになるのを監視し、ティーチモードに移行した場合I/Oをオンオフするプログラムです。

```
Function BGMain
  Do
    Wait 0.1
    If TeachOn = True Then
      On teachBit
    Else
      Off teachBit
    EndIf
    If SafetyOn = True Then
      On safetyBit
    Else
      Off safetyBit
    EndIf
    If PauseOn = True Then
      On PauseBit
    Else
      Off PauseBit
    EndIf
  Loop
Fend
```

3.23.16 TeachPoint

SPEL+のプログラムからEpson RC+のティーチポイント画面を起動します。

書式

TeachPoint (プロンプト, タイトル, ポイント番号)

パラメーター

- プロンプト*: 画面に表示するメッセージの文字列 (最大64文字)
- タイトル*: 画面のタイトルバーに表示する文字列 (最大32文字)
- ポイント番号: ポイントの番号を整数値で指定 (0-999の整数値)

*: 省略可能です。

戻り値

ティーチに成功した場合は"True"、失敗した (あるいはキャンセルした) 場合は"False"を返します。

解説

SPEL+のタスクからEpson RC+のティーチポイント画面を起動/表示するときに、本コマンドを使います。

タスクは、画面が閉じられるまで一時停止します。



Epson RC+の画面からロボットコマンドを実行するときは、その画面表示中にロボットを制御する他のタスクがないことを確認してください。

他にロボットを制御するタスクが起動している場合は、エラーになります。

参照

[RunDialog](#), [ToolWizard関数](#)

TeachPoint使用例

```
Boolean sts

sts = TeachPoint ("ACME Gripper", "Teach Reference Point", 1)
If sts = True Then
    Print "The point was taught"
Else
```

```
Print "The point was not taught"  
EndIf
```


3.23.17 TGo

現在のツール座標系での、オフセットPTP動作を実行します。

書式

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

パラメーター

目標座標

ポイントデータを用いて動作の目標位置を指定します。

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

! 並列処理!

動作中に、I/Oその他のコマンドを実行するよう、並列処理ステートメントを付加することができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

現在のツール座標系での、オフセットPTP動作を実行します。

ポイントデータで与えられた姿勢フラグは無視され、現在の姿勢フラグが維持されます。ただし、垂直6軸型ロボット (Nシリーズを含む) では、関節移動量が小さくなるよう姿勢フラグを自動的に変更します。

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

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

!並列処理!を使って、動作と並列に他の処理を実行することができます。

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

参照

[Accel](#), [CP](#), [Find](#), [!...! 並列処理](#), [P# \(2. ポイント指定\)](#), [Speed](#), [Till](#), [TMove](#), [Tool](#)

TGo使用例

```
> TGo XY(100, 0, 0, 0) ' (ツール座標系で) x方向に100 mm移動
Function TGoTest

    Speed 50
    Accel 50, 50
    Power High

    Tool 0
```

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

```
Go P1
Print Here
TGo XY(0, 0, -30, 0)
Print Here
```

```
Go P2
Print Here
TGo XY(0, 0, -30, 0)
Print Here
```

```
Fend
```

[出力結果]

X:	300.000	Y:	300.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/R	/0
X:	300.000	Y:	300.000	Z:	-50.000	U:	0.000	V:	0.000	W:	0.000	/R	/0
X:	300.000	Y:	300.000	Z:	-20.000	U:	0.000	V:	0.000	W:	0.000	/L	/0
X:	300.000	Y:	300.000	Z:	-50.000	U:	0.000	V:	0.000	W:	0.000	/L	/0

3.23.18 Till

Jump, Go, Move, その他の動作命令でTill指定したときに、動作途中で停止し、処理を終了させる条件の設定と表示を行います。

書式

Till [イベント条件式]

パラメーター

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子 (=, <>, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

- 関数: Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, Force, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
- 変数: Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort型のバックアップ変数、グローバル変数、モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

- 演算子: And, Or, Xor

[例]

```
Till Sw(5) = On  
Till Sw(5) = On And Sw(6) = Off
```

解説

Till ステートメントは、単独で記述するか、動作命令ステートメントの修飾子として記述してください。

Till条件式には、上記の関数を1つ以上含んでいなくてはなりません。

Till条件式に変数が含まれたときは、その値は、Till条件設定時に演算されます。意図しない条件になる場合があるので、条件式には変数を使用しないことを推奨します。複数のTillステートメントを記述することもできます。その場合、最後に実行されたTill 条件が有効となります。

パラメーターを省略すると、現在のTill設定が表示されます。

注意

- コントローラー電源オン時のTill 設定

コントローラー電源オン時のTill 条件は未設定です。電源オン後は、条件を設定してから使用してください。

- Till 条件成立をチェックするStat関数とTillOn関数

Till修飾子を用いた動作コマンドを実行した後で、Till条件が成立したかどうかを、Stat関数またはTillOn関数を用いてチェックすることができます。

- イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。

- 変数値が0.01秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1システム内で使用できる変数待ちの個数は最大64個です (Waitなどのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数をByrefで参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数が含まれたときは、その値は動作命令開始時に演算されます。意図しない条件になる場合があるので、整数式には変数を使用しないことを推奨します。

参照

[Find](#), [Go](#), [In関数](#), [InW関数](#), [Jump](#), [MemIn関数](#), [MemSw関数](#), [Move](#), [Stat関数](#), [Sw関数](#), [TillOn関数](#), [SF_GetStatus関数](#)

Till使用例

下記は、プログラム中でTill命令を用いた例です。

Till Sw(1) = Off	'Till条件を設定 (入力ビット1オフ)
Go P1 Till	'前の行の条件を満たす場合は停止
Till Sw(1) = On And Sw(\$1) = On	'新しいTill条件を設定
Move P2 Till	'前の行の条件を満たす場合は停止
Move P5 Till Sw(10) = On	'この行の条件を満たす場合は停止

3.23.19 TillOn関数

Tillの状態を返します。

書式

TillOn

戻り値

Tillを使った直前の動作コマンドで、Tillの条件が成立したらTrueを返します。

解説

Tillの条件が成立したらTrueを返します。

TillOn は下記と同様です。

```
((Stat(1) And 2) <> 0)
```

参照

[EStopOn関数](#), [SafetyOn関数](#), [Sense](#), [Stat関数](#), [Till](#)

TillOn関数使用例

```
Go P0 Till Sw(1) = On
If TillOn Then
    Print "Till condition occurred during move to P0"
EndIf
```

3.23.20 Time

時間の表示をします。

書式

Time

解説

現在の時間を24時間表記で表示します。

参照

[Date, Time\\$関数](#)

Time使用例

コマンドウィンドウでの実行例

```
> Time  
10:15:32
```

3.23.21 Time関数

コントローラーの積算通電時間を返します。

書式

Time(単位指定)

パラメーター

単位指定

0～2の整数値を指定します。0～2でコントローラーの積算通電時間の単位を示します。

- 0: 時
- 1: 分
- 2: 秒

戻り値

コントローラーの積算通電時間を実数値で返します。

参照

[Hour](#)

Time関数使用例

```
Function main
  Real t_h, t_m, t_s

  t_h = Time(0)    '通電時間を時間で返す
  t_m = Time(1)    '通電時間を分で返す
  t_s = Time(2)    '通電時間を秒で返す
  Print "This controller has been used:"
  Print t_h, "hours, ",
  Print t_m, "minutes, ",
  Print t_s, "seconds"
Fend
```

3.23.22 Time\$関数

現在のシステム時刻を返します。

書式

Time\$

戻り値

時間を24時間表記の文字列で返します。

書式はhh:mm:ss [時間:分:秒]です。

参照

[Date](#), [Date\\$関数](#), [Time](#)

Time\$関数使用例

```
Print "The current time is: ", Time$
```


3.23.23 TLClr

ツール座標系の設定をクリアします。

書式

TLClr ツール番号

パラメーター

ツール番号

クリアするツールを、整数または式で指定します。(ツール0はデフォルトなので、クリアできません。)

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLSet](#)

TLClr使用例

```
TLClr 1
```

3.23.24 TLDef関数

ツールの設定状態を返します。

書式

TLDef(ツール番号)

パラメーター

ツール番号

状態を返すツールを、整数または式で指定します。

戻り値

指定したツールが設定されている場合"True"、設定されていない場合"False"を返します。

参照

[Arm](#), [ArmClr](#), [ArmSet](#), [ECPSet](#), [Local](#), [LocalClr](#), [Tool](#), [TLClr](#), [TLSet](#)

TLDef関数使用例

```
Function DisplayToolDef(toolNum As Integer)

    If TlDef(toolNum) = False Then
        Print "Tool ", toolNum, "is not defined"
    Else
        Print "Tool ", toolNum, ": ",
        Print TlSet(toolNum)
    EndIf
End
```

3.23.25 TLSet

ツール座標系を設定, 表示します。

書式

- (1) TLSet ツール座標系番号, ツール設定データ
- (2) TLSet ツール座標系番号
- (3) TLSet

パラメーター

ツール座標系番号

設定するツールを1～15の整数値で指定します。(Tool 0はデフォルトツールで変更することはできません。)

ツール設定データ

設定するツール座標系の原点および向きを、P番号, P(式), ポイントラベル, ポイント式のいずれかで指定します。

結果

パラメーターをすべて省略すると、すべてのTLSet 設定を表示します。

ツール番号だけを指定すると、指定のTLSet 設定を表示します。

解説

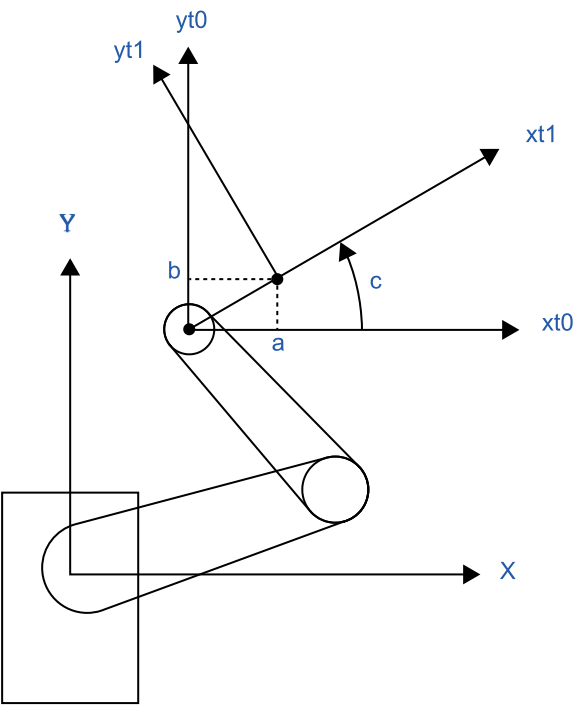
Tool 0座標系 (ハンド座標系)に対する相対原点位置と相対回転角度を指定して、ツール座標系Tool 1, Tool 2, Tool 3を定義します。

```
TLSet 1, XY(50,100,-20,30)
TLSet 2, P10 +X(20)
```

上記の場合、座標値P10が参照され、X値に20を加えています。アーム属性とローカル座標系番号は無視されます。

```
TLSET 1, XY(100,60,-20,30)
      a      b      c      d      e
```

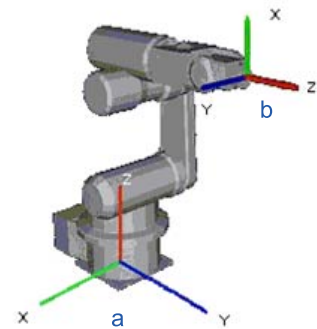
記号	説明
a	ツール座標系番号
b	ツール座標系の原点のX軸方向位置(次図a)
c	ツール座標系の原点のY軸方向位置(次図b)
d	ツール座標系の原点のZ軸方向位置
e	ツール座標系の回転角度(次図c)



記号	説明
X, Y	ロボット座標系
xt0, yt0	ツール0座標系
xt1, yt1	ツール1座標系

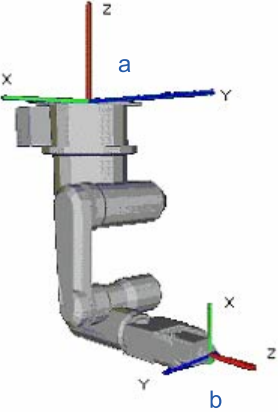
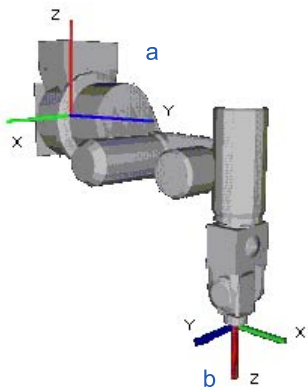
垂直6軸型ロボットでのTLSetの使用

垂直6軸型ロボットでは、すべての関節を0度の位置にしたとき、第6関節のフランジ面中心を原点とし、垂直上方向にX軸、ロボット座標系X軸方向にY軸、第6関節フランジ面に対して鉛直方向にZ軸を持つ座標系がツール0座標系です。(下図参照)



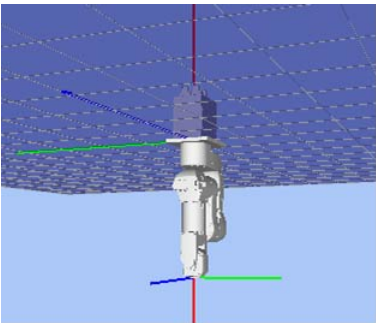
記号	説明
a	ロボット座標系
b	ツール0座標系

垂直6軸型ロボットの設置方法により、ツール0座標系の定義が異なります。

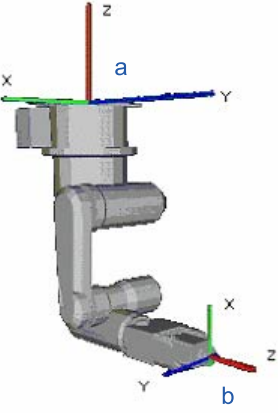
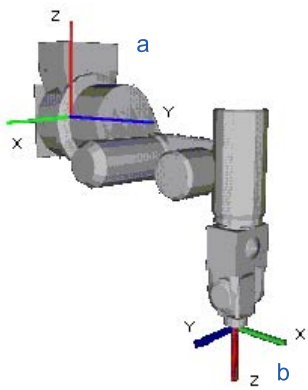
天吊設置	壁掛け設置
	
a: ロボット座標系 b: ツール0座標系	

NシリーズでのTLSetの使用

Nシリーズでは、すべての関節を0度の位置にしたとき、ロボット座標系-X軸方向にX軸、ロボット座標系Y軸方向にY軸、ロボット座標系-Z軸方向にZ軸を持つ座標系がツール0座標系です。(下図参照)



Nシリーズの設置方法により、ツール0座標系の定義が異なります。

天吊設置	壁掛け設置
	
a: ロボット座標系 b: ツール0座標系	

注意

- TLSet 値は保持されます

TLSet 値は保持されています。ツール定義をクリアするには、TLClrを使用します。

参照

[Tool](#), [Arm](#), [ArmSet](#), [TLSet関数](#), [TLClr](#)

TLSet使用例

下記は、ツール定義をした場合の動作と、ツール定義をしない場合の動作の違いを理解するために、コマンドウィンドウから操作を行った例です。

```
> TLSet 1, XY(100, 0, 0, 0)  'Tool 1 (ハンドの座標系からX方向に100 mm)を  
                             'ツール座標系で定義  
> Tool 1                    'TLSetで定義したTool 1を選択  
> TGo P1                    'Tool 1の目標座標をP1に設定  
> Tool 0                    '今後の動作でツールを使用しないように設定  
> Go P1                     'U関節の中央をP1に設定
```

3.23.26 TLSet関数

設定されたツール座標系データを返します。

書式

TLSet(ツール座標系番号)

パラメーター

ツール座標系番号

ツール番号を整数値で指定します。

戻り値

ツール座標系データを返します。

参照

[TLSet](#)

TLSet関数使用例

```
P1 = TLSet(1)
```

3.23.27 TMOut

Wait命令実行時のタイムアウトエラー (エラー2280)発生までの時間を設定します。

書式

TMOut 秒

パラメーター

秒

タイムアウト時間を実数値で指定します。範囲は0～2147483です。(単位: 秒)

解説

TMOutはWait命令実行時のタイムアウトエラー (エラー2280)発生までの時間を設定します。タイムアウト時間を0秒と設定した場合、タイムアウトは働かず、Wait命令は指定した条件が成立するまで無期限に待機します。

TMOut のデフォルト値は "0"です。

参照

[In関数](#), [MemSw関数](#), [OnErr](#), [Sw関数](#), [TW関数](#), [Wait](#)

TMOut使用例

```
TMOut 5
Wait MemSw(0) = On
```


3.23.28 TMove

現在のツール座標系で、オフセット直線補間動作を実行します。

書式

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

パラメーター

目標座標

ポイントデータを用いて動作の目標位置を指定します。

ROT

ツール姿勢変化を優先させ、動作の速度、加減速度を決めます。省略可能です。

CP

パスモーションを指定します。省略可能です。

Till | Find

TillまたはFind式を記述します。省略可能です。

```
Till | Find
Till Sw(式) = {On | Off}
Find Sw(式) = {On | Off}
```

! 並列処理!

動作中にI/Oその他のコマンドを実行するために、並列処理ステートメントを付け加えることができます。省略可能です。

SYNC

動作コマンドを予約します。SyncRobotsによる動作開始まで、ロボットは動作しません。

解説

現在のツール座標系で、オフセット直線補間動作を実行します。

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

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

ロボットのツール先端位置を、特定の座標に固定したままツール姿勢のみを変化させようとした場合や、ツール先端の移動距離に対してツール姿勢変化が大きい場合、ツール姿勢変化速度が著しく速くなる恐れがあります。これを防ぐため、ツール姿勢変化速度が大きい場合に自動で動作速度を制限する機能が働きます。

CP動作時のツール姿勢変化速度の上限値を手動で設定したい場合は、SpeedRLimitationをオンにしてください。SpeedRLimitationをオンにした場合、CP動作時のツール姿勢変化速度が設定したSpeedRを超える場合は、ツール姿勢変化速度がSpeedRになるように動作速度を制限します。ツール姿勢変化速度が設定したSpeedRを超えない場合は、設定したSpeedSで動作します。ツール姿勢変化速度の上限値はSpeedRであらかじめ設定してください。

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

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

!並列処理!を使って、動作と並列に他の処理を実行することができます。

注意

- TMoveをCPとともに用いる

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

参照

[AccelS](#), [CP](#), [Find](#), [!...! 並列処理](#), [P# \(2. ポイント指定\)](#), [SpeedS](#), [TGo](#), [Till](#), [Tool](#)

TMove使用例

```
> TMove XY(100, 0, 0, 0) ' (ツール座標系で) x方向に100 mm移動
Function TMoveTest

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

    Tool 0
    P1 = XY(300, 300, -20, 0)
    P2 = XY(300, 300, -20, 0) /L

    Go P1
    Print Here
    TMove XY(0, 0, -30, 0)
    Print Here

    Go P2
    Print Here
    TMove XY(0, 0, -30, 0)
    Print Here

Fend
```

[出力結果]

```
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /R /0
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /R /0
X: 300.000 Y: 300.000 Z: -20.000 U: 0.000 V: 0.000 W: 0.000 /L /0
X: 300.000 Y: 300.000 Z: -50.000 U: 0.000 V: 0.000 W: 0.000 /L /0
```

3.23.29 Tmr関数

Tmr関数は、タイマーがスタートしてからの経過時間を、単位秒で返します。

書式

Tmr(タイマー番号)

パラメーター

タイマー番号

64のタイマーのうち、どのタイマーをチェックするか、整数 (0～63)を式または数値で指定します。

戻り値

指定したタイマーの経過時間を実数値 (秒単位)で返します。タイマーの範囲は、0～約1.7E+31です。タイマ解像度は、0.001秒です。

解説

指定したタイマーがスタートしてからの経過時間を返します。ElapsedTime関数と異なり、プログラムが一時停止状態となっている時間も経過時間としてカウントします。

タイマーは TmResetでリセットすることができます。

```
Real overhead  
  
TmReset 0  
overHead = Tmr(0)
```

参照

[ElapsedTime関数](#), [TmReset](#)

Tmr関数使用例

```
TmReset 0          'タイマー0をリセット  
For i = 1 To 10    '10回実行  
    GoSub Cycle  
Next  
Print Tmr(0) / 10  'サイクルタイムを計算して表示
```

3.23.30 TmReset

Tmr関数で使用するタイマーをリセットします。

書式

TmReset タイマー番号

パラメーター

タイマー番号

64個のタイマーのうち、リセットするタイマーの番号を整数 (0～63) で指定します。

解説

タイマー番号で指定したタイマーをリセットし、スタートします。

Tmr関数は、指定したタイマーの経過時間を取得します。

参照

[Tmr関数](#)

TmReset使用例

```
TmReset 0          'タイマー0をリセット
For i = 1 To 10    '10回実行
    GoSub CYL
Next
Print Tmr(0)/10    'サイクルタイムを計算して表示
```

3.23.31 Toff

LCDに実行行の表示をオフします。

書式

Toff

解説

Toffを実行するとタスクの実行行がLCDに表示されません。

注意

- 使用可能なコントローラーについて

RC90/T/VTシリーズでは使用できません。

参照

[Ton](#)

Toff使用例

下記は、コマンドウィンドウからテストする例です。ツール設定時と未設定時の動作の違いを理解することができます。

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

3.23.32 Ton

LCDに実行行を表示するタスクを指定します。

書式

Ton タスク識別子

Ton

パラメーター

タスク識別子

タスク名またはタスク番号を整数値、または式で指定します。タスク名は、Xqtステートメントで使われたファンクション名か、Runウィンドウかオペレーターウィンドウから起動されたファンクションを指定します。

タスク番号の指定 (整数)

- 通常タスク: 1～32

注意

- 使用可能なコントローラーについて

RC90/T/VTシリーズでは使用できません。

解説

初期状態では、タスク番号1の実行行が表示されます。

Tonを使用すると指定したタスクの実行行をLCDに表示することができます。

タスク識別子を省略した場合、Tonを実行したタスクの実行行がLCDに表示されます。

参照

[Toff](#)

Ton使用例

```
Function main
  Ton MyTask
  ...
  Toff
Fend
```

3.23.33 Tool

ツールの選択、または選択されているツール番号の表示。

書式

(1) Tool ツール番号

(2) Tool

パラメーター

ツール番号

次に続く動作命令で、16個あるツール (整数値0~15)のうち、どれを使うか指定します。省略可能です。

結果

パラメーターを省略すると、現在設定されているツール番号を表示します。

解説

Toolは、ツール番号で指定されたツールを選択します。ツール番号"0"のときは、選択ツールはなく、すべての動作は、先端関節 (回転関節) 中心に対して行われます。しかし、ツール番号"1", "2", "3"などが選択されたときは、ツール設定されたツールの先端に対して動作が行われます。

注意

- 電源オフ時のツール選択への影響

電源をオフしても、選択されたツール座標系は変更されません。

参照

[TGo](#), [TLSet](#), [TMove](#)

Tool使用例

下記は、コマンドウィンドウからテストする例です。ツール設定時と未設定時の動作の違いを理解することができます。

```
>tlset 1, 100, 0, 0, 0    'Tool 1 (ハンドの座標系からx方向に100mm) を  
                          'ツール座標系で定義  
>tool 1                  'TLSetで定義したTool 1を選択  
>tgo p1                  'Tool 1の目標座標をP1に設定  
>tool 0                  '今後の動作でツールを使用しないように設定  
>go p1                   'U関節の中央をP1に設定
```

3.23.34 Tool関数

現在設定されているツール番号を返します。

書式

Tool

戻り値

ツール番号を整数値で返します。

参照

[Tool](#)

Tool関数使用例

```
Integer savTool  
  
savTool = Tool  
Tool 2  
Go P1  
Tool savTool
```


3.23.35 ToolWizard関数

SPEL+のプログラムからEpson RC+のツールウィザード画面を起動します。

書式

ToolWizard (ツール番号)

パラメーター

- ツール番号: ツールの番号を整数値で指定 (0-15の整数値)

戻り値

正常にウィザードが完了した場合はTrue、失敗した (あるいはキャンセルした) 場合はFalseを返します。

解説

SPEL+ のタスクからEpson RC+のツールウィザード画面を起動/表示するときに、本関数を使います。

タスクは、画面が閉じられるまで一時停止します。



Epson RC+の画面からロボットコマンドを実行するときは、その画面表示中にロボットを制御する他のタスクがないことを確認してください。

他にロボットを制御するタスクが起動している場合は、エラーになります。

参照

[RunDialog](#), [TeachPoint](#)

ToolWizard関数使用例

```
Boolean sts

sts = ToolWizard(1)
If sts = True Then
    Print "The tool was defined"
Else
    Print "The tool was not defined"
EndIf
```

3.23.36 Trap (ユーザー定義トリガー)

割込み、および割込み発生時の処理の定義を行います。

Trap命令を使用すると、イベント発生により、ラベルにジャンプする、または、ファンクションを呼び出すことができます。Trap命令には2つのタイプがあります。

- ユーザーの定義する入力状態をトリガーにする4つのTrap
- システム状態をトリガーにする7つのTrapです。

本項目ではユーザ定義トリガーのTrapを説明します。

書式

Trap トラップ番号, イベント条件式 GoTo ラベル

Trap トラップ番号, イベント条件式 Call ファンクション名

Trap トラップ番号, イベント条件式 Xqt ファンクション名

Trap トラップ番号

パラメーター

トラップ番号

トラップ番号 (1～4の整数)を、式または直接数値で指定します。(SPEL+では同時に有効なTrapを4つまでサポートします。)

イベント条件式

トリガーとする入力状態を指定します。

[イベント] 比較演算子(=, <>, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

- 関数: Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
- 変数: Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクを掛けることや、複合的に組み合わせることができます。

- 演算子: And, Or, Xor

[例]

```
Trap 1, Sw(5) = On Call TrapFunc
Trap 1, Sw(5) = On And Sw(6) = Off Call TrapFunc
```

ラベル

Trap条件が成立した場合に、プログラムの実行を移行させる先のラベルです。

ファンクション名

Trap条件が成立したときCall又はXqtされるファンクションです。引数のあるファンクションを指定することはできません。

解説

Trapは、条件成立時にGoTo, Call, Xqtなどで指定された割込み処理を実行します。Trap条件式には、上記の関数を1つ以上含んでいなくてははいけません。

一度割込み処理が実行されると、そのTrap 設定はクリアされます。同じ割込み処理を行うためには、再度Trap 命令を実行しなくてはなりません。

Trap設定をキャンセルするためには、トラップ番号パラメーターだけを指定してTrap 命令を実行します。

[例] "Trap 3" は、Trap #3をキャンセルします。

また、Trap Gotoは宣言したファンクションから抜けると自動的にキャンセルされます。

Trap Callは宣言したタスクが終了するとキャンセルされます。

Trap Xqtはすべてのタスクが終了するまでキャンセルされません。

GoTo を指定したとき

Trapを設定したタスクで、実行中のコマンドは下記のように処理され、制御は、指定先のラベルに移行します。

- アーム動作を直ちに一時停止(クイックポーズ)します。
- WaitやInputコマンドによる待機状態を中断します。
- その他のコマンドはすべて、制御が移る前に完了します。

Callを指定したとき

GoToと同じ処理を実行した後、制御は、指定先のファンクションに移行します。

ファンクションが終了すると、プログラムは割込み発生時の次のステートメントに戻ります。

トラップ処理のファンクションにはCallを使うことはできません。

また、トラップ処理のファンクション中でエラーが発生した場合、OnErrによるエラーハンドリングは無効となり、必ずエラーが発生します。

Xqtが指定されたとき

プログラム制御は指定されたファンクションを割込み処理専用のタスクとして生成します。この場合、Trapコマンドを実行するタスクはそのまま継続されます。

割込み処理タスクからXqtでさらにタスクを実行することはできません。

注意

■ EPSON RC+ 4.xユーザーの方へ

EPSON RC+ 4.x までの Trap Callの機能は、Epson RC+ 8.0ではTrap Xqtに置き換えられています。

EPSON RC+ 4.x までの Trap GoSubの機能は、Epson RC+ 8.0では削除されました。Trap Callを使うことにより代用してください。

■ イベント条件式で変数を使用する場合

- 使用できる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が0.01秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1システム内で使用できる変数待ちの個数は最大64個です (Waitなどのイベント条件式で使用された変数待ちも含みます)。最大数を越えた場合には、プロジェクトビルド時にエラーになります。

- 変数待ちを行う変数をByrefで参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数、関数が含まれたときは、その値はTrap条件設定時に演算されます。意図しない条件になる場合があるので、整数式には変数、関数を使用しないことを推奨します。

参照

[Call](#), [GoTo](#), [Xqt](#), [SF_GetStatus](#)関数

Trap使用例

[例 1] ユーザー定義のエラー処理

Sw(0) Input は、ユーザー定義のエラー入力とします。

```
Function Main
    Trap 1, Sw(0)= On GoTo EHandle  'Trapを定義
    .
    .
    .
EHandle:
    On 31                      'シグナルタワー点灯
    OpenCom #1
    Print #1, "Error is issued"
    CloseCom #1
Fend
```

[例 2] マルチタスク的な使用

```
Function Main
    Trap 2, MemSw(0) = On Or MemSw(1) = On Call Feeder
    .
    .
    .
Fend
.

Function Feeder
    Select TRUE
        Case MemSw(0) = On
            MemOff 0
            On 2
        Case MemSw(1) = On
            MemOff 1
            On 3
    Send

    '次のサイクルのTrapを再設定
    Trap 2, MemSw(0) = On Or MemSw(1) = On Call Feeder
Fend
```

[例 3] グローバル変数をイベント条件に使用

```
Global Integer gi

Function main
    Trap 1, gi = 5 GoTo THandle
    Xqt sub
    Wait 100
    Exit Function

THandle:
    Print "IN Trap ", gi
```

```
Fend
```

```
Function sub  
  For gi = 0 To 10  
    Print gi  
    Wait 0.5  
  Next  
Fend
```

3.23.37 Trap (システム状態トリガー)

割込み、および割込み発生時の処理の定義を行います。

Trap命令を使用すると、イベント発生により、ラベルにジャンプする、または、ファンクションを呼び出すことができます。Trap命令には2つのタイプがあります。

- ユーザーの定義する入力状態をトリガーにする4つのTrap
- システム状態をトリガーにする7つのTrap

本項目ではシステム状態トリガーのTrapを説明します。

書式

Trap {Emergency | Error | Pause | SGOpen | SGClose | Abort | Finish } Xqt ファンクション名

Trap {Emergency | Error | Pause | SGOpen | SGClose | Abort | Finish }

パラメーター

Emergency

非常停止が発生したとき、指定のファンクションが実行されます。

Error

エラーが発生したとき、指定のファンクションが実行されます。

Pause

一時停止状態に入ったとき、指定のファンクションが実行されます。

SGOpen

安全扉回路が開いたとき、指定のファンクションが実行されます。

SGClose

安全扉回路が閉じたとき、指定のファンクションが実行されます。

Abort

ユーザー、またはシステムによってすべてのタスク (バックグラウンドタスクをのぞく) が停止したとき (Abort Allに相当する命令が実行されたときや、中断ボタンが押されたとき)、指定のファンクションが実行されます。

Finish

すべてのタスク (バックグラウンドタスクをのぞく) が終了したとき、指定のファンクションが実行されます。Trap Abortが実行される条件では実行されません。

ファンクション名

システム状態が成立したときXqtされる割込み処理タスクのファンクションです。引数のあるファンクションを指定することはできません。ただし、パラメーターに"Error"が指定されている場合は、3つの引数を指定できます。

注意

EPSON RC+ 4.x までの Trap *** Callの機能は、Epson RC+ 8.0ではTrap *** Xqtに置き換えられています。

解説

システム状態が成立したとき指定された割込み処理タスクを実行します。

割込み処理タスクを実行しても、そのTrap 設定はクリアされません。

Trap設定をクリアするためには、ファンクション名を省略してTrap 命令を実行します。

[例] "Trap Emergency" は、"Trap Emergency"をクリアします。

通常タスクがすべて終了してコントローラーがReady状態になるとすべてのTrap設定はクリアされます。

割り込み処理タスクからXqtでさらにタスクを実行することはできません。

注意

■ Forcedフラグ

On, OffなどのI/O出力命令に、Forcedフラグを指定することで、非常停止中, 安全扉開時, ティーチモード中, エラー発生時にもI/O出力のオンオフが可能になります。

Forcedフラグを指定するI/O出力には、アクチュエーターのような機械的動作を伴う外部機器を絶対に接続しないでください。非常停止中, 安全扉開時, ティーチモード中, エラー発生時に外部機器が動作することになり大変危険です。

Forcedフラグを指定するI/O出力は、状態表示LEDのような機械的動作をともしない外部機器が接続されることを想定しています。

■ Emergency を指定したとき

非常停止が発生したとき、指定のファンクションがNoEmgAbortタスク属性で実行されます。

割り込み処理タスクから実行可能なコマンドはNoEmgAbortタスクが実行可能なコマンドです。

非常停止の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready状態になりません。割り込み処理タスクからResetコマンドを実行し、自動的に非常停止を解除することはできません。

割り込み処理タスクからI/Oのオンオフをタスクが行う場合は、[コントローラー設定]-[環境設定]-[非常停止で出力ポートをOFF]ボックスのチェックをはずしてください。この設定がチェックされたままの場合、コントローラーによるI/OのOffとタスクによるI/OのOnのどちらが先に実行されるか保障されません。

■ Error を指定したとき

エラーが発生したとき、指定のファンクションがNoEmgAbortタスク属性で実行されます。

割り込み処理タスクから実行可能なコマンドはNoEmgAbortタスクが実行可能なコマンドです。

エラーの割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready状態になりません。

ユーザファンクションに3つの省略可能なパラメーター (エラー番号, ロボット番号, 関節番号)を指定できます。これらのパラメーターを使いたいときはトラップファンクションに、3つのbyval整数パラメーターをつけてください。

モーション系のエラーが発生した場合、エラー番号, ロボット番号, 関節番号が設定されます。

モーション系以外のエラーが発生した場合は、ロボット番号, 関節番号には"0"が設定されます。

■ Pause を指定したとき

一時停止状態に入ったとき、指定のファンクションがNoPauseタスク属性で実行されます。

■ SGOpen を指定したとき

安全扉回路が開いたとき、指定のファンクションがNoPauseタスク属性で実行されます。

- SGCclose を指定したとき

安全扉回路が閉じてラッチされたとき、指定のファンクションがNoPauseタスク属性で実行されます。

割り込み処理タスクからCont命令を実行した場合はエラーとなります。

- Abortを指定したとき

ユーザー、またはシステムによってすべてのタスク（バックグラウンドタスクをのぞく）が停止したとき（Abort Allに相当する命令が実行されたときや中断ボタンが押されたとき）、指定のファンクションがNoPauseタスク属性で実行されます。

中断の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready状態になりません。

Trap Abort で実行されるタスクにおいてエラーが発生してもTrap Errorの処理タスクは実行されません。

Restart 命令でタスクが中断された場合は、Trap Abortの処理タスクもTrap Finishの処理タスクも実行されません。

- Finishを指定したとき

すべてのタスク（バックグラウンドタスクをのぞく）が終了したとき、指定のファンクションがNoPauseタスク属性で実行されます。Trap Abortの処理タスクが実行される条件では実行されません。

終了の割り込み処理が完了したら速やかにタスクは終了するようにしてください。タスクが終了しないとコントローラーは、Ready状態になりません。

参照

[Era関数](#), [Erl関数](#), [Err関数](#), [Ert関数](#), [ErrMsg\\$関数](#), [OnErr](#), [Reset](#), [Restart](#), [SysErr関数](#), [Xqt](#)

Trap使用例

```
Function main
:
  Trap Error Xqt suberr
:
Fend

Function suberr
  Print "Error =", Err
  On ErrorSwitch
Fend

Function main

  Trap Error Xqt trapError

FEnd

Function trapError(errNum As Integer, robotNum As Integer, jointNum As Integer)
Print "error number = ", errNum
Print "robot number = ", robotNum
Print "joint number = ", jointNum
If Ert = 0 Then
  Print "system error"
Else
  Print "task error"
  Print "function = ", Erf$(Ert)
  Print "line number = ", Erl(Ert)
EndIf
FEnd
```


3.23.38 Trim\$関数

前後に半角スペースを含まずに、指定された文字列と同じ文字列を返します。

書式

Trim\$(文字列)

パラメーター

文字列

文字列式を指定します。

戻り値

指定された文字列が前後に半角スペースを含む場合、半角スペースは削除されます。

参照

[LTrim\\$関数](#), [RTrim\\$関数](#)

Trim\$関数使用例

```
str$ = "  data  "
str$ = Trim$(str$)   ' str$ = "data"
```

3.23.39 TW関数

Wait命令, WaitNet命令, WaitSig命令の状態を返します。

書式

TW

戻り値

時間内にWait状態が成立した場合は"False"を返します。

タイムアウトが発生した場合は"True"を返します。

解説

直前のWait命令の条件が成立した場合は"False"、タイムアウトが発生した場合は"True"を返します。

参照

[TMOOut](#), [Wait](#), [Hand_TW](#)

ハンド制御コマンドについては、以下のマニュアルを参照してください。

"Hand機能"

TW関数使用例

```
Wait Sw(0) = On, 5      '入力ビット0がオンになるまで5秒間待機
If TW = True Then
    Print "Time Up"      '5秒以上の場合は"Time Up"と表示
EndIf
```

3.24 U

3.24.1 UBound関数

指定した配列に設定できる要素番号の最大値を返します。

書式

UBound (配列変数名 [, 次元])

パラメーター

配列変数名

通常の変数名の名前のつけ方にしたがって、名前をつけます。

次元

要素番号の最大値を返す次元を、下記の整数値で設定します。省略可能で、省略すると、"1"と仮定されます。

- 1: 1番目の次元
- 2: 2番目の次元
- 3: 3番目の次元

参照

[Redim](#)

UBound関数使用例

```
Integer i, a(10)

For i = 0 to UBound(a)
    a(i) = i
Next
```

3.24.2 UByte

UByte型の変数を宣言します。(符号なし整数型、サイズ: 2バイト)

書式

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

パラメーター

変数名

UByte型と宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

UByteは、変数をUByte型と宣言するときに使います。UByte変数の範囲は、0～255までです。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UInt32](#), [UInt64](#), [UShort](#)

UByte使用例

次はUByteを使って、Bit7の値を表示するプログラム例です。

```
Function main
    UByte varByte

    varByte = 127
    Call BitCheckUByte(varByte)

    varByte = 255
    Call BitCheckUByte(varByte)
Fend

Function BitCheckUByte(var As UByte)
    Print "Value:", var, " Bit7 =", BTst(var, 7)
Fend
```

[出力結果]

```
Value:127 Bit7 = 0
Value:255 Bit7 = 1
```

3.24.3 UCase\$関数

文字列を小文字から大文字化して返します。

書式

UCase\$(文字列)

パラメーター

文字列

大文字化する文字列を指定します。

戻り値

大文字化した文字列を返します。

参照

[LCase\\$関数](#), [LTrim\\$関数](#), [Trim\\$関数](#), [RTrim\\$関数](#)

UCase\$関数使用例

```
str$ = "Data"  
str$ = UCase$(str$)   ' str$ = "DATA"
```

3.24.4 UInt32

UInt32型の変数を宣言します。(符号なし4バイト整数型変数)

書式

UInt32 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

UInt32は整数型変数の宣言をします。整数型変数の範囲は0～4294967295です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt64](#), [UShort](#)

UInt32使用例

次はUInt32を使って整数型の変数を宣言するプログラム例です。

```
Function uint32test
    UInt32 A(10)           'UInt32型の1次元配列
    UInt32 B(10, 10)       'UInt32型の2次元配列
    UInt32 C(5, 5, 5)      'UInt32型の3次元配列
    UInt32 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

3.24.5 UInt64

UInt64型の変数を宣言します。(符号なし8バイト整数型変数)

書式

UInt64 変数名 [(配列変数の最大要素番号)] [, 変数名 [(配列変数の最大要素番号)]...]

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

UInt64は整数型変数の宣言をします。整数型変数の範囲は0～18446744073709551615です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UShort](#)

UInt64使用例

次はUInt64を使って整数型の変数を宣言するプログラム例です。

```
Function uint64test
    UInt64 A(10)           'UInt64型の1次元配列
    UInt64 B(10, 10)       'UInt64型の2次元配列
    UInt64 C(5, 5, 5)      'UInt64型の3次元配列
    UInt64 var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

3.24.6 UOpen

読み出しと書き込みの両方のモードでファイルを開きます。

書式

UOpen ファイル名As #ファイル番号 .. Close #ファイル番号

パラメーター

ファイル名

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

ファイル番号

30～63の整数値または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは指定したファイルに書き込んだり、データを読み出しするときに使います。

注意

ネットワークパスが使用できます。

存在しないファイルを指定すると、そのファイルを作成し、データを書き込みます。存在するファイルを指定すると、既存データの先頭から、データを読み書きします。

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

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

Closeステートメントでファイルを閉じ、ファイル番号を解放します。

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

参照

[Close](#), [Print #](#), [Input #](#), [AOpen](#), [BOpen](#), [ROpen](#), [WOpen](#), [FreeFile関数](#), [Seek](#)

UOpen使用例

```
Integer fileNum, i, j

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

fileNum = FreeFile
UOpen "TEST.DAT" As #fileNum
Seek #fileNum, 10
Input #fileNum, j
Print "data = ", j
Close #fileNum
```


3.24.7 UpdateDB

オープンしているデータベース内の検索されたテーブルのデータを更新します。

書式

UpdateDB #データベース番号, 項目, 値

パラメーター

データベース番号

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

項目

更新するテーブルの項目名を指定します。

値

更新する値を指定します。

解説

オープンされたデータベースのSelectされたテーブルのデータを指定された項目の値で更新します。

データの更新前に必ずSelectDBを実行し、更新するレコードを選択する必要があります。

注意

RC+ がインストールされているPCの接続が必要です

参照

[OpenDB](#), [CloseDB](#), [SelectDB関数](#), [DeleteDB](#)

UpdateDB使用例

SQLデータベースを使用した例 下記に、SQL サーバー2000のサンプルデータベースNorthwindのテーブルEmployeesにデータを登録し、登録したデータの項目を更新する簡単な例を示します。

```
Integer count, i, eid
String Lastname$, Firstname$, Title$

OpenDB #501, SQL, "(LOCAL)", "Northwind"
count = SelectDB(#501, "Employees", "TitleOfCourtesy = 'Mr.'")
Print #501, "Epson", "Taro", "Engineer", "Mr."
count = SelectDB(#501, "Employees", "LastName = 'Epson' and      FirstName =
'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
UpdateDB #501, "Title", "Chief Engineer"
count = SelectDB(#501, "Employees", "LastName = 'Epson' and      FirstName =
'Taro'")
Input #501, eid, Lastname$, Firstname$, Title$
Print eid, ",", Lastname$, ",", Firstname$, ",", Title$
CloseDB #501
```

3.24.8 UserErrorDef関数

指定したユーザーエラー番号またはユーザーエラーラベルが定義されているかを返します。

書式

UserErrorDef (ユーザーエラー番号)

UserErrorDef (ユーザーエラーラベル)

パラメーター

- ユーザーエラー番号: ユーザーエラー番号を表す整数値
- ユーザーエラーラベル: ユーザーエラーラベルを指定する文字列式

戻り値

指定したユーザーエラー番号またはユーザーエラーラベルが定義されている場合は"True"、それ以外は"False"を返します。

参照

[UserErrorLabel\\$関数](#), [UserErrorNumber関数](#)

UserErrorDef関数使用例

```
Integer i
For i = 8950 To 8999
  If UserErrorDef(i) = TRUE Then
    Print "UserError " , i, " is defined"
  Else
    Print "UserError " , i, " is undefined"
  EndIf
Next i
```

3.24.9 UserErrorLabel\$関数

指定したユーザーエラー番号のユーザーエラーラベルを返します。

書式

UserErrorLabel\$(ユーザーエラー番号)

パラメーター

- ユーザーエラー番号: ユーザーエラー番号を表す整数値

戻り値

ラベルを含む文字列を返します。

参照

[UserErrorDef関数](#), [UserErrorNumber関数](#)

UserErrorLabel\$関数使用例

```
Integer i
For i = 8950 To 8999
    Print "Input ", i, ": ", UserErrorLabel$(i)
Next i
```

3.24.10 UserErrorNumber関数

指定したユーザーエラーラベルのユーザーエラー番号を返します。

書式

UserErrorNumber (ユーザーエラーラベル)

パラメーター

- ユーザーエラーラベル: ユーザーエラーラベルを指定する文字列式

参照

[UserErrorDef関数](#), [UserErrorLabel\\$関数](#)

UserErrorNumber関数使用例

```
Integer i  
i = UserErrorNumber("ERR_OUTOFRANGE")
```

3.24.11 UShort

UShort型の変数を宣言します。(符号なし2バイト整数型変数)

書式

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

パラメーター

変数名

変数宣言する変数名を指定します。

配列変数の最大要素番号

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

(最大要素番号1, [最大要素番号2], [最大要素番号3])

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

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

解説

UShortは整数型変数の宣言をします。整数型変数の範囲は0～65535です。ローカル変数は、Functionの先頭で宣言します。グローバル変数とモジュール変数は、Functionの外で宣言します。

参照

[Boolean](#), [Byte](#), [Double](#), [Global](#), [Int32](#), [Int64](#), [Integer](#), [Long](#), [Real](#), [Short](#), [String](#), [UByte](#), [UInt32](#), [UInt64](#)

UShort使用例

次はUShortを使って整数型の変数を宣言するプログラム例です。

```
Function ushorttest
    UShort A(10)           'UShort型の1次元配列
    UShort B(10, 10)       'UShort型の2次元配列
    UShort C(5, 5, 5)      'UShort型の3次元配列
    UShort var1, arrayvar(10)
    Integer i
    Print "Please enter an Integer Number"
    Input var1
    Print "The Integer variable var1 = ", var1
    For i = 1 To 5
        Print "Please enter an Integer Number"
        Input arrayvar(i)
        Print "Value Entered was ", arrayvar(i)
    Next i
Fend
```

3.25 V

3.25.1 Val関数

数字からなる指定文字列を数値に変換し、その値を返します。

書式

Val(文字列)

パラメーター

文字列

数字だけからなる文字列式を指定します。文字列にはプレフィックスも含まれます
&H (16進法), &O (8進法), &B (2進法)

戻り値

整数、または入力された文字列式による浮動少数点数を返します。小数点を含む入力文字列式は、浮動少数点数に変換されます。それ以外で、戻り値は、整数値が返されます。

解説

Valは、数字からなる文字列式を数値に変換します。結果は、整数または浮動少数点数です。Val命令に小数点を含む入力文字列式が与えられると、浮動少数点数が返されます。それ以外は、整数値が返されます。

参照

[Abs関数](#), [Asc関数](#), [Chr\\$関数](#), [Int関数](#), [Left\\$関数](#), [Len関数](#), [Mid\\$関数](#), [Mod](#) 演算子, [Right\\$関数](#), [Sgn関数](#), [Space\\$関数](#), [Str\\$関数](#)

Val関数使用例

下記は、いくつかの異なる文字列式を数値に変換し、その結果を画面に表示するプログラムの例です。

```
Function ValDemo
  String realstr$, intstr$
  Real realsqr, realvar
  Integer intsqr, intvar

  realstr$ = "2.5"
  realvar = Val(realstr$)
  realsqr = realvar * realvar
  Print "The value of ", realstr$, " squared is: ", realsqr

  intstr$ = "25"
  intvar = Val(intstr$)
  intsqr = intvar * intvar
  Print "The value of ", intstr$, " squared is: ", intsqr
Fend
```

下記は、コマンドウィンドウからの操作例です。

```
> Print Val("25.999")
25.999
>
```

3.25.2 VSD

スカラロボットの可変速度CP動作機能を設定します。

書式

VSD { ON | Off }

パラメーター

On | Off

- On: スカラロボットの可変速度CP動作機能を有効にします。
- Off: スカラロボットの可変速度CP動作機能を無効にします。

解説

VSDは下記の命令で有効です。

Move, Arc, Arc3

本命令は、スカラ型ロボットのみ有効です。スカラ型ロボット以外の機種では、AvoidSingularity SING_VSDを使用してください。

可変速度CP動作機能は、スカラロボットがCP動作実行中に、加速度エラーや過速度エラーが発生するのを防止します。動作軌跡を維持したまま、関節速度を自動的に制限して動作する機能です。関節速度が制限された場合、SpeedSで設定されたツール先端速度は維持されませんが、関節速度が制限を下回ると元のツール先端速度に回復します。等速度が優先される場合には、AccelS, DecelS, SpeedSを小さくして、エラーの発生をなくしてください。

VSDを使用しても加速度エラーや過速度エラーが発生する場合は、AccelS, DecelS, SpeedSを小さくしてください。

VSDの設定値を変更した場合、次回コントローラー起動時まで有効です。

コントローラー起動時は、VSDはオフになります。

参照

[VSD関数](#)

VSD使用例

```
VSD On '可変速度CP動作を有効にして動作させる'
Move P1
Move P2
VSD Off
```

3.25.3 VSD関数

スカラロボットの可変速度CP動作機能の設定を返します。

書式

VSD

戻り値

- On = 可変速度CP動作機能が有効
- Off = 可変速度CP動作機能が無効

参照

[VSD](#)

VSD関数使用例

```
If VSD = Off Then
    Print "Variable Speed Drive is off"
EndIf
```


3.25.4 VxCalib

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータを作成します。

書式

(1) VxCalib CalNo

(2) VxCalib CalNo, CamOrient, P(pixel_st : pixel_ed), P(robot_st : robot_ed) [, TwoRefPoints]

(3) VxCalib CalNo, CamOrient, P(pixel_st : pixel_ed), P(robot_st : robot_ed), P(ref0) [, P(ref180)]

パラメーター

CalNo

キャリブレーションデータの番号を整数値で指定します。0～15の整数で、合計16まで定義することができます。

CamOrient

カメラ取付方向を下記の整数値で指定します。1～3は、書式 (2)でのみ指定できます。4～7は、書式 (3)でのみ指定できます。

- 1: 固定カメラ
- 2: 下向き固定カメラ
- 3: 上向き固定カメラ
- 4: 可動カメラ 第2軸搭載
- 5: 可動カメラ 第4軸搭載
- 6: 可動カメラ 第5軸搭載
- 7: 可動カメラ 第6軸搭載

P(pixel_st : pixel_ed)

ピクセル座標 (X, Yのみ)を連続ポイントデータで指定します。

P(robot_st : robot_ed)

ロボット座標を連続ポイントデータで指定します。指定するポイントデータは、CamOrientで指定するカメラ取付方向によって異なります。CamOrient = 1～3の場合、ロボット座標は、現在のTOOLとARMで設定してください。CamOrient = 4～7の場合、ロボット座標は、TOOL: 0、ARM: 0で設定してください。

TwoRefPoints

書式 (1)の場合に、指定できます。2つの測定ポイントを使用する場合は"True"、1つの測定ポイントを使用する場合は"False"を指定します。2つの測定ポイントを指定することにより、より正確なキャリブレーションを実施することができます。デフォルトは"False"です。省略可能です。

P(ref0)

書式 (3)の場合に、指定できます。基準点のロボット座標をポイントデータで指定します。

P(ref180)

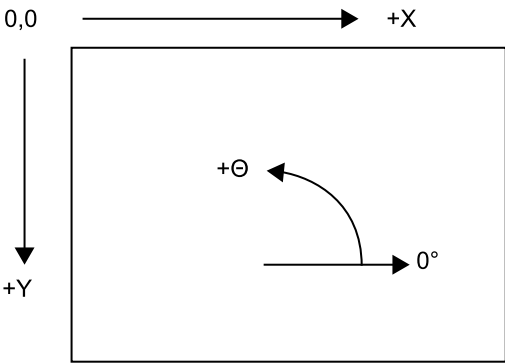
書式 (3)の場合に、指定できます。2つめの基準点のロボット座標をポイントデータで指定します。基準点を2つ指定することにより、より正確なキャリブレーションを実施することができます。省略可能です。

解説

指定したカメラ姿勢を使った指定キャリブレーション番号、引数で渡されたピクセル座標, ロボット座標, 基準点 (可動カメラのみ)から、ビジョンキャリブレーションデータを計算します。

CalNoのみを指定した場合、定義時のポイントデータなどが表示されます (コマンドウィンドウのみ)。

ピクセル座標の座標系は、次の表のようになります。単位はピクセルです。



ピクセル座標, ロボット座標は、画面上の上左位置をポイント1とし、以下表の順番にしたがい、下右位置がポイント9になるように設定してください。CamOrient, TwoRefPoints引数により、4種類に分類されます。

1) CamOrient = 1～3 (固定, 下向き固定, 上向き固定)、TwoRefPoints = Falseの場合

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標1	測定点座標1
2	上中	検出座標2	測定点座標2
3	上右	検出座標3	測定点座標3
4	中右	検出座標4	測定点座標4
5	中中	検出座標5	測定点座標5
6	中左	検出座標6	測定点座標6
7	下左	検出座標7	測定点座標7
8	下中	検出座標8	測定点座標8
9	下右	検出座標9	測定点座標9

2) CamOrient = 2 (下向き固定), TwoRefPoints = Trueの場合

ツールを正確に定義してある場合、TwoRefPointsは必要ありません。"False"に設定してください。

TwoRefPointsを"True"にすると、2つの測定ポイントを使用するため、より正確なキャリブレーションを実施することができます。

ロボット座標のみ、18ポイントが必要になります。

1～9の測定点座標を設定した後、U軸を180度回転し、ハンド (ロッドなど)をキャリブレーションターゲット位置に合わせた測定点座標10～18を設定してください。

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標1	測定点座標1
2	上中	検出座標2	測定点座標2
3	上右	検出座標3	測定点座標3

データ順序	画面位置	ピクセル座標	ロボット座標
4	中右	検出座標4	測定点座標4
5	中中	検出座標5	測定点座標5
6	中左	検出座標6	測定点座標6
7	下左	検出座標7	測定点座標7
8	下中	検出座標8	測定点座標8
9	下右	検出座標9	測定点座標9
10	上左	---	測定点座標10
11	上中	---	測定点座標11
12	上右	---	測定点座標12
13	中右	---	測定点座標13
14	中中	---	測定点座標14
15	中左	---	測定点座標15
16	下左	---	測定点座標16
17	下中	---	測定点座標17
18	下右	---	測定点座標18

3) CamOrient = 3 (上向き固定), TwoRefPoints = Trueの場合

ツールを正確に定義してある場合、TwoRefPointsは必要ありません。"False"に設定してください。

TwoRefPointsを"True"にすると、2つの検出ポイントを使用するため、より正確なキャリブレーションを実施することができます。

ピクセル座標のみ、18ポイントが必要になります。

各測定点座標で1～9の検出座標を設定した後、U軸を180度回転した位置の検出座標10～18を設定してください。

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標1	測定点座標1
2	上中	検出座標2	測定点座標2
3	上右	検出座標3	測定点座標3
4	中右	検出座標4	測定点座標4
5	中中	検出座標5	測定点座標5
6	中左	検出座標6	測定点座標6

データ順序	画面位置	ピクセル座標	ロボット座標
7	下左	検出座標7	測定点座標7
8	下中	検出座標8	測定点座標8
9	下右	検出座標9	測定点座標9
10	上左	検出座標10	---
11	上中	検出座標11	---
12	上右	検出座標12	---
13	中右	検出座標13	---
14	中中	検出座標14	---
15	中左	検出座標15	---
16	下左	検出座標16	---
17	下中	検出座標17	---
18	下右	検出座標18	---

4) CamOrient = 4~7の場合

データ順序	画面位置	ピクセル座標	ロボット座標
1	上左	検出座標1	測定点座標1
2	上中	検出座標2	測定点座標2
3	上右	検出座標3	測定点座標3
4	中右	検出座標4	測定点座標4
5	中中	検出座標5	測定点座標5
6	中左	検出座標6	測定点座標6
7	下左	検出座標7	測定点座標7
8	下中	検出座標8	測定点座標8
9	下右	検出座標9	測定点座標9

注意

- 上表の他に、基準点のロボット座標を指定してください。

2つの基準点を使用すると、より正確なキャリブレーションを実施することができます。その場合、U軸が180°異なる2ポイントが必要になります。

1つめの基準点座標を設定した後、U軸を180度回転し、ハンド（ロッドなど）をキャリブレーションターゲット位置に合わせた2つめの基準点座標を設定してください。ツールを正確に定義してある場合、2つの基準点を使用する必要はありません。

参照

[VxTrans関数](#), [VxCalInfo関数](#), [VxCalDelete](#), [VxCalSave](#), [VxCalLoad](#)

VxCalib使用例

```
Function MobileJ2

  Integer i
  Double d(8)

  Robot 1
  LoadPoints "MobileJ2.pts"

  VxCalib 0, 4, P(21:29), P(1:9), P(0)

  If (VxCalInfo(0, 1) = True) Then
    For i = 0 To 7
      d(i) = VxCalInfo(0, i + 2)
    Next i
    Print "Calibration result:"
    Print d(0), d(1), d(2), d(3), d(4), d(5), d(6), d(7)

    P52 = VxTrans(0, P51, P50)
    Print "Coordinates conversion result:"
    Print P52
    SavePoints "MobileJ2.pts"
    VxCalSave "MobileJ2.caa"
  Else
    Print "Calibration failed"
  EndIf

Fend
```

3.25.5 VxCalDelete

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータを削除します。

書式

VxCalDelete CalNo

パラメーター

CalNo

キャリブレーションデータの番号を整数値で指定します。0～15の整数で、合計16まで定義することができます。

解説

指定したキャリブレーション番号で定義してあるキャリブレーションデータを削除します。

参照

[VxCalib](#), [VxTrans関数](#), [VxCalInfo関数](#), [VxCalSave](#), [VxCalLoad](#)

VxCalDelete使用例

```
VxCalDelete "MobileJ2.caa"
```

3.25.6 VxCalLoad

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータをファイルから読み込みます。

書式

VxCalLoad FileName

パラメーター

FileName

キャリブレーションデータを読み込むファイル名を文字列式で指定します。拡張子は、".caa"固定です。省略した場合、".caa"が追加されます。".caa"以外の場合、".caa"に変換されます。パスを指定することはできません。

解説

カレントプロジェクト内にある、指定したファイルからキャリブレーションデータを読み込みます。

参照

[VxCalib](#), [VxTrans関数](#), [VxCalInfo関数](#), [VxCalDelete](#), [VxCalSave](#)

VxCalLoad使用例

```
VxCalLoad "MobileJ2.caa"
```

3.25.7 VxCalInfo関数

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションの終了状態, キャリブレーションデータを返します。

書式
VxCalInfo (CalNo,CalData)

パラメーター

CalNo
キャリブレーションデータの番号を整数値で指定します。0～15の整数で、合計16まで定義することができます。

CalData
取得したいキャリブレーションデータ種類を下表の整数値で指定します。

CalData	キャリブレーションデータ種類
1	キャリブレーションの終了状態
2	X方向の平均偏差 [mm]
3	X方向の最大偏差 [mm]
4	X方向の1ピクセル当たりの長さ [mm]
5	X方向の傾き
6	Y方向の平均偏差 [mm]
7	Y方向の最大偏差 [mm]
8	Y方向の1ピクセル当たりの長さ [mm]
9	Y方向の傾き

戻り値
指定されたキャリブレーションデータを、CalData = 1の場合はBool型値で、CalData = 2～9の場合はDouble型値で返します。

解説
どのキャリブレーション番号でキャリブレーションデータが定義されているか確認することができます。

キャリブレーションデータの値を取得することもできます。

参照
[VxCalib](#), [VxTrans関数](#), [VxCalDelete](#), [VxCalSave](#), [VxCalLoad](#)

VxCalInfo関数使用例

```
Print VxCalInfo(0, 1)
```


3.25.8 VxCalSave

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

お客様がご用意したビジョンシステムのキャリブレーションデータをファイルへ保存します。

書式

VxCalSave FileName

パラメーター

FileName

キャリブレーションデータを保存するファイル名を文字列式で指定します。拡張子は、".caa"固定です。省略した場合、".caa"が自動的に追加されます。".caa"以外の場合、".caa"に自動的に変換されます。パスを指定することはできません。

解説

指定したファイル名でキャリブレーションデータを保存します。ファイルはカレントプロジェクト内に保存されます。同じ名前のファイルが存在する場合、キャリブレーションデータは上書きされます。

参照

[VxCalib](#), [VxTrans関数](#), [VxCalInfo関数](#), [VxCalDelete](#), [VxCalLoad](#)

VxCalSave使用例

```
VxCalSave "MobileJ2.caa"
```

3.25.9 VxTrans関数

このコマンドは、Vision Guide以外の、お客様がご用意したビジョンシステムでご使用ください。

ピクセル座標からロボット座標への座標変換を行い、変換したポイントデータを返します。

書式

VxTrans (CalNo, P(pixel)[, P(camRobot)]) As Pose

パラメーター

CalNo

キャリブレーションデータの番号を整数値で指定します。0～15の整数で、合計16まで定義することができます。

P(pixel)

ビジョンピクセル座標 (X, Y, Uのみ)をポイントデータで指定します。

P(camRobot)

省略可能。可動カメラの場合、撮像時のロボットの位置を指定します。省略した場合、ロボットの現在位置が使用されます。ポイントデータは、TOOL: 0, ARM: 0で設定してください。

戻り値

算出されたロボット座標をポイントデータで返します。

解説

指定したキャリブレーション番号のキャリブレーションデータを使用し、ピクセル座標からロボット座標への座標変換を行います。

可動カメラで現在値以外を指定する場合、撮像時のロボット座標P(camRobot)を指定します。P(camRobot)はTOOL: 0、ARM: 0で設定してください。設定したロボット座標の第4関節の角度, 第6関節の角度が計算に使用されます。

参照

[VxCalib](#), [VxCalInfo関数](#), [VxCalDelete](#), [VxCalSave](#), [VxCalLoad](#)

VxTrans関数使用例

```
P52 = VxTrans (0, P51, P50)
```

3.26 W

3.26.1 Wait

MemSwやSwを使用して、指定したイベント条件が成立するまで、または指定時間の間、プログラムを待機します。(SwのかわりにOportを使って、I/O出力をチェックすることもできます。)

グローバル変数の値が変化するのが待機することもできます。

書式

(1) Wait 時間

(2) Wait イベント条件式

(3) Wait イベント条件式, 時間

パラメーター

時間

待機時間を0～2147483 の実数 (単位: 秒)で指定します。最小有効桁は0.01秒です。

イベント条件式

以下の書式でイベント条件を指定します。

[イベント] 比較演算子(=, <>, >=, >, <, <=) [整数式]

次の関数や変数をイベントに使うことができます。

- 関数: Sw, In, InW, Oport, Out, OutW, MemSw, MemIn, MemInW, Ctr, GetRobotInsideBox, GetRobotInsidePlane, MCalComplete, Motor, LOF, ErrorOn, SafetyOn, EstopOn, TeachOn, Cnv_QueueLen, WindowsStatus, AtHome, LatchState, WorkQueue_Len, PauseOn, AIO_In, AIO_InW, AIO_Out, AIO_OutW, Hand_On, Hand_Off, SF_GetStatus
- 変数: Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort 型のバックアップ変数, グローバル変数, モジュール変数

また、以下の演算子で複数のイベント条件式にマスクをかけることや、複合的に組み合わせることができます。

- 演算子: And, Or, Xor, Mask

解説

(1) 時間だけを指定した場合

タイマーとしてWait命令を使うときは、指定時間だけプログラムを一時停止させ、その後プログラムの実行を継続します。

(2) イベント条件式だけを指定した場合

条件的インターロック機能としてWait 命令を使うときは、指定されたイベント条件が成立するまで、プログラムを待機します。TMout命令でタイムアウト指定している場合は、その指定時間が経過してもイベント条件式が成立しないとエラーになります。And, Mask, Or, またはXor命令などを用いて、1つのWait命令に対し複数の条件をチェックできます。使用例を参照してください。

(3) 時間とイベント条件式を指定した場合

イベント条件式と時間を指定したときは、条件が成立するか、指定時間が経過するかの、どちらかが成立したときに次のコマンドを実行します。Tw関数を使って、イベント条件式が成立したか、指定時間が経過したかを確認することができます。

注意

■ Waitにタイムアウトを併用する場合

Wait命令に待機時間の指定がないとき、タイムアウトを指定すると指定された状態を待つ制限時間を設定することができます。タイムアウトは、TMOOut命令を使って指定します。詳細は、このTMOOut命令の項を参照してください。(TMOOut命令のデフォルト値は、"0"で、これは時間制限がないことを意味します。)

■ Waitで変数待ちを行う場合

- 変数待ちできる変数型は整数型 (Byte, Int32, Integer, Long, Short, UByte, UInt32, UShort)です。
- 配列変数は、使用できません。
- ローカル変数は、使用できません。
- 変数値が0.01秒以上の間、イベント条件を満足していない場合、変数の変化をシステムが検出できない場合があります。
- システム内で使用できる変数待ちの数に制限があります。1システム内で使用できる変数待ちの個数は最大64個です (Tillなどのイベント条件式で使用された変数待ちも含みます)。最大数を超えた場合には、プロジェクトビルド時にエラーになります。
- 変数待ちを行う変数をByrefで参照渡しするとエラーになります。
- イベント条件式の右辺の整数式に変数または関数が含まれたときは、その値はWait条件設定時に演算されます。意図しない条件になる場合があるので、整数式には変数または関数を使用しないことを推奨します。

■ PC COMポート (1001~1008)を使用している場合

Wait命令でLof関数を使用することはできません。

■ Wait実行中に、プログラムが一時停止した場合

Wait命令実行中に、プログラムが一時停止 (Pause状態)となった場合でも、Wait命令は停止しません。イベント条件式が成立するか、指定時間が経過すると、Wait命令は終了します。

Wait命令で時間が指定されている場合、指定時間が経過する前に、継続実行でプログラムを再開すると、それまでの経過時間はリセットされ、プログラムは、指定された待機時間の間、待機します。

参照

[AtHome関数](#), [Cnv_QueueLen関数](#), [Ctr関数](#), [ErrorOn関数](#), [EStopOn関数](#), [GetRobotInsideBox関数](#), [GetRobotInsidePlane関数](#), [In関数](#), [InW関数](#), [LatchState関数](#), [Lof関数](#), [Mask](#) 演算子, [MCalComplete関数](#), [MemIn関数](#), [MemInW関数](#), [MemSw関数](#) Motor, Oport, Out, OutW, PauseOn, SafetyOn, Sw, TeachOn, TMOOut, WindowsStatus, Tw, WorkQueue_Len, SF_GetStatus

Wait使用例

下記例では、それぞれ動作命令を起動できる2つのタスクに対し、他方がロボット制御をしていないときに限ってロボット制御できるインターロック機能が働いています。これにより、それぞれのタスクが指定された動作を順序よく行うことができます。MemSwを併用したWait命令が、メモリーI/Oビット1が適切な値になるまでプログラムを待機し、安全になってから動作を再開させます。

```
Function main
  Integer I
  MemOff 1
  Xqt !2, task2
  For I = 1 to 100
    Wait MemSw(1) = Off
    Go P(i)
    MemOn 1
  Next I
Fend
```

```
Function task2
  Integer i
  For i = 101 to 200
    Wait MemSw(1) = On
    Go P(i)
    MemOff 1
  Next i
Fend

'入力0オンまで待機
Wait Sw(0) = On

'60.5秒待機した後に継続実行
Wait 60.5

'入力0がオフになり入力1がオンになるまで待機
Wait Sw(0) = Off And Sw(1) = On

'メモリービット0がオンまたはメモリービット1がオンになるまで待機
Wait MemSw(0) = On Or MemSw(1) = On

'1秒待機して出力1をオン
Wait 1; On 1

'入力ポート0の下位3ビットが1になるまで待機
Wait In(0) Mask 7 = 1

'グローバルInteger型の変数giCounterの値が10を超えるまで待機
Wait giCounter > 10

'グローバルLong型の変数glCheckの値が30000になるまで10秒間待機
Wait glCheck = 30000, 10
```

3.26.2 WaitNet

TCP/IPポートの接続が確立するまで待ちます。

書式

WaitNet # ポート番号 [, タイムアウト時間]

パラメーター

ポート番号

接続待ちするTCP/IPのポートの番号を201～216の整数値で指定します。

タイムアウト時間

最大待ち時間(秒)を実数値で指定します。省略可能です。

参照

[OpenNet](#), [CloseNet](#)

WaitNet使用例

次は、2つのコントローラーのTCP/IP設定の例です。

Controller #1:

Port: #201

Host Name: 192.168.0.2

TCP/IP Port: 1000

```
Function tcpip
  OpenNet #201 As Server
  WaitNet #201
  Print #201, "Data from host 1"
Fend
```

Controller #2:

Port: #201

Host Name: 192.168.0.1

TCP/IP Port: 1000

```
Function tcpip
  String data$
  OpenNet #201 As Client
  WaitNet #201
  Input #201, data$
  Print "received '", data$, "' from host 1"
Fend
```

3.26.3 WaitPos

パスモーションが有効な状態でも、次のステートメントを実行する前にロボットが減速停止するのを待ちます。

書式

WaitPos

解説

通常、パスモーションが有効な状態 (CP OnやCPパラメーター指定) の場合、動作命令は減速開始と同時に制御を次のステートメントに移します。

しかし、直後にWaitPos命令を挿入しておくと、減速動作を完了させてから次の動作に移ります。

参照

[Wait](#), [WaitSig](#), [CP](#)

WaitPos使用例

```
Off 1
CP On
Move P1
Move P2
WaitPos   ' ロボットが減速するまで待機
On 1
CP Off
```

3.26.4 WaitSig

他のタスクからのSignal命令による同期シグナルを待ちます。

書式

WaitSig シグナル番号 [, タイムアウト時間]

パラメーター

シグナル番号

受信するシグナル番号を整数値 (0～63)で指定します。

タイムアウト時間

最大待ち時間(秒)を実数値で指定します。省略可能です。

解説

他のタスクからのシグナルを待ちます。WaitSig実行後にシグナル待ちとなり、それ以前のシグナルは無視されます。

参照

[Wait](#), [WaitPos](#), [Signal](#)

WaitSig使用例

```
Function Main
  Xqt SubTask
  Wait 1
  Signal 1
  .
  .
Fend

Function SubTask
  WaitSig 1
  Print "signal received"
  .
Fend
```


3.26.5 Weight

PTP動作時の、速度、加減速度を補正するためのパラメーター設定と表示をします。

書式

Weight [ハンド質量 [, {アーム長さ | S | T }]]

Weight

パラメーター

ハンド質量

アームにかかるハンドの質量を指定します。(単位: kg 小数点以下3桁)省略可能 (ハンド質量のみを省略することは不可能)です。

アーム長さ

水平多関節型ロボット (RSシリーズを含む)のみ有効です。第2関節中心から、第3関節中心までを指定します。(単位: mm) 省略可能です。

S

付加軸S関節に対する負荷質量を指定します。(単位: kg 小数点以下3桁)

T

付加軸T関節に対する負荷質量を指定します。(単位: kg 小数点以下3桁)

結果

パラメーターを省略すると、現在のWeight設定値を表示します。

[アーム長さ]を省略すると、入力された [ハンド質量]が設定され、デフォルト値の [アーム長さ]が設定されます。

[ハンド質量]のみを省略することはできません。

解説

PTP動作における最大加減速度を演算するためのパラメーターを指定します。Weight命令では、ハンドとワークの質量を設定します。アーム長指定は、水平多関節型ロボット (RSシリーズを含む)の場合のみ必要となります。これは、第2関節中心から第3関節中心までの距離です。水平多関節型ロボット (RSシリーズを含む)以外では無効になります。ロボットに付加軸が設定されている場合、S, Tパラメーターを使って付加軸が持つ負荷質量を個別に設定する必要があります。

設定した値から計算される等価質量が最大可搬質量を超えると、エラーになります。

"負荷, イナーシャ, 偏心/オフセット測定ユーティリティ"による設定も可能です。

詳細は、以下のマニュアルに記載しています。

"Epson RC+ ユーザーズガイド - 負荷, イナーシャ, 偏心/オフセット測定ユーティリティ"

起こりやすいエラー

■ 最大許容負荷質量をこえた場合

設定した値から計算される等価質量が最大許容負荷質量を超えると、エラーになります。

■ マニピュレーターアームへの損傷

Weightのハンド質量を、実際の質量からあまりにかけ離れて軽い値で設定すると、過剰な加速値および減速値が設定され、マニピュレーターに損傷を与えることがありますので注意してください。

注意

- Weight の設定値は、電源オフでも変更されません

Weightの設定値は、一度設定するとコントローラーに記憶されます。電源をオフしても変更されません。

何も設定しない場合は、前回設定された値のままになります。

参照

[Accel](#), [Inertia](#)

ハンドについては、以下のマニュアルを参照してください。

"Hand機能"

Weight使用例

コマンドウィンドウからWeight 命令で現在の設定値を表示させる例です。

```
> weight
2.000, 200.000
>
```

Weight命令でハンドの質量 (3 kg)を設定する例です。

```
Weight 3.0
```

Weight命令で付加軸Sの負荷質量 (30 kg)を設定する例です。

```
Weight 30.0, S
```

3.26.6 Weight関数

Weight命令で設定したハンド質量とアーム長さを返します。

書式

Weight(パラメーター番号)

パラメーター

パラメーター番号

パラメーター番号を次に示す整数値で設定します。

- 1: ハンド質量
- 2: アーム長
- 3: 付加軸S関節負荷質量
- 4: 付加軸T関節負荷質量

戻り値

パラメーターを実数値で返します。

参照

[Inertia](#), [Weight](#)

ハンドについては、以下のマニュアルを参照してください。

"Hand機能"

Weight関数使用例

```
Print "The current Weight parameters are: ", Weight(1)
```

3.26.7 Where

ロボットの現在の位置データを表示します。

書式

Where [ローカル番号]

パラメーター

ローカル番号

ローカル座標系番号を指定します。デフォルトは "Local 0" です。省略可能です。

参照

[Joint](#), [PList](#), [Pulse](#)

Where使用例

表示される形式は、ロボットのタイプや付加軸の有無で変わります。

以下の例はスカラ型ロボットで付加軸無しの場合です。

```
>where
WORLD: X: 350.000 mm Y: 0.000 mm Z: 0.000 mm U: 0.000 deg V: 0.000 deg W: 0.000
deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 mm 4: 0.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 0 pls

> local 1, 100,100,0,0

> where 1
WORLD: X: 250.000 mm Y:-100.000 mm Z: 0.000 mm U: 0.000 deg V: 0.000 deg W: 0.000
deg
JOINT: 1: 0.000 deg 2: 0.000 deg 3: 0.000 mm 4: 0.000 deg
PULSE: 1: 0 pls 2: 0 pls 3: 0 pls 4: 0 pls
```

3.26.8 WindowsStatus関数

Windowsの起動状態を返します。

書式

WindowsStatus

戻り値

現在のWindows起動状態を整数値で返します。Windows起動状態はビットイメージで返され、以下の状態を示します。

機能名	システム予約	RC+機能可能	PC機能可能
ビット番号	15 - 2	1	0
利用可能機能の詳細		Vision Guide(フレームグラバータイプ) RC+ API フィールドバスマスター	PCファイル PC RS-232C データベースアクセス DLL呼び出し

注意

- 使用可能なコントローラーについて

T/VTシリーズでは使用できません。

解説

本関数は、コントローラー設定を"独立モード"に設定した場合に、コントローラーの起動状態を確認するために使用します。コントローラー設定が"連携モード"に設定されている場合は、RC+機能, PC機能の両方が使用可能になるまでプログラム実行を開始できません。

WindowsStatus関数使用例

```
Print "The current PC Booting up Status is: ", WindowsStatus
```

3.26.9 WOpen

ファイルを書き込みモードで開きます。

書式

WOpen ファイル名 As #ファイル番号

.

.

Close #ファイル番号

パラメーター

ファイル名

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

ファイル番号

30～63の整数値、または式で指定します。

解説

指定したファイルを指定したファイル番号で開きます。このステートメントは、指定したファイルを開いてデータを書き込むときに使います。(すでにあるファイルにデータを書き加えるときは、AOpenについての説明を参照してください。)

指定したファイルがないときは、新規でファイルを作成して、そこに書き込みます。指定したファイルがあれば、既存のデータをすべて削除して、新たに書き込みを行います。

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

Closeステートメントでファイルを閉じ、ファイル番号を解放します。

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

注意

- ネットワークパスが使用できます。
- ファイルへの書き込みはバッファリングされます。

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

参照

[AOpen](#), [BOpen](#), [Close](#), [Print #](#), [ROpen](#), [UOpen](#), [FreeFile関数](#)

WOpen使用例

```
Integer fileNum, i, j

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

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

3.26.10 WorkQue_Add

指定したワークキューにワークキューデータ (ポイントデータとユーザーデータ)を追加します。

書式

WorkQue_Add ワークキュー番号, ポイントデータ [, ユーザーデータ]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1~16)で指定します。

ポイントデータ

ワークキューに追加するポイントデータを指定します。

ユーザーデータ

ポイントデータと一緒に登録するユーザーデータを実数値で指定します。省略可能です。省略すると、0 (実数)がユーザーデータとして登録されます。

解説

ポイントデータとユーザーデータはワークキューの最後尾に追加されます。

ただし、WorkQue_SortによりSort方法が設定されている場合は、その設定されたSort方法にしたがって登録されます。

WorkQue_Rejectにより2重登録防止距離が設定されている場合は、既に登録済みのポイントデータとの距離が計算され、その距離以下のポイントデータであれば、ポイントデータとユーザーデータはワークキューに追加されません。この場合、エラーにはなりません。

ワークキューデータの上限値は "1000"です。ワークキューデータを使い終わったときにWorkQue_Removeにより、ワークキューデータを削除します。

参照

[WorkQue_AutoRemove](#), [WorkQue_Len関数](#), [WorkQue_Reject](#), [WorkQue_Remove](#), [WorkQue_Sort](#)

WorkQueAdd使用例

```
Integer x, y
Real u

P0 = XY(300, 300, 300, 90, 0, 180)
P1 = XY(200, 280, 150, 90, 0, 180)
P2 = XY(200, 330, 150, 90, 0, 180)
P3 = XY(-200, 280, 150, 90, 0, 180)

Pallet 1, P1, P2, P3, 10, 10
x = 1
y = 1
u = 5.3
WorkQue_Add 1, Pallet(1, x, y), u
```


3.26.11 WorkQue_AutoRemove

指定したワークキューに自動削除機能を設定します。

書式

WorkQue_AutoRemove ワークキュー番号,{True | False}

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16)で指定します。

True | False

True: 自動削除機能を有効にします。False: 自動削除機能を無効にします。

解説

ワークキューに自動削除機能を設定します。自動削除機能を有効にした場合、WorkQue_Getによりワークキューからポイントデータを取得したときに、ポイントデータとユーザーデータがワークキューから自動削除されます。

自動削除機能を無効にした場合は、ポイントデータとユーザーデータは削除されません。削除するには、WorkQue_Removeを使います。

WorkQue_UserDataにより、ユーザーデータを取得した場合は、自動削除されません。

ワークキュー毎に、自動削除機能の有効/無効が設定可能です。

参照

[WorkQue_AutoRemove関数](#), [WorkQue_Get関数](#)

WorkQue_AutoRemove使用例

```
WorkQue_AutoRemove 1, True
```

3.26.12 WorkQue_AutoRemove関数

ワークキューに設定されている自動削除機能の状態を返します。

書式

WorkQue_AutoRemove (ワークキュー番号)

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

戻り値

指定ワークキューの自動削除機能が有効の場合は"True"、無効の場合は"False"を返します。

参照

[WorkQue_AutoRemove](#), [WorkQue_Get関数](#)

WorkQue_AutoRemove関数使用例

```
Boolean autoremove  
  
autoremove = WorkQue_AutoRemove (1)
```

3.26.13 WorkQueue_Get関数

指定したワークキューからポイントデータを返します。

書式

WorkQueue_Get(ワークキュー番号 [, インデックス])

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

インデックス

取得するキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。)省略可能です。

戻り値

指定したワークキューからポイントデータを返します。

解説

WorkQueue_Getは、ワークキューからポイントデータを取得します。インデックスを省略した場合、キューデータの先頭データを返します。インデックスを設定した場合は、設定したインデックスのポイントデータを返します。

WorkQueue_AutoRemoveにより、キューデータの自動削除機能を有効にしている場合は、WorkQueue_Getにより、ポイントデータとユーザーデータを削除します。

無効にしている場合は、ポイントデータとユーザーデータは削除しません。削除するにはWorkQueue_Removeを使います。

参照

[WorkQueue_AutoRemove](#), [WorkQueue_Len関数](#), [WorkQueue_Reject](#), [WorkQueue_Remove](#), [WorkQueue_Sort](#)

WorkQueue_Get関数使用例

```
・ キューの先頭のワークへJumpし、追従する
Jump WorkQueue_Get(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
WorkQueueRemove 1
```

3.26.14 WorkQue_Len関数

指定したワークキューに登録された有効なワークキューデータの数 returns します。

書式

WorkQue_Len(ワークキュー番号)

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

戻り値

有効なワークキューデータの登録数を整数値で returns します。

解説

有効なワークキューデータ数の登録数を returns します。

Wait命令の引数としても使うことが可能です。

参照

[WorkQue_Add](#), [WorkQue_Get関数](#), [WorkQue_Remove](#)

WorkQue_Len関数使用例

```
Do
  Do While WorkQue_Len(1) > 0
    WorkQue_Remove 1, 0
  Loop
  If WorkQue_Len(1) > 0 Then
    Jump WorkQue_Get(1, 0) C0
    On gripper
    Wait .1
    WorkQue_Remove 1, 0
    Jump place
    Off gripper
    Jump idlePos
  EndIf
Loop
```

3.26.15 WorkQue_List

指定したワークキューのワークキューデータ一覧 (ポイントデータとユーザーデータ)を表示します。

書式

WorkQue_List ワークキュー番号 [, 表示数]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1~16)で指定します。

表示数

表示数データ数を整数値で指定します。省略可能です。省略すると、すべてのキューデータを表示します。

注意

このコマンドはコマンドウィンドウのみ実行可能です。

参照

[WorkQue_Add](#), [WorkQue_Get関数](#), [WorkQue_Remove](#)

WorkQue_List使用例

コマンドウィンドウでの操作例

```
> WorkQue_List 1
Queue 0    = XY(      1.000,      1.000,      0.000,      0.000 )  /R /0 (      0.000)
Queue 1    = XY(      3.000,      1.000,      0.000,      0.000 )  /R /0 (      2.000)
Queue 2    = XY(      4.000,      1.000,      0.000,      0.000 )  /R /0 (      3.000)
Queue 3    = XY(      5.000,      1.000,      0.000,      0.000 )  /R /0 (      4.000)
Queue 4    = XY(      6.000,      1.000,      0.000,      0.000 )  /R /0 (      5.000)
```

3.26.16 WorkQue_Reject

指定したワークキューにポイントデータの2重登録防止のための最小距離を設定、表示します。

書式

WorkQue_Reject ワークキュー番号 [, 2重登録防止距離]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

2重登録防止距離

2重登録防止距離の最小値を実数値 (単位: mm) で指定します。マイナス値を指定した場合は、0 mm が設定されます。コマンドウィンドウから実行する場合、省略可能です。省略すると、現在の2重登録防止距離が表示されます。

解説

ポイントデータの2重登録防止のための最小距離を設定します。WorkQue_Addにより、最小距離以下のポイントデータを登録しても、ワークキューには登録されません。WorkQue_Rejectは2重登録防止のシステムフィルターとなります。デフォルトは0 mmです。

WorkQue_Addにより、ワークキューデータ (ポイントデータとユーザーデータ) を追加する前に、WorkQue_Rejectを実行する必要があります。

ワークキュー毎に、2重登録防止距離が設定可能です。

参照

[WorkQue_Add](#), [WorkQue_Reject関数](#)

WorkQue_Reject使用例

```
WorkQue_Reject 1, 2.5
```

3.26.17 WorkQue_Reject関数

指定したワークキューに設定されている2重登録防止距離を返します。

書式

WorkQue_Reject (ワークキュー番号)

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

戻り値

実数値 (単位: mm) を返します。

参照

[WorkQue_Add](#), [WorkQue_Reject](#)

WorkQue_Reject関数使用例

```
Real rejectDist  
  
RejectDist = WorkQue_Reject(1)
```

3.26.18 WorkQue_Remove

指定したワークキューからワークキューデータ (ポイントデータとユーザーデータ) を削除します。

書式

WorkQue_Remove ワークキュー番号 [, インデックス | All]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

インデックス

削除するワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。)省略可能です。ワークキューからすべてのワークキューデータを削除する場合はAllで指定します。

解説

ワークキューデータから1つ以上のワークキューデータ (ポイントデータとユーザーデータ) を削除します。ワークキューデータを使い終わったときに、キューデータを削除します。

参照

[WorkQue_Add](#)

WorkQue_Remove使用例

```
Jump WorkQue_Get(1)
On gripper
Wait .1
Jump place
Off gripper
Wait .1
```

```
・ ワークキューからデータを削除
WorkQue_Remove 1
```


3.26.19 WorkQue_Sort

指定したワークキューにSort方法を設定、表示します。

書式

WorkQue_Sort ワークキュー番号 [, Sort方法]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16)で指定します。

Sort方法

Sort方法を整数値 (0～6)、もしくは以下に示す定数で指定します。コマンドウインドウから実行する場合、省略可能です。省略すると、現在のSort方法が表示されます。

定数	値	内容
QUE_SORT_NONE	0	並べ替えなし (ワークキューへの登録順)
QUE_SORT_POS_X	1	X座標昇順
QUE_SORT_INV_X	2	X座標降順
QUE_SORT_POS_Y	3	Y座標昇順
QUE_SORT_INV_Y	4	Y座標降順
QUE_SORT_POS_USER	5	ユーザーデータ (実数)昇順
QUE_SORT_INV_USER	6	ユーザーデータ (実数)降順

解説

ワークキューにSort方法を設定します。WorkQue_Addにより、ポイントデータとユーザーデータを追加した際、設定されたSort方法にしたがってワークキューに登録されます。

WorkQue_UserDataにより、ユーザーデータの再設定をした際、設定されたSort方法にしたがってワークキューの並び順が変わります。

WorkQue_Addにより、ワークキューデータ (ポイントデータとユーザーデータ)を追加する前に、WorkQue_Sortを実行する必要があります。

WorkQue_UserDataにより、ユーザーデータの再設定を行う前に、WorkQue_Sortを実行する必要があります。

ワークキュー毎に、Sort方法が設定可能です。

参照

[WorkQue_Add](#), [WorkQue_UserData](#)

WorkQue_Sort使用例

```
WorkQue_Sort 1, QUE_SORT_POS_X
```

3.26.20 WorkQue_Sort関数

指定したワークキューに設定されているSort方法を返します。

書式

WorkQue_Sort (ワークキュー番号)

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

戻り値

ワークキューに設定されているSort方法を整数値で返します。

- 0 = 並び替えなし (ワークキューへの登録順)
- 1 = X座標昇順
- 2 = X座標降順
- 3 = Y座標昇順
- 4 = Y座標降順
- 5 = ユーザーデータ (実数) 昇順
- 6 = ユーザーデータ (実数) 降順

参照

[WorkQue_Add](#), [WorkQue_Sort](#), [WorkQue_UserData](#)

WorkQue_Sort関数使用例

```
Integer quesort  
  
quesort = WorkQue_Sort(1)
```

3.26.21 WorkQue_UserData

指定したワークキューに登録されているユーザーデータ (実数) の再設定, 表示をします。

書式

WorkQue_UserData ワークキュー番号 [, インデックス] [, ユーザーデータ]

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

インデックス

ワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。) コマンドウインドウから実行する場合、省略可能です。

ユーザーデータ

再設定するユーザーデータを実数値で指定します。コマンドウインドウから実行する場合、省略可能です。省略すると、現在のユーザーデータ (実数) が表示されます。

解説

ワークキューに登録されているユーザーデータの再設定, 表示をします。

WorkQue_Sortにより、以下のSort方法が設定されている場合は、その設定されたSort方法にしたがってワークキューデータの並び順が変わります。

- QUE_SORT_POS_USER: ユーザーデータ (実数) 昇順
- QUE_SORT_INV_USER: ユーザーデータ (実数) 降順

参照

[WorkQue_UserData関数](#)

WorkQue_UserData使用例

```
WorkQue_UserData 1, 1, angle
```

3.26.22 WorkQue_UserData関数

指定したワークキューに登録されているユーザーデータ (実数) を返します。

書式

WorkQue_UserData (ワークキュー番号 [, インデックス])

パラメーター

ワークキュー番号

ワークキューの番号を整数値 (1～16) で指定します。

インデックス

ワークキューデータのインデックスを整数値で指定します。(先頭インデックス番号は0です。)省略可能です。

戻り値

実数値を返します。

参照

[WorkQue_UserData](#)

WorkQue_UserData関数使用例

```
' キューから削除
angle = WorkQue_UserData(1)  'デフォルトのインデックス"0"になる
Jump WorkQue_Get(1) :U(angle)
WorkQue_Remove 1
```

3.26.23 Wrist

ポイントの手首姿勢を設定します。

書式

(1) Wrist ポイント指定 [, Flip | NoFlip]

(2) Wrist

パラメーター

ポイント指定

P番号, P(式), ポイントラベルのいずれかを指定します。

Flip | NoFlip

手首姿勢を指定します。

結果

2つのパラメーターを両方省略すると、ロボットの現在位置の手首姿勢を表示します。

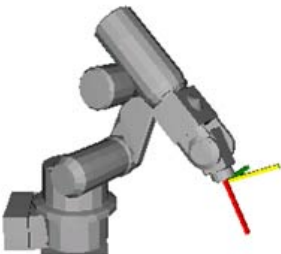
Flip | NoFlipを省略すると、指定ポイントの手首姿勢を表示します。

参照

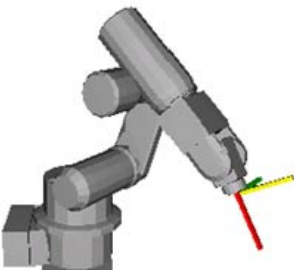
[Elbow](#), [Hand](#), [J4Flag](#), [J6Flag](#), [Wrist関数](#)

Wrist使用例

```
Wrist P0, Flip  
Wrist P(my point), NoFlip  
  
P1 = 320.000, 400.000, 350.000, 140.000, 0.000, 150.000
```



```
Wrist P1, NoFlip  
Go P1
```



```
Wrist P1, Flip  
Go P1
```

3.26.24 Wrist関数

ポイントの手首姿勢を返します。

書式

Wrist [(ポイント指定)]

パラメーター

ポイント指定

P番号, P(式), ポイントラベル, ポイント式のいずれかで指定します。省略可能で、ポイント指定を省略すると、現在のロボット位置の手首姿勢を返します。

戻り値

- 1: NoFlip (/NF)
- 2: Flip (/F)

参照

[Elbow](#), [Hand](#), [J4Flag](#), [J6Flag](#), [Wrist](#)

Wrist関数使用例

```
Print Wrist(pick)
Print Wrist(P1)
Print Wrist
Print Wrist(P1 + P2)
```

3.26.25 Write

ファイル、または通信ポートに文字列を書き込みます。行末ターミネーターは付加しません。

書式

Write # ポート番号, 文字列

パラメーター

ポート番号

ファイルまたは通信ポートを示すID番号です。ファイル番号は、ROpen, WOpen, AOpen などのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C) または OpenNet (TCP/IP) ステートメントで指定した番号です。

文字列

書き込む文字列を指定します。

解説

Write コマンドは Print コマンドとは異なり、行末ターミネーターは付加しません。

注意

- 使用可能なコントローラーについて

T/VT シリーズでは、コントローラーの RS-232C ポートを指定した場合、動作時エラーとなります。

- ファイルへの書き込みはバッファリングされます

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

参照

[Print](#), [Read](#), [WriteBin](#)

Write 使用例

```
OpenCom #1
For i = 1 to 10
    Write #1, data$(i)
Next i
CloseCom #1
```

3.26.26 WriteBin

ファイル、または通信ポートへバイナリーデータを書き出します。

書式

WriteBin #ポート番号, 書き込みデータ

WriteBin #ポート番号, 配列変数名(), バイト数

パラメーター

ポート番号

ファイルまたは通信ポートを示すID番号です。ファイル番号は、BOpenなどのステートメントで指定した番号です。通信ポート番号は、OpenCom (RS-232C)またはOpenNet (TCP/IP)ステートメントで指定した番号です。

書き込みデータ

書き込むデータを整数または式で指定します。

配列変数名()

書き出すデータバイトを格納している、Byte型変数, 整数変数, またはLong型変数の名前を指定します。指定可能なのは1次元の配列変数です。

バイト数

書き出すバイト数を指定します。配列の最大要素番号以下で、かつ256Byte以下である必要があります。通信ポート (TCP/IP)が対象の場合は、配列の最大要素番号以下で、かつ1024Byte以下である必要があります。

注意

- 使用可能なコントローラーについて

T/VTシリーズでは、コントローラーのRS-232Cポートを指定した場合、動作時エラーとなります。

参照

[ReadBin](#), [Write](#)

WriteBin使用例

```
Integer i, data(100)

OpenCom #1
For i = 0 To 100
    WriteBin #1, i
Next i
WriteBin #1, data(), 100
CloseCom #1
```


3.27 X

3.27.1 Xor 演算子

2つの値をビット単位でXor演算 (排他的OR演算)します。

書式

result = 値1 Xor 値2

パラメーター

値1, 値2

数値、または変数名を指定します。

result

整数を返します。

結果

ビット単位でXor演算したresultを返します。

解説

Xor オペレーターはビット単位でXor演算を行います。resultのビットは、2つの値をビット単位でXorした結果です。

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

参照

[And 演算子](#), [LShift関数](#), [Not 演算子](#), [Or 演算子](#), [RShift関数](#)

Xor演算子使用例

```
>print 2 Xor 6
4
>
```

3.27.2 Xqt

ファンクション名で指定したプログラムを実行し、タスクを生成します。

書式

Xqt [タスク番号,] ファンクション名 [(引数リスト)] [, Normal | NoPause | NoEmgAbort]

パラメーター

タスク番号

実行するタスクのタスク番号を、1～32の整数で指定します。省略可能です。バックグラウンドタスクの場合、65～80の整数を指定します。

ファンクション名

実行するファンクション名を指定します。

引数リスト

呼び出されたときにファンクションに渡される引数のリストを指定します。引数が複数ある場合は、カンマで区切ってください。省略可能です。

タスクタイプ

省略可能です。通常は省略します。バックグラウンドタスクの場合、タスクタイプの指定は意味を持ちません。

Normal

通常のタスクを生成します。

NoPause

Pauseステートメントや、Pause入力信号が発生したときに、および、安全扉開状態で一時停止しないタスクを生成する場合に指定します。

NoEmgAbort

非常停止時、およびエラー発生時に処理を継続するタスクを生成する場合に指定します。

解説

Xqt は、指定されたファンクションを開始し、すぐに戻ります。

通常は、タスク番号パラメーターは必要ありません。タスク番号を省略すると、SPEL+が自動的にファンクションにタスク番号をつけるので、ユーザーがタスク番号の管理をする必要はありません。

注意

■ タスクタイプ

タスクタイプで、NoPauseやNoEmgAbortを指定することにより、コントローラー全体を監視するようなタスクを生成することができます。

ただし、これら特殊なタスクは、SPEL+のタスクの動作や特殊なタスクの制限事項を十分理解した上で使用することを強く推奨します。

特殊なタスクの詳細については、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - 特殊なタスク"

■ バックグラウンドタスク

バックグラウンドタスクでXqtを実行した場合、生成されるタスクもバックグラウンドタスクとなります。

バックグラウンドタスクからメインファンクションを実行するには、StartMain命令を使用してください。

バックグラウンドタスクの詳細は、以下のマニュアルを参照してください。

"Epson RC+ ユーザーズガイド - 特殊なタスク"

NoEmgAbort タスクとバックグラウンドタスクで実行できないコマンド

NoEmgAbortタスクとバックグラウンドタスクでは以下のコマンドを実行することはできません。

A	Accel
	AccelR
	AccelS
	AIO_TrackingStart
	AIO_TrackingEnd
	Arc
	Arc3
	Arch
	Arm
	ArmCalib
	ArmCalibCLR
	ArmCalibSET
	ArmClr
	ArmSet
	AutoLJM
	AutoOrientationFlag
	AutoSTM
	AvoidSingularity
B	Base
	BGo
	BMove
	Box
	BoxClr
	Brake
C	Calib
	Cnv_AbortTrack

	Cnv_Accel
	Cnv_AccelLim
	Cnv_Adjust
	Cnv_AdjustClear
	Cnv_AdjustGet
	Cnv_AdjustSet
	Cnv_DownStream
	Cnv_Fine
	Cnv_Mode
	Cnv_OffsetAngle
	Cnv_QueueAdd
	Cnv_QueueMove
	Cnv_QueueReject
	Cnv_QueueRemove
	Cnv_QueueUserData
	Cnv_Trigger
	Cnv_UpStream
	CollisionDetect
	CP
	CP_Offset
	Curve
	CVMove
E	ECP
	ECPClr
	ECPSet
F	Find
	Fine
	FineDist
	Force_Calibrate

	Force_ClearTrigger
	Force_Sensor
	Force_SetTrigger
G	Go
H	Hand_On
	Hand_Off
	Home
	HomeClr
	HomeSet
	Hordr
I	Inertia
J	JTran
	Jump
	Jump3
	Jump3CP
	JRange
L	LatchEnable
	LimitTorque
	LimZ
	LimZMargin
	Local
	LocalClr
M	MCal
	MCordr
	Motor
	Move
O	OLAccel
P	Pass
	PerformMode

	Pg_LSpeed
	Pg_Scan
	Plane
	PlaneClr
	Power
	PTPBoost
	Pulse
Q	QP
	QPDcelR
	QPDcelS
R	Range
	Reset *1
	Restart *2
S	Sense
	SetLatch
	SFree
	SingularityAngle
	SingularityDist
	SingularitySpeed
	SLock
	SoftCP
	Speed
	SpeedFactor
	SpeedR
	SpeedS
	SyncRobots
T	TC
	TGo
	Till

	TLSet
	TLClr
	TMove
	Tool
	Trap
V	VCal
	VcalPoints
	VClS
	VCreateCalibration
	VCreateObject
	VCreateSequence
	VDefArm
	VDefGetMotionRange
	VDefLocal
	VDefSetMotionRange
	VDefTool
	VDeleteCalibration
	VDeleteObject
	VDeleteSeuence
	VEditWindow
	VGet
	VGoCenter
	VLoad
	VLoadModel
	VRun
	VSave
	VSaveImage
	VSaveModel
	VSet

	VShowModel
	VStatsShow
	VStatsReset
	VStatsResetAll
	VStatsSave
	VSD
	VTech
	VTrain
W	WaitPos
	Weight
	WorkQue_Add
	WorkQue_Reject
	WorkQue_Remove
	WorkQue_Sort
	WorkQue_UserData
X	Xqt *3
	XYLim

- *1 Reset Errorは実行可能
- *2 Trap Error の処理タスクからは実行可能
- *3 バックグラウンドタスクからは実行可能

ループ文の中でXQT命令を頻繁に繰り返すような使い方はしないでください Do…Loop等ループ文の中でXQT命令を頻繁に繰り返すような使い方はしないでください。コントローラーがハングアップ状態になる可能性があります。このような使い方をする場合は、Wait命令(Wait 0.1)を入れてください。

参照

[Function...Fend, Halt, Resume, Quit, StartMain, Trap](#) (ユーザー定義トリガー)

Xqt使用例

```
Function main
  Xqt flash           'タスクflashを開始
  Xqt Cycle(5)        'タスクCycleを開始

  Do
    Wait 3            'タスクflashを3秒間実行
    Halt flash        'タスクを一時停止

    Wait 3
```



```
        Resume flash    'タスクを再開
    Loop
Fend

Function Cycle(count As Integer)
    Integer i

    For i = 1 To count
        Jump pick
        On vac
        Wait .2
        Jump place
        Off vac
        Wait .2
    Next i
Fend

Function flash
    Do
        On 1
        Wait 0.2
        Off 1
        Wait 0.2
    Loop
Fend
```

3.27.3 XY関数

指定された座標値をポイントデータとして返します。

書式

XY(x座標要素, y座標要素, z座標要素, u座標要素 [, v座標要素, w座標要素])

パラメーター

x座標要素

x座標を実数値で指定します。

y座標要素

y座標を実数値で指定します。

z座標要素

z座標を実数値で指定します。

u座標要素

u座標を実数値で指定します。

v座標要素

v座標を実数値で指定します。垂直6軸型ロボット (Nシリーズを含む)用パラメーターで、省略可能です。

w座標要素

w座標を実数値で指定します。垂直6軸型ロボット (Nシリーズを含む)用パラメーターで、省略可能です。

戻り値

指定された座標値をポイントデータとして返します。

解説

付加軸(S軸, T軸)を使用していない場合は特別な注意はありません。

Go XY (60,30,-50,45) というような使い方でロボットを指定の座標値へ動作させることができます。

付加軸 (S軸, T軸)を使用している場合は注意が必要です。

XY関数は付加軸を除いたロボットのポイントデータしか返しません。

Go XY (60,30,-50,45) というような使い方をした場合、ロボットは指定の座標値へ動作しますが、付加軸は動きません。付加軸も同時に動作させるようなケースでは Go XY (60,30,-50,45) : ST (10,20) のような指定をしてください。

詳細は、以下のマニュアルを参照してください。"Epson RC+ ユーザーズガイド - 付加軸"

参照

[JA関数, P# \(2. ポイント指定\), ST関数](#)

XY関数使用例

```
P10 = XY(60, 30, -50, 45) + P20
```

3.27.4 XYLim

許容動作エリアの設定と表示をします。

書式

XYLim X軸下限位置, X軸上限位置, Y軸下限位置, Y軸上限位置 [, Z軸下限位置] [, Z軸上限位置]

XYLim

パラメーター

X軸下限位置

マニピュレーターが動作できる下限位置のX座標値 (実数) を、数値または式で指定します。

X軸上限位置

マニピュレーターが動作できる上限位置のX座標値 (実数) を、数値または式で指定します。

Y軸下限位置

マニピュレーターが動作できる下限位置のY座標値 (実数) を、数値または式で指定します。

Y軸上限位置

マニピュレーターが動作できる上限位置のY座標値 (実数) を、数値または式で指定します。

Z軸下限位置

マニピュレーターが動作できる下限位置のZ座標値 (実数) を、数値または式で指定します。省略可能です。

Z軸上限位置

マニピュレーターが動作できる上限位置のZ座標値 (実数) を、数値または式で指定します。省略可能です。

結果

パラメーターを省略すると、現在のXYLim 値を表示します。

解説

XYLimは、許容動作エリアを設定します。多くのロボットシステムでは、ユーザーが関節の動作許容範囲を設定できるようになっていますが、SPEL+言語では関節の動作範囲だけでなく、動作許容範囲も設定できるようになっています。これにより、ロボットのアプリケーションに応じて稼動範囲を制限できます。

XYLim値で設定した動作範囲は、XYLimModeコマンドで設定された監視方法で適用されます。監視方法の詳細は、XYLimModeステートメントを参照してください。

注意

- 許容動作エリアの設定をオフにする

許容動作エリアの設定が必要でない用途も数多くあります。そのため、この設定は簡単にオフできるようになっています。この設定をオフにするためには、各パラメーター値 (X軸下限位置, X軸上限位置, Y軸下限位置, Y軸上限位置) を "0" に設定してください。例: XYLim 0, 0, 0, 0.

- 許容動作制限値のデフォルト

XYLim のデフォルトは、"0, 0, 0, 0" です。(許容動作エリアの設定は、オフになります。)

ヒント

- XYLimをポイント&クリックで設定する

Epson RC+ では、[ツール]メニュー-[ロボットマネージャー]の[動作許容エリア]パネルからXYLim値を設定できます。

参照[Range](#), [XYLimMode](#)**XYLim使用例**

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

```
> XYlim -200, 300, 0, 500  
  
> XYLim  
-200.000, 300.000, 0.000, 500.000
```

3.27.5 XYLim関数

設定されたXY平面許容動作エリアを返します。

書式

XYLim(参照データ)

パラメーター

参照データ

戻り値として返す許容エリアを整数値で指定します。

- 1: 下限値
- 2: 上限値

戻り値

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

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

参照

[XYLim](#)

XYLim関数使用例

```
P1 = XYLim(1)
P2 = XYLim(2)
```

3.27.6 XYLimClr

設定されたXYLimをクリアします。

書式

XYLimClr

参照

[XYLim](#), [XYLimDef](#)関数

XYLimClr関数使用例

下記はXYLimClr関数を使用したプログラムです。

```
Function ClearXYLim
    If XYLimDef = True Then
        XYLimClr
    EndIf
Fend
```

3.27.7 XYLimDef関数

XYLimを設定したかどうかを返します。

書式

XYLimDef

戻り値

XYLimが設定されている場合は"True"、設定されていなし場合は"False"を返します。

参照

[XYLim](#), [XYLimClr](#)

XYLimDef関数使用例

下記はXYLimDef関数を使用したプログラムです。

```
Function ClearXYLim
    If XYLimDef = True Then
        XYLimClr
    EndIf
Fend
```

3.27.8 XYLimMode

XYLimの監視方法の設定と表示をします。

書式

(1) XYLimMode 監視方法

(2) XYLimMode

パラメーター

監視方法

使用するXYLimの監視方法を表す整数式

定数	値	内容
XYLIM_STANDARD	0	動作コマンドの目標座標に対してXYLimを適用します。(Pulseに対しては影響しません。)
XYLIM_STRICT	1	XYLIM_STANDARDの監視方法に加え、動作軌跡とパルス動作に対してもXYLimを適用します。

結果

パラメーターを省略すると、現在設定されているXYLimの監視方法を表示します。

解説

XYLimModeは、指定されているロボットのXYLimの監視方法を設定します。

XYLIM_STANDARDを指定した場合、XYLimで設定した動作範囲は、動作コマンドの目標座標に対してのみ有効となります。動作の開始地点から目標座標までの動作軌跡に対しては適用されません。したがって、動作中、アームがXYLimで設定した範囲外を通過することもあります。このモードでは、XYLimはパルス動作に対しては適用されません。

XYLIM_STRICTを指定した場合、XYLimで設定した動作範囲は、動作コマンドの目標座標と、動作の開始地点から目標座標までの動作軌跡に対して適用されます。したがって、動作中、アームがXYLimで設定した範囲を超えて動こうとするとエラーになります。このモードでは、XYLimはパルス動作に対しても適用されます。ただし、開始地点がXYLim範囲外で目標座標がXYLim範囲内のようにXYLim範囲外から範囲内に動く場合、XYLim範囲外エラーにならずに動けます。

ロボットと装置との干渉を防ぐために、XYLIM_STRICTでご使用いただくことを推奨します。

Epson RC+ のコントローラー環境設定により、コントローラー起動時の監視方法のデフォルト値を変更することができます。XYLimModeコマンドで設定された値は、コントローラー再起動まで有効です。コントローラーを再起動した場合、XYLimの監視方法はコントローラー環境設定で指定した監視方法にリセットされます。

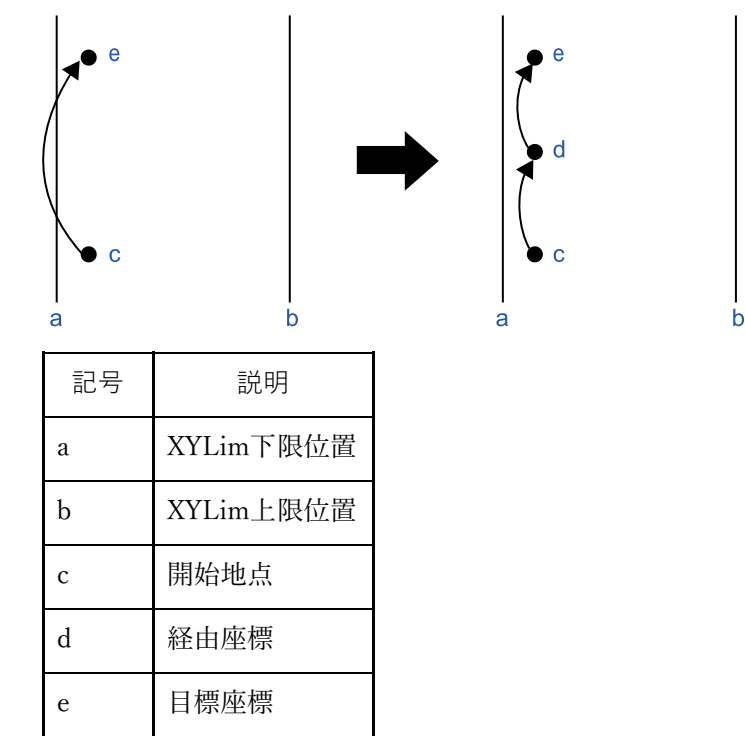
注意

XYLIM_STANDARDは、アームがXYLimで設定した範囲外を通過することがあります。そのため、XYLimを実際の障害物と余裕をもって設定したり、XYLimの境界付近の動作は低速で軌道を確認して、ロボットと装置とが干渉しないことに注意してください。

起こりやすいエラー

- XYLim境界付近でPTP動作を実行した場合

Go命令などのPTP動作を実行すると、開始地点と目標座標は左下図の矢印のように動きます。そのため、XYLim_STRICTでは、目標座標は範囲内ですが、動作軌跡は範囲外なのでエラーになります。このような場合は、右下図のように開始位置と目標座標の間に経由座標を追加し、動作軌跡がXYLimを超えないようにして回避してください。



■ 過去プロジェクト使った場合

XYLimの監視方法をXYLIM_STRICTにしたコントローラーに、コントローラーFWのVer 7.5.2.0、7.5.52.0未満で使っていたプロジェクトを適用すると、動作軌跡が範囲外エラーになる場合があります。その場合、経由座標を追加するなどXYLimを超えないようにプログラムを修正してください。

参照

XYLim

XYLimMode コマンド使用例

下記はXYLimModeを使用する例です。現在位置からP1まではXYLIM_STANDARDで移動し、P1からP2まではXYLIM_STRICTで移動します。

```
Function XYLimMode_sample
Motor On
XYLimMode XYLIM_STANDARD
Go P1      ' 目標座標のみにXYLimを適用
XYLimMode XYLIM_STRICT
Go P2      ' 目標座標と動作軌跡にXYLimを適用
Fend
```

下記は、XYLimModeをコマンドウィンドウで使用する例です。現在のXYLimの監視方法を表示します。

```
> XYLimMode
1
```

3.27.9 XYLimMode関数

設定されたXYLimの監視方法を取得します。

書式

XYLimMode

戻り値

設定されたXYLimの監視方法を返します。

- 0 = 動作コマンドの目標座標に対してXYLimを適用する。(パルス動作に適用しない)
- 1 = 動作コマンドの目標座標と動作軌跡に対してXYLimを適用する。(パルス動作にも適用する)

参照

[XYLimMode](#)

XYLimMode関数使用例

下記はXYLimMode関数の使用例です。変数にXYLimの監視方法を取得し、それを表示するプログラムです。

```
Function XYLimMode_sample
Integer iVar

iVar = XYLimMode
Print iVar

Fend
```

4. Appendix A: SPEL+ コマンド 使用条件一覧

4.1 Appendix A: SPEL+ コマンド 使用条件一覧

- コマンドウィンドウ: コマンドウィンドウで使えます。
- プログラム: SPEL+プログラム中でステートメントとして使えます。
- 関数: 関数として使えます。

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
A	AbortMotion	○	○	○	-
	Abs	-	-	○	○
	Accel	○	○	○	○
	AccelMax	-	-	○	○
	AccelR	○	○	○	○
	AccelS	○	○	○	○
	Acos	-	-	○	○
	Agl	-	-	○	○
	AglToPls	-	-	○	○
	AIO_In	○	○	-	○
	AIO_InW	○	○	-	○
	AIO_Out	○	○	○	○
	AIO_OutW	○	○	○	○
	AIO_Set	○	○	○	○
	AIO_TrackingSet	○	○	○	-
	AIO_TrackingStart	-	-	○	-
	AIO_TrackingEnd	-	-	○	-
	AIO_TrackingOn	○	○	○	○
	Align	-	-	○	○
	AlignECP	-	-	○	○
	And	-	-	○	-
	AOpen	○	○	○	-
	Arc	○	○	○	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Arc3	○	○	○	-
	Arch	○	○	○	○
	AreaCorrection	○	○	○	○
	AreaCorrectionClr	○	○	○	-
	AreaCorrectionDef	○	○	○	○
	AreaCorrectionInv	○	○	○	○
	AreaCorrectionOffset	○	○	○	○
	AreaCorrectionSet	○	○	○	-
	Arm	○	○	○	○
	ArmCalib	○	○	○	○
	ArmCalibClr	○	○	○	-
	ArmCalibDef	-	-	○	○
	ArmCalibSet	○	○	○	○
	ArmClr	○	○	○	-
	ArmDef	-	-	○	○
	ArmLib	-	-	○	-
	ArmReserve	-	-	-	○
	ArmSet	○	○	○	○
	Asc	-	-	○	○
	Asin	-	-	○	○
	Atan	-	-	○	○
	Atan2	-	-	○	○
	ATCLR	○	○	○	-
	AtHome	-	-	○	○
	ATRQ	○	○	○	○
	AutoLJM	○	○	○	○
	AutoSTM	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	AvoidSingularity	○	○	○	○
B	Base	○	○	○	-
	BClr	-	-	○	○
	BClr64	-	-	○	○
	BGo	○	○	○	-
	BMove	○	○	○	-
	Boolean	-	-	○	-
	BOpen	○	○	○	-
	Box	○	○	○	○
	BoxClr	○	○	○	-
	BoxDef	-	-	○	○
	Brake	○	○	○(関数のみ)	○
	BSet	-	-	○	○
	BSet64	-	-	○	○
	BTst	-	-	○	○
	BTst64	-	-	○	○
	Byte	-	-	○	-
C	Calib	○	○	○	-
	Call	-	-	○	-
	CalPls	○	○	○	○
	ChDir	○	○	○	-
	ChDisk	○	○	○	-
	ChDrive	○	○	○	-
	ChkCom	-	-	○	○
	ChkNet	-	-	○	○
	Chr\$	-	-	○	○
	ClearHistory	○	-	-	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	ClearPoints	○	○	○	-
	Close	○	○	○	-
	CloseCom	○	○	○	-
	CloseDB	○	○	○	-
	CloseNet	○	○	○	-
	Cls	○	○	○	-
	Cnv_AbortTrack	○	○	○	-
	Cnv_Accel	○	○	○	○
	Cnv_AccelLim	○	○	○	○
	Cnv_Adjust	○	○	○	-
	Cnv_AdjustClear	○	○	○	-
	Cnv_AdjustGet	○	○	○	○
	Cnv_AdjustSet	○	○	○	-
	Cnv_Downstream	○	○	○	○
	Cnv_Fine	○	○	○	○
	Cnv_LPulse	-	-	○	○
	Cnv_Mode	○	○	○	○
	Cnv_Name\$	-	-	○	○
	Cnv_Number	-	-	○	○
	Cnv_OffsetAngle	○	○	○	○
	Cnv_Point	-	-	○	○
	Cnv_PosErr	-	-	○	○
	Cnv_PosErrOffset	-	-	○	-
	Cnv_Pulse	-	-	○	○
	Cnv_QueueAdd	○	○	○	-
	Cnv_QueueGet	-	-	○	○
	Cnv_QueueLen	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Cnv_QueueList	○	○	-	-
	Cnv_QueueMove	○	○	○	-
	Cnv_QueueReject	○	○	○	○
	Cnv_QueueRemove	○	○	○	-
	Cnv_QueueUserData	○	○	○	○
	Cnv_RobotConveyor	-	-	○	○
	Cnv_Speed	-	-	○	○
	Cnv_Trigger	○	○	○	-
	Cnv_Upstream	○	○	○	○
	CollisionDetect	○	○	○	○
	Cont	○	-	○	-
	Copy	○	○	○	-
	Cos	-	-	○	○
	CP	○	○	○	○
	Ctr	-	-	○	○
	CTReset	○	○	○	-
	CtrlDev	-	-	○	○
	CtrlInfo	-	-	○	○
	CtrlPref	-	-	-	○
	CurDir\$	-	-	○	○
	CurDisk\$	-	-	○	○
	CurDrive\$	-	-	○	○
	CurPos	-	-	○	○
	Curve	○	○	○	-
	CVMove	○	○	○	-
	CP_Offset	○	○	○	○
	CR	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	CS	○	○	○	○
	CT	○	○	○	○
	CU	○	○	○	○
	CV	○	○	○	○
	CW	○	○	○	○
	CX	○	○	○	○
	CY	○	○	○	○
	CZ	○	○	○	○
D	Date	○	○	○	-
	Date\$	-	-	○	○
	Declare	-	-	○	-
	DegToRad	-	-	○	○
	Del	○	○	○	-
	DeleteDB	○	○	○	-
	DiffPoint	○	○	○	○
	DispDev	○	○	○	○
	Dist	-	-	○	○
	Do...Loop	-	-	○	-
	Double	-	-	○	-
E	ECP	○	○	○	○
	ECPClr	○	○	○	-
	ECPDef	-	-	○	○
	ECPSet	○	○	○	○
	ElapsedTime	-	-	○	○
	Elbow	○	○	○	○
	EncTemper	○	○	○	○
	Eof	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Era	-	-	○	○
	EResume	○	○	○	-
	Erf\$	-	-	○	○
	Erl	-	-	○	○
	Err	-	-	○	○
	Errb	○	○	○	○
	ErrMsg\$	-	-	○	○
	Error	○	○	○	-
	ErrorOn	-	-	○	○
	Ert	-	-	○	○
	EStopOn	-	-	○	○
	Eval	-	-	○	○
	Exit	-	-	○	-
	ExportPoints	○	○	○	-
F	FbusIO_GetBusStatus	-	-	○	○
	FbusIO_GetDeviceStatus	-	-	○	○
	FbusIO_SendMsg	○	○	○	-
	FileDateTime\$	-	-	○	○
	FileExists	-	-	○	○
	FileLen	-	-	○	○
	Find	○	○	○	-
	FindPos	-	-	○	○
	Fine	○	○	○	○
	FineDist	○	○	○	○
	FineStatus	○	○	○	○
	Fix	-	-	○	○
	Flush	○	○	○	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	FmtStr	○	○	○	-
	FmtStr\$	-	-	○	○
	FolderExists	-	-	○	○
	For...Next	-	-	○	-
	FreeFile	-	-	○	○
	Function...Fend	-	-	○	-
G	GetIODef	-	-	○	-
	GetProjectInfo	-	-	○	-
	GetRobotInsideBox	-	-	○	○
	GetRobotInsidePlane	-	-	○	○
	Global	-	-	○	-
	GetSetNet	-	-	○	-
	Go	○	○	○	-
	Gosub...Return	-	-	○	-
	Goto	-	-	○	-
	GUIVer\$	-	-	-	○
H	Halt	-	-	○	-
	Hand	○	○	○	○
	HealthCalcPeriod	○	○	○	○
	HealthCtrlAlarmOn	○	○	○	○
	HealthCtrlInfo	○	○	○	○
	HealthCtrlRateOffset	○	○	○	-
	HealthCtrlReset	○	○	○	-
	HealthCtrlWarningEnable	○	○	○	○
	HealthRateCtrlInfo	○	○	○	○
	HealthRateRBInfo	○	○	○	○
	HealthRBArmOn	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	HealthRBAnalysis	○	○	○	○
	HealthRBDistance	○	○	○	○
	HealthRBInfo	○	○	○	○
	HealthRBRateOffset	○	○	○	-
	HealthRBReset	○	○	○	-
	HealthRBSpeed	○	○	○	○
	HealthRBStart	○	○	○	-
	HealthRBStop	○	○	○	-
	HealthRBTRQ	○	○	○	○
	HealthRBWarningEnable	○	○	○	○
	Here	○	○	○	○
	Hex\$	-	-	○	○
	History	○	-	-	-
	Hofs	○	○	○	○
	HofsJointAccuracy	○	○	○	-
	Home	○	○	○	-
	HomeClr	○	○	○	-
	HomeDef	-	-	○	○
	HomeSet	○	○	○	○
	Hordr	○	○	○	○
	Hour	○	○	○	○
I	If...Then..Else...EndIf	-	-	○	-
	ImportPoints	○	○	○	-
	In	-	-	○	○
	InBCD	-	-	○	○
	Inertia	○	○	○	○
	InPos	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Input	○	○	○	-
	Input #	○	○	○	-
	InputBox	○	○	○	-
	InReal	-	-	○	○
	InsideBox	-	-	○	○
	InsidePlane	-	-	○	○
	InStr	-	-	○	○
	Int	-	-	○	○
	Int32	-	-	○	-
	Integer	-	-	○	-
	InW	-	-	○	○
	IODef	-	-	○	○
	IOLabel\$	-	-	○	○
	IONumber	-	-	○	○
J	J1Angle	○	○	○	○
	J4Angle	○	○	○	○
	J1Flag	○	○	○	○
	J2Flag	○	○	○	○
	J4Flag	○	○	○	○
	J6Flag	○	○	○	○
	JA	-	-	○	○
	JogPanel	-	-	○	-
	Joint	○	○	○	-
	JointAccuracy	○	○	○	○
	JRange	○	○	○	○
	JS	-	-	○	○
	JT	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	JTran	○	○	○	-
	Jump	○	○	○	-
	Jump3	○	○	○	-
	Jump3CP	○	○	○	-
	JumpTLZ	○	○	○	-
L	LatchEnable	○	○	○	-
	LatchPos	-	-	○	○
	LatchState	-	-	○	○
	LCase\$	-	-	○	○
	Left\$	-	-	○	○
	Len	-	-	○	○
	LimitTorque	○	○	○	○
	LimitTorqueLP	○	○	○	○
	LimitTorqueStop	○	○	○	○
	LimitTorqueStopLP	○	○	○	○
	LimZ	○	○	○	○
	LimZMargin	○	○	○	○
	Line Input	○	○	○	-
	Line Input #	○	○	○	-
	LJM	-	-	○	○
	LoadPoints	○	○	○	-
	Local	○	○	○	○
	LocalClr	○	○	○	-
	LocalDef	-	-	○	○
	LocalReserve	-	-	-	○
	Lof	-	-	○	○
	LogIn	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Long	-	-	○	-
	LSet\$	-	-	○	○
	LShift	-	-	○	○
	LShift64	-	-	○	○
	LTrim\$	-	-	○	○
M	Mask	-	-	○	-
	MCal	○	○	○	-
	MCalComplete	-	-	○	○
	MCordr	○	○	○	○
	MemIn	-	-	○	○
	MemInW	-	-	○	○
	MemOff	○	○	○	-
	MemOn	○	○	○	-
	MemOut	○	○	○	-
	MemOutW	○	○	○	-
	MemSw	-	-	○	○
	MHour	-	-	○	○
	Mid\$	-	-	○	○
	MkDir	○	○	○	-
	Mod	-	-	○	-
	Motor	○	○	○	○
	Move	○	○	○	-
	MsgBox	○	○	○	○
	MyTask	-	-	○	○
N	Next	-	-	○	-
	Not	-	-	○	-
O	Off	○	○	○	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	OLAccel	○	○	○	○
	OLRate	○	○	○	○
	On	○	○	○	-
	OnErr	-	-	○	-
	OpBCD	○	○	○	-
	OpenCom	○	○	○	○
	OpenDB	○	○	○	-
	OpenNet	○	○	○	○
	Oport	-	-	○	○
	Or	-	-	○	-
	Out	○	○	○	○
	OutReal	○	○	○	○
	OutW	○	○	○	○
P	P#	○	○	○	-
	PAgl	-	-	○	○
	Pallet	○	○	○	○
	PalletClr	○	○	○	-
	ParseStr	○	○	○	○
	Pass	○	○	○	-
	Pause	-	-	○	-
	PauseOn	-	-	○	○
	PDescription	○	○	○	-
	PDescription\$	○	○	-	○
	PDef	-	-	○	○
	PDel	○	○	○	-
	PerformMode	○	○	○	○
	PG_FastStop	○	○	○	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	PG_LSpeed	○	○	○	○
	PG_Scan	○	○	○	-
	PG_SlowStop	○	○	○	-
	PLabel	○	○	○	-
	PLabel\$	-	-	○	○
	Plane	○	○	○	○
	PlaneClr	○	○	○	-
	PlaneDef	-	-	○	○
	PList	○	○	○	-
	PLocal	○	○	○	○
	Pls	-	-	○	○
	PNumber	-	-	○	○
	PosFound	-	-	○	○
	Power	○	○	○	○
	PPls	-	-	○	○
	Print	○	○	○	-
	Print #	○	○	○	-
	PTCLR	○	○	○	-
	PTPBoost	○	○	○	○
	PTPBoostOK	-	-	○	○
	PTPTime	-	-	○	○
	PTran	○	○	○	-
	PTRQ	○	○	○	○
	Pulse	○	○	○	○
Q	QP	○	○	○	-
	QPDcelR	○	○	○	○
	QPDcelS	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Quit	-	-	○	-
R	RadToDeg	-	-	○	○
	Randomize	○	○	○	-
	Range	○	○	○	-
	Read	○	○	○	-
	ReadBin	○	○	○	-
	Real	-	-	○	-
	RealAccel	-	-	○	○
	RealPls	-	-	○	○
	RealPos	-	-	○	○
	RealTorque	-	-	○	○
	Recover	○	-	○	○
	RecoverPos	-	-	○	○
	Redim	○	○	○	-
	Rename	○	○	○	-
	RenDir	○	○	○	-
	Reset	○	○	○	-
	ResetElapsedTime	○	○	○	-
	Restart	○	-	○	-
	Resume	-	-	○	-
	Return	-	-	○	-
	Right\$	-	-	○	○
	RmDir	○	○	○	-
	Rnd	-	-	○	○
	Robot	○	○	○	○
	RobotInfo	-	-	○	○
	RobotInfo\$	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	RobotModel\$	-	-	○	○
	RobotName\$	-	-	○	○
	RobotSerial\$	-	-	○	○
	RobotType	-	-	○	○
	ROpen	○	○	○	-
	ROTok	○	○	○	○
	RSet\$	-	-	○	○
	RShift64	-	-	○	○
	RShift	-	-	○	○
	RTrim\$	-	-	○	○
	RunDialog	-	-	○	-
S	SafetyOn	-	-	○	○
	SavePoints	○	○	○	-
	Seek	○	○	○	-
	Select...Send	-	-	○	-
	SelectDB	○	○	○	○
	Sense	○	○	○	-
	SetCom	○	○	○	-
	SetIn	○	○	○	-
	SetInReal	○	○	○	-
	SetInW	○	○	○	-
	SetIODef	-	-	○	-
	SetLatch	○	○	○	-
	SetNet	○	○	○	-
	SetSw	○	○	○	-
	SF_GetParam	○	○	○	○
	SF_GetParam\$	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	SF_GetStatus	○	○	○	○
	SF_LimitSpeedS	○	○	○	○
	SF_LimitSpeedSEnable	○	○	○	○
	SF_RealSpeedS	○	○	○	○
	SF_PeakSpeedS	○	○	○	○
	SF_PeakSpeedSClear	○	○	○	-
	SF_SetIn	○	-	○	-
	SF_SetSw	○	-	○	-
	SFree	○	○	○	○
	Sgn	-	-	○	○
	Short	-	-	○	-
	Signal	○	○	○	-
	SignalReserve	-	-	-	○
	SimGet	-	-	○	-
	SimSet	○	○	○	-
	Sin	-	-	○	○
	SingularityAngle	○	○	○	○
	SingularityDist	○	○	○	○
	SingularitySpeed	○	○	○	○
	SLock	○	○	○	-
	SoftCP	○	○	○	○
	Space\$	-	-	○	○
	Speed	○	○	○	○
	SpeedFactor	○	○	○	○
	SpeedR	○	○	○	○
	SpeedRLimitation	○	○	○	○
	SpeedS	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	Sqr	-	-	○	○
	ST	-	-	○	○
	STM	○	○	○	○
	StartMain	-	-	○	-
	Stat	-	-	○	○
	Str\$	-	-	○	○
	String	-	-	○	-
	Sw	-	-	○	○
	SyncLock	-	-	○	-
	SyncLockReserve	-	-	-	○
	SyncUnlock	-	-	○	-
	SyncRobots	○	○	○	○
	SysConfig	○	○	-	-
	SysErr	-	-	○	○
T	Tab\$	-	-	○	○
	Tan	-	-	○	○
	TargetOK	-	-	○	○
	TaskDone	-	-	○	○
	TaskInfo	-	-	○	○
	TaskInfo\$	-	-	○	○
	TaskReserve	-	-	-	○
	TaskState	○	○	○	○
	TaskWait	○	○	○	-
	TC	○	○	○	-
	TCLim	○	○	○	○
	TCPSpeed	-	-	○	○
	TCSpeed	○	○	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	TeachOn	-	-	○	○
	TeachPoint	-	-	-	○
	TGo	○	○	○	-
	Till	○	○	○	-
	TillOn	-	-	○	○
	Time	○	○	○	○
	Time\$	-	-	○	○
	TimerReserve	-	-	-	○
	TLClr	○	○	○	-
	TLDef	-	-	○	○
	TLReserve	-	-	-	○
	TLSet	○	○	○	○
	TMOut	○	○	○	-
	TMove	○	○	○	-
	Tmr	-	-	○	○
	TmReset	○	○	○	-
	Toff	○	○	○	-
	Ton	○	○	○	-
	Tool	○	○	○	○
	ToolLib	-	-	○	-
	ToolWizard	-	-	-	○
	Trap	-	-	○	-
	Trim\$	-	-	○	○
	TW	-	-	○	○
U	UBound	-	-	○	○
	UByte	-	-	○	-
	UCase\$	-	-	○	○

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	UInt32	-	-	○	-
	UOpen	○	○	○	-
	UpdateDB	○	○	○	-
	UserErrorDef	-	-	-	○
	UserErrorLabel\$	-	-	-	○
	UserErrorNumber	-	-	-	○
	UShort	-	-	○	-
V	Val	-	-	○	○
	VSD	○	○	○	○
	VxCalib	○	○	○	-
	VxCalDelete	○	○	○	-
	VxCalLoad	○	○	○	-
	VxCalInfo	-	-	○	○
	VxCalSave	○	○	○	-
	VxTrans	-	-	○	○
W	Wait	○	○	○	-
	WaitNet	○	○	○	-
	WaitPos	○	○	○	-
	WaitSig	○	○	○	-
	Weight	○	○	○	○
	Where	○	○	-	-
	WindowsStatus	-	-	○	○
	WorkQue_Add	○	○	○	-
	WorkQue_AutoRemove	○	○	○	○
	WorkQue_Get	○	○	○	○
	WorkQue_Len	○	○	○	○
	WorkQue_List	○	○	-	-

命令		コマンドウィンドウ		プログラム	関数
		RC+	TP3/TP4		
	WorkQue_Reject	○	○	○	○
	WorkQue_Remove	○	○	○	-
	WorkQue_Sort	○	○	○	○
	WorkQue_UserData	○	○	○	○
	WOpen	○	○	○	-
	Wrist	○	○	○	○
	Write	○	○	○	-
	WriteBin	○	○	○	-
X	Xor	-	-	○	-
	Xqt	-	-	○	-
	XY	-	-	○	○
	XYLim	○	○	○	○
	XYLimClr	○	○	○	-
	XYLimDef	-	-	○	○
	XYLimMode	○	○	○	○

5. Appendix B: 互換性に関する注意

5.1 B-1: EPSON RC+ 6.0 からの互換性に関する注意

5.1.1 概要

Epson RC+ 8.0は、EPSON RC+ 7.0の互換性を保ちます。

すでに、RC620コントローラーで、EPSON RC+ 6.0をご利用の方が、RC800シリーズコントローラーでEPSON RC+ 7.0, Epson RC+ 8.0を使う場合に読んでいただきたい内容をまとめました。

EPSON RC+ 7.0, Epson RC+ 8.0を使用する場合は、ハードウェアの違い、対応マネジュレーターや関節数の違い、ソフトウェアの実行環境の違いなどにより、EPSON RC+ 6.0とは異なっている部分があります。あらかじめその内容をご理解いただき、安全にロボットをご使用してください。

EPSON RC+ 7.0, Epson RC+ 8.0は、できるだけ今までの製品と互換性を保ちながら改善し、先進のソフトウェア技術を取り入れて設計しています。しかしながら、ロボットコントローラーに特化し、より使いやすくするために、これまでのEPSON RC+ 6.0との互換性を全て満たしてはいない部分や、削除された部分があります。

以下の互換性に関する記述は、EPSON RC+ 6.0を元に、EPSON RC+ 7.0, Epson RC+ 8.0と比較をしています。

5.1.2 全般にわたる違い

EPSON RC+ 6.0と、EPSON RC+ 7.0, Epson RC+ 8.0の全般にわたる違いは下記のとおりです。

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 6.0
タスク数	最大32タスク (バックグラウンドタスクは最大16タスク)	最大32タスク (バックグラウンドタスクは最大16タスク)
タスクの種類	NoPauseタスクの指定が可 NoEmgAbortタスクの指定が可 バックグラウンドタスクの指定が可	NoPauseタスクの指定が可 NoEmgAbortタスクの指定が可 バックグラウンドタスクの指定が可
TRAP ERRORなどの特殊なTRAP	サポートされています	サポートされています
TRAP トラップ番号で起動されるタスク	専用のタスク番号	専用のタスク番号
マルチマネジュレーター	サポートされています	サポートされています
ロボット番号	1～16	1～16
Real型の有効桁数	6桁	6桁
Double型の有効桁数	14桁	14桁
配列要素数	文字列変数以外 ローカル変数: 2,000 グローバル変数: 100,000 モジュール変数: 100,000 バックアップ変数: 4,000	文字列変数以外 ローカル変数: 2,000 グローバル変数: 100,000 モジュール変数: 100,000 バックアップ変数: 4,000

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 6.0
	文字列変数 ローカル変数: 200 グローバル変数: 10,000 モジュール変数: 10,000 バックアップ変数: 400	文字列変数 ローカル変数: 200 グローバル変数: 10,000 モジュール変数: 10,000 バックアップ変数: 400
デバイス番号	21:PC 22:リモート 20:TP4	21:PC 22:リモート 28:LCD
コントロールデバイス	リモートI/O PC リモートEthernet TP4	リモートI/O PC
タイマー番号の範囲	0～63	0～63
プログラム容量	8MB	8MB
SyncLock, SyncUnlock用のシグナル番号の範囲	0～63	0～63
WaitSig, Signal用のシグナル番号の範囲	0～63	0～63
メモリーI/O点数	1024	1024
I/Oポートの番号	EPSON RC+ 6.0と共通	
イーサネットのポート番号	201～216	201～216
リモート入出力割りあて	標準で機能割りあて済	デフォルト設定なし
RS-232C通信ポート番号 SPEL+制御部	1～8, 1001～1008	1～8, 1001, 1002
RS-232C通信ポートの OpenCom実行	必須	必須
ファイルへの入出力	サポートされています	サポートされています
ファイルアクセスのファイル 番号	30～63	30～63
データベースアクセス番号	501～508	501～508
VisionGuide	スマートカメラタイプ フレームグラバタイプ	スマートカメラタイプ フレームグラバタイプ
コンベヤートラッキング	サポートされています	サポートされています
PGロボット	サポートされています	サポートされています

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 6.0
OCR	サポートされています	サポートされています
セキュリティー	サポートされています	サポートされています
VB Guide 6.0 (RC+ API)	サポートされています	サポートされています
フィールドバスI/Oの扱い	通常のI/O命令を使用	通常のI/O命令を使用
フィールドバス マスター	応答性が保証されません	応答性が保証されません
フィールドバス スレーブ	応答性が保証されています	応答性が保証されています
GUI Builder	サポートされています	サポートされています
エラー番号	Epson RC+6.0と共通	

5.1.3 命令の互換性一覧

- +: 機能拡張、機能変更されているが上位互換が保たれている命令
- -: 変更がない命令
- !: 機能変更または、文法変更があり注意が必要な命令
- !!: 大幅な変更があり注意が必要な命令
- ×: 削除された命令

命令		互換性	備考
A	Abs関数	-	
	Accel	-	
	Accel関数	-	
	AccelMax関数	-	
	AccelR	-	
	AccelR関数	-	
	AccelS	-	
	AccelS関数	-	
	Acos関数	-	
	AglToPls関数	-	
	Agl関数	-	
	AlignECP関数	-	
	Align関数	-	

命令		互換性	備考
	And	-	
	Arc	-	
	Arc3	-	
	Arch	-	
	Arch関数	-	
	Arm	-	
	ArmClr	-	
	ArmDef関数	-	
	ArmSet	-	
	ArmSet関数	-	
	Arm関数	-	
	Asc関数	-	
	Asin関数	-	
	Atan2関数	-	
	Atan関数	-	
	ATCLR	-	
	ATRQ	-	
	ATRQ関数	-	
B	Base	-	
	Base関数	-	
	BClr関数	-	
	BGo	+	動作モード指定を追加
	BMove	-	
	Boolean	-	
	Box	+	リモート出力論理設定の指定を追加
	Box関数	-	
	BoxClr関数	-	
	BoxDef関数	-	

命令		互換性	備考
	Brake	-	
	Brake関数	-	
	BSet関数	-	
	BTst関数	-	
	Byte	-	
C	Call	-	
	ChkCom関数	-	
	ChkNet関数	-	
	Chr\$関数	-	
	ClearPoints	-	
	CloseCom	-	
	CloseNet	-	
	Cls	-	
	Cos関数	-	
	CP	-	
	CP関数	-	
	CTReset	-	
	CtrlDev関数	-	
	CtrlInfo関数	-	
	Ctr関数	-	
	CurPos関数	-	
	Curve	-	
	CVMove	-	
	CX～CW	-	
	CX～CW関数	-	
D	Date	-	
	Date\$関数	-	
	DegToRad関数	-	

命令		互換性	備考
	DispDev	-	
	DispDev関数	-	
	Dist関数	-	
	Do...Loop	-	
	Double	-	
E	ECP	-	
	ECPClr	-	
	EcpDef関数	-	
	ECPSet	-	
	ECPSet関数	-	
	ECP関数	-	
	Elbow	-	
	Elbow関数	-	
	Era関数	-	
	EResume	-	
	Erf\$関数	-	
	Erl関数	-	
	ErrMsg\$関数	-	
	Error	-	
	ErrorOn関数	-	
	Err関数	-	
	Ert関数	-	
	EStopOn関数	-	
	Exit	-	
	Find	-	
	FindPos関数	-	
	Fine	-	
	Fine関数	-	

命令		互換性	備考
	Fix関数	-	
	FmtStr\$	-	
	For...Next	-	
	Function...Fend	-	
G	Global	-	
	Go	+	動作モード指定を追加
	Gosub...Return	-	
	Goto	-	
H	Halt	-	
	Hand	-	
	Hand関数	-	
	Here	-	
	Here関数	-	
	Hex\$関数	-	
	Home	-	
	HomeClr	-	
	HomeDef関数	-	
	HomeSet	-	
	HomeSet関数	-	
	HOrdr	-	
	HOrdr関数	-	
	Hour	-	
	Hour関数	-	
I	If...EndIf	-	
	In関数	-	
	InBCD関数	-	
	Inertia	-	
	Inertia関数	-	

命令		互換性	備考
	InPos関数	-	
	Input	-	
	Input#	-	
	InsideBox関数	-	
	InsidePlane関数	-	
	InStr関数	-	
	Integer	-	
	Int関数	-	
	InW関数	-	
	IOLabel\$関数	-	
	IONumber関数	-	
J	J1Flag	-	
	J1Flag関数	-	
	J2Flag	-	
	J2Flag関数	-	
	J4Flag	-	
	J4Flag関数	-	
	J6Flag	-	
	J6Flag関数	-	
	JA関数	-	
	Joint	-	
	JRange	-	
	JRange関数	-	
	JS関数	-	
	JTran	-	
	JT関数	-	
	Jump	+	動作モード指定を追加
	Jump3	-	

命令		互換性	備考
	Jump3CP	-	
L	LCase\$関数	-	
	Left\$関数	-	
	Len関数	-	
	LimZ	-	
	LimZ関数	-	
	Line Input	-	
	Line Input#	-	
	LJM関数	-	
	LoadPoints	-	
	Local	-	
	LocalClr	-	
	LocalDef関数	-	
	Local関数	-	
	Lof関数	-	
	Long	-	
	LSet\$関数	-	
	LShift関数	-	
	LTrim\$関数	-	
M	Mask	-	
	MemInW関数	-	
	MemIn関数	-	
	MemOff	-	
	MemOn	-	
	MemOut	-	
	MemOutW	-	
	MemSw関数	-	
	Mid\$関数	-	

命令		互換性	備考
	Mod	-	
	Motor	-	
	Motor関数	-	
	Move	-	
	MyTask関数	-	
N	Not	-	
O	Off	-	
	OLAccel	-	
	OLAccel関数	-	
	OLRate	-	
	OLRate関数	-	
	On	-	
	OnErr	-	
	OpBCD	-	
	OpenCom	-	
	OpenNet	-	
	Oport関数	-	
	Or	-	
	Out	-	
	OutW	-	
	OutW関数	-	
	Out関数	-	
P	P#	-	
	PAgl関数	-	
	Pallet	+	座標値指定の追加
	Pallet関数	-	
	ParseStr	-	
	ParseStr関数	-	

命令		互換性	備考
	Pass	-	
	Pause	-	
	PauseOn関数	-	
	PDef関数	-	
	PDel	-	
	PLabel	-	
	PLabel\$関数	-	
	Plane	-	
	Plane関数	-	
	PlaneClr	-	
	PlaneDef関数	-	
	PList	-	
	PLocal	-	
	PLocal関数	-	
	Pls関数	-	
	PNumber関数	-	
	PosFound関数	-	
	Power	-	
	Power関数	-	
	PPls関数	-	
	Print	-	
	Print#	-	
	PTCLR	-	
	PTPBoost	-	
	PTPBoostOK関数	-	
	PTPBoost関数	-	
	PTPTime関数	-	
	PTran	-	

命令		互換性	備考
	PTRQ	-	
	PTRQ関数	-	
	Pulse	-	
	Pulse関数	-	
Q	QP	-	
	Quit	-	
R	RadToDeg関数	-	
	Randomize	-	
	Range	-	
	Read	-	
	ReadBin	-	
	Real	-	
	RealPls関数	-	
	RealPos関数	-	
	RealTorque	-	
	Redim	-	
	Reset	-	
	Resume	-	
	Return	-	
	RobotInfo関数	-	
	RobotInfo\$関数	-	
	RobotModel\$関数	-	
	RobotName\$関数	-	
	RobotSerial\$関数	-	
	RobotType関数	-	
	RSet\$関数	-	
	RShift関数	-	
	RTrim\$関数	-	

命令		互換性	備考
S	SafetyOn関数	-	
	SavePoints	-	
	Select...Send	-	
	Sense	-	
	SetCom	-	
	SetInW	-	
	SetIn	-	
	SetNet	-	
	SetSw	-	
	SFree	-	
	SFree関数	-	
	Sgn関数	-	
	Signal	-	
	Sin関数	-	
	SLock	-	
	SoftCP	-	
	SoftCP関数	-	
	Space\$関数	-	
	Speed	-	
	SpeedR	-	
	SpeedR関数	-	
	SpeedS	-	
	SpeedS関数	-	
	Speed関数	-	
	SPELCom_Event	-	
	Sqr関数	-	
	Stat関数	-	
	Str\$関数	-	

命令		互換性	備考
	String	-	
	Sw関数	-	
	SyncLock	-	
	SyncUnlock	-	
	SysConfig	-	
	SysErr関数	-	
T	Tab\$関数	-	
	Tan関数	-	
	TargetOK関数	-	
	TaskDone関数	-	
	TaskInfo関数	-	
	TaskInfo\$関数	-	
	TaskState	-	
	TaskState関数	-	
	TaskWait	-	
	TC	-	
	TCLim	-	
	TCLim関数	-	
	TCSpeed	-	
	TCSpeed関数	-	
	TGo	+	動作モード指定を追加
	TillOn関数	-	
	Time	-	
	Time関数	-	
	Time\$関数	-	
	TLClr	-	
	TIDef関数	-	
	TLSet	-	

命令		互換性	備考
	TLSet関数	-	
	TMOut	-	
	TMove	-	
	TmReset	-	
	Tmr関数	-	
	Toff	-	
	Ton	-	
	Tool	-	
	Tool関数	-	
	Trap	-	
	Trim\$関数	-	
	Tw関数	-	
U	UBound関数	-	
	UCase\$関数	-	
V	Val関数	-	
W	Wait	-	
	WaitNet	-	
	WaitPos	-	
	WaitSig	-	
	Weight	-	
	Weight関数	-	
	Where	-	
	Wrist	-	
	Wrist関数	-	
	Write	-	
	WriteBin	-	
X	Xor	-	
	Xqt	-	

命令		互換性	備考
	XY関数	-	
	XYLim	-	
	XYLim関数	-	
	XYLimClr	-	
	XYLimDef	-	
	XYLimDef関数	-	

5.2 B-2: EPSON RC+ 5.0 からの互換性に関する注意

5.2.1 概要

EPSON RC+ 8.0は、EPSON RC+ 7.0の互換性を保ちます。

すでに、RC180コントローラーで、EPSON RC+ 5.0をご利用の方が、RC800シリーズでEPSON RC+ 7.0, Epson RC+ 8.0を使う場合に読んでいただきたい内容をまとめました。

EPSON RC+ 7.0, Epson RC+ 8.0を使用する場合は、ハードウェアの違い、対応マネジュレーターや関節数の違い、ソフトウェアの実行環境の違いなどにより、EPSON RC+ 5.0とは異なっている部分があります。あらかじめその内容をご理解いただき、安全にロボットをご使用してください。

EPSON RC+ 7.0, Epson RC+ 8.0は、できるだけ今までの製品と互換性を保ちながら改善し、先進のソフトウェア技術を取り入れて設計しています。しかしながら、ロボットコントローラーに特化し、より使いやすくなるために、これまでのEPSON RC+ 5.0との互換性を全て満たしてはいない部分や、削除された部分があります。

以下の互換性に関する記述は、EPSON RC+ 5.0を元に、EPSON RC +7.0, Epson RC+ 8.0と比較をしています。

5.2.2 全般にわたる違い

EPSON RC+ 5.0と、EPSON RC+ 7.0, Epson RC+ 8.0の全般にわたる違いは下記のとおりです。

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 5.0
タスク数	最大32タスク (バックグラウンドタスクは最大16タスク)	最大16タスク
タスクの種類	NoPauseタスクの指定が可 NoEmgAbortタスクの指定が可 バックグラウンドタスクの指定が可	NoPauseタスクの指定が可 NoEmgAbortタスクの指定が可
TRAP ERRORなどの特殊なTRAP	サポートされています	サポートされません
TRAP トラップ番号で起動されるタスク	専用のタスク番号	専用のタスク番号
マルチマネジュレーター	サポートされています	サポートされません
ロボット番号	1～16	1
Real型の有効桁数	6桁	6桁
Double型の有効桁数	14桁	14桁
配列要素数	文字列変数以外 ローカル変数: 2,000 グローバル変数: 100,000 モジュール変数: 100,000 バックアップ変数: 4,000	文字列変数以外 ローカル変数: 1,000 グローバル変数: 10,000 モジュール変数: 10,000 バックアップ変数: 1,000

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 5.0
	文字列変数 ローカル変数: 200 グローバル変数: 10,000 モジュール変数: 10,000 バックアップ変数: 400	文字列変数 ローカル変数: 100 グローバル変数: 1,000 モジュール変数: 1,000 バックアップ変数: 100
デバイス番号	21:PC 22:リモート 20:TP4	21:PC 22:リモート 23:OP
コントロールデバイス	リモートI/O PC リモートEthernet TP4	リモートI/O PC OP1 リモートEthernet
タイマー番号の範囲	0～63	0～15
プログラム容量	8MB	4MB
SyncLock, SyncUnlock用のシグナル番号の範囲	0～63	0～15
WaitSig, Signal用のシグナル番号の範囲	0～63	0～5
メモリーI/O点数	1024	256
I/Oポートの番号	EPSON RC+ 5.0と共通	
イーサネットのポート番号	201～216	201～208
リモート入出力割りあて	標準で機能割りあて済	標準で機能割りあて済
RS-232C通信ポート番号 SPEL+制御部	1～8, 1001～1008	1～8
RS-232C通信ポートのOpenCom実行	必須	必須
ファイルへの入出力	サポートされています	サポートされません
ファイルアクセスのファイル番号	30～63	サポートされません
データベースアクセス番号	501～508	サポートされません
VisionGuide	スマートカメラタイプ フレームグラバータイプ	スマートカメラタイプ
コンベヤトラッキング	サポートされています	サポートされません
PGロボット	サポートされています	サポートされません
OCR	サポートされています	サポートされません

項目	EPSON RC+ 7.0, Epson RC+ 8.0	EPSON RC+ 5.0
セキュリティー	サポートされています	サポートされません
VBGuide 5.0 (RC+ API)	サポートされています	Lite版がサポートされています
フィールドバスI/Oの扱い	通常のI/O命令を使用	通常のI/O命令を使用
フィールドバス マスター	応答性が保証されません	サポートされません
フィールドバス スレーブ	応答性が保証されています	応答性が保証されています
GUI Builder	サポートされています	サポートされません
エラー番号	EPSON RC+ 5.0と共通	

5.2.3 命令の互換性一覧

- +: 機能拡張、機能変更されているが上位互換が保たれている命令
- -: 変更がない命令
- !: 機能変更または、文法変更があり注意が必要な命令
- !!: 大幅な変更があり注意が必要な命令
- ×: 削除された命令

命令		互換性	備考
A	Abs関数	-	
	Accel	-	
	Accel関数	-	
	AccelMax関数	-	
	AccelR	-	
	AccelR関数	-	
	AccelS	-	
	AccelS関数	-	
	Acos関数	-	
	AglToPls関数	-	
	Agl関数	-	
	AlignECP関数	-	
	Align関数	-	

命令		互換性	備考
	And	-	
	Arc	-	
	Arc3	-	
	Arch	-	
	Arch関数	-	
	Arm	-	
	ArmClr	-	
	ArmDef関数	-	
	ArmSet	-	
	ArmSet関数	-	
	Arm関数	-	
	Asc関数	-	
	Asin関数	-	
	Atan2関数	-	
	Atan関数	-	
	ATCLR	-	
	ATRQ	-	
	ATRQ関数	-	
B	Base	-	
	Base関数	-	
	BClr関数	-	
	BGo	+	動作モード指定を追加
	BMove	-	
	Boolean	-	
	Box	+	ロボット番号の指定を追加
	Box関数	+	ロボット番号の指定を追加
	BoxClr関数	+	ロボット番号の指定を追加
	BoxDef関数	+	ロボット番号の指定を追加

命令		互換性	備考
	Brake	-	
	Brake関数	-	
	BSet関数	-	
	BTst関数	-	
	Byte	-	
C	Call	+	外部ファンクション呼び出し
	ChkCom関数	-	
	ChkNet関数	-	
	Chr\$関数	-	
	ClearPoints	-	
	CloseCom	-	
	CloseNet	-	
	Cls	-	
	Cos関数	-	
	CP	-	
	CP関数	-	
	CTReset	-	
	CtrlDev関数	!	コントローラデバイス変更
	CtrlInfo関数	-	
	Ctr関数	-	
	CurPos関数	-	
	Curve	-	
	CVMove	-	
	CX～CW	+	CR, CS, CT追加
	CX～CW関数	+	CR, CS, CT追加
D	Date	!	表示のみ
	Date\$関数	-	
	DegToRad関数	-	

命令		互換性	備考
	DispDev	-	
	DispDev関数	-	
	Dist関数	-	
	Do...Loop	-	
	Double	-	
E	ECP	-	
	ECPClr	-	
	EcpDef関数	-	
	ECPSet	-	
	ECPSet関数	-	
	ECP関数	-	
	ElapsedTime関数	-	
	Elbow	-	
	Elbow関数	-	
	Era関数	-	
	EResume	-	
	Erf\$関数	-	
	Erl関数	-	
	ErrMsg\$関数	-	
	Error	-	
	ErrorOn関数	-	
	Err関数	-	
	Ert関数	-	
	EStopOn関数	-	
	Exit	-	
	Find	-	
	FindPos関数	-	
	Fine	-	

命令		互換性	備考
	Fine関数	-	
	Fix関数	-	
	FmtStr\$	-	
	For...Next	-	
	Function...Fend	-	
G	Global	-	
	Go	+	動作モード指定を追加
	Gosub...Return	-	
	Goto	-	
H	Halt	-	
	Hand	-	
	Hand関数	-	
	Here	-	
	Here関数	-	
	Hex\$関数	-	
	Home	-	
	HomeClr	-	
	HomeDef関数	-	
	HomeSet	-	
	HomeSet関数	-	
	HOrdr	-	
	HOrdr関数	-	
	Hour	-	
	Hour関数	-	
I	If...EndIf	-	
	In関数	-	
	InBCD関数	-	
	Inertia	-	

命令		互換性	備考
	Inertia関数	-	
	InPos関数	-	
	Input	-	
	Input#	+	デバイス番号追加
	InsideBox関数	!	ロボット番号、Allの指定を追加、Wait待ちできない
	InsidePlane関数	!	ロボット番号、Allの指定を追加、Wait待ちできない
	InStr関数	-	
	Integer	-	
	Int関数	-	
	InW関数	-	
	IOLabel\$関数	-	
	IONumber関数	-	
J	J1Flag	-	
	J1Flag関数	-	
	J2Flag	-	
	J2Flag関数	-	
	J4Flag	-	
	J4Flag関数	-	
	J6Flag	-	
	J6Flag関数	-	
	JA関数	-	
	Joint	-	
	JRange	-	
	JRange関数	-	
	JS関数	-	
	JTran	-	
	JT関数	-	
	Jump	+	動作モード指定を追加

命令		互換性	備考
	Jump3	+	
	Jump3CP	+	
L	LCase\$関数	-	
	Left\$関数	-	
	Len関数	-	
	LimZ	-	
	LimZ関数	-	
	Line Input	-	
	Line Input#	+	デバイス番号追加
	LJM関数	-	
	LoadPoints	-	
	Local	-	
	LocalClr	-	
	LocalDef関数	-	
	Local関数	-	
	Lof関数	-	
	Long	-	
	LSet\$関数	-	
	LShift関数	-	
	LTrim\$関数	-	
M	Mask	-	
	MemInW関数	-	
	MemIn関数	-	
	MemOff	-	
	MemOn	-	
	MemOut	-	
	MemOutW	-	
	MemSw関数	-	

命令		互換性	備考
	Mid\$関数	-	
	Mod	-	
	Motor	-	
	Motor関数	-	
	Move	-	
	MyTask関数	-	
N	Not	-	
O	Off	-	
	OLAccel	-	
	OLAccel関数	-	
	OLRate	-	
	OLRate関数	-	
	On	-	
	OnErr	-	
	OpBCD	-	
	OpenCom	-	
	OpenNet	-	
	Oport関数	-	
	Or	-	
	Out	-	
	OutW	-	
	OutW関数	-	
	Out関数	-	
P	P#	-	
	PAgl関数	-	
	Pallet	-	座標値指定の追加
	Pallet関数	-	
	ParseStr	-	

命令		互換性	備考
	ParseStr関数	-	
	Pass	+	
	Pause	-	
	PauseOn関数	-	
	PDef関数	-	
	PDel	-	
	PLabel	-	
	PLabel\$関数	-	
	Plane	+	ロボット番号の指定を追加
	Plane関数	+	ロボット番号の指定を追加
	PlaneClr	+	ロボット番号の指定を追加
	PlaneDef関数	+	ロボット番号の指定を追加
	PList	!	表示形式変更
	PLocal	-	
	PLocal関数	-	
	Pls関数	-	
	PNumber関数	-	
	PosFound関数	-	
	Power	-	
	Power関数	-	
	PPls関数	-	
	Print	-	
	Print#	+	デバイス番号変更
	PTCLR	-	
	PTPBoost	-	
	PTPBoostOK関数	-	
	PTPBoost関数	-	
	PTPTime関数	-	

命令		互換性	備考
	PTran	-	
	PTRQ	-	
	PTRQ関数	-	
	Pulse	-	
	Pulse関数	-	
Q	QP	-	
	Quit	-	
R	RadToDeg関数	-	
	Randomize	-	
	Range	-	
	Read	-	
	ReadBin	-	
	Real	-	
	RealPls関数	-	
	RealPos関数	-	
	RealTorque	-	
	Redim	-	
	Reset	-	
	ResetElapsedTime	-	
	Resume	-	
	Return	-	
	RobotInfo関数	+	情報の追加
	RobotInfo\$関数	+	デフォルトポイントファイル名の表示追加
	RobotModel\$関数	-	
	RobotName\$関数	-	
	RobotSerial\$関数	-	
	RobotType関数	-	
	RSet\$関数	-	

命令		互換性	備考
	RShift関数	-	
	RTrim\$関数	-	
S	SafetyOn関数	-	
	SavePoints	-	
	Select...Send	-	
	Sense	-	
	SetCom	-	
	SetInW	-	
	SetIn	-	
	SetNet	-	
	SetSw	-	
	SFree	-	
	SFree関数	-	
	Sgn関数	-	
	Signal	-	
	Sin関数	-	
	SLock	-	
	SoftCP	-	
	SoftCP関数	-	
	Space\$関数	-	
	Speed	-	
	SpeedR	-	
	SpeedR関数	-	
	SpeedS	-	
	SpeedS関数	-	
	Speed関数	-	
	SPELCom_Event	-	
	Sqr関数	-	

命令		互換性	備考
	Stat関数	+	情報の追加
	Str\$関数	-	
	String	-	
	Sw関数	-	
	SyncLock	-	
	SyncUnlock	-	
	SysConfig	+	情報の追加
	SysErr関数	+	ワーニング取得機能追加
T	Tab\$関数	-	
	Tan関数	-	
	TargetOK関数	-	
	TaskDone関数	-	
	TaskInfo関数	-	
	TaskInfo\$関数	-	
	TaskState	+	バックグラウンドタスクの表示追加
	TaskState関数	-	
	TaskWait	-	
	TC	-	
	TCLim	-	
	TCLim関数	-	
	TCSpeed	-	
	TCSpeed関数	-	
	TGo	+	動作モード指定を追加
	TillOn関数	-	
	Time	!	表示のみ
	Time関数	-	
	Time\$関数	-	
	TLClr	-	

命令		互換性	備考
	TlDef関数	-	
	TlSet	-	
	TlSet関数	-	
	TMOut	-	
	TMove	-	
	TmReset	-	
	Tmr関数	-	
	Toff	-	
	Ton	-	
	Tool	-	
	Tool関数	-	
	Trap	!	コントローラー状態をトリガーとするTrapを追加
	Trim\$関数	-	
	Tw関数	-	
U	UBound関数	-	
	UCase\$関数	-	
V	Val関数	-	
W	Wait	!	グローバル変数などが待ち条件に追加
	WaitNet	-	
	WaitPos	-	
	WaitSig	-	
	Weight	+	S,Tの指定追加
	Weight関数	+	S,Tの指定追加
	Where	-	
	Wrist	-	
	Wrist関数	-	
	Write	-	
	WriteBin	-	

命令		互換性	備考
X	Xor	-	
	Xqt	-	
	XY関数	-	
	XYLim	-	
	XYLim関数	-	
	XYLimClr	-	
	XYLimDef	-	
	XYLimDef関数	-	

5.2.4 EPSON RC+ Ver.4.*からの移行コマンド一覧 (EPSON RC+5.0未サポート)

AOpen	BOpen	Calib
CalPls	ChDir	ChDrive
Close	Cnv_AbortTrack	Cnv_Downstream
Cnv_Fine	Cnv_Fine関数	Cnv_Name\$関数
Cnv_Number関数	Cnv_Point関数	Cnv_PosErr関数
Cnv_Pulse関数	Cnv_QueueAdd	Cnv_QueueGet関数
Cnv_QueueLen関数	Cnv_QueueList	Cnv_QueueMove
Cnv_QueueReject	Cnv_QueueReject関数	Cnv_QueueRemove
Cnv_QueueUserData	Cnv_QueueUserData関数	Cnv_RobotConveyor関数
Cnv_Speed関数	Cnv_Trigger	Cnv_Upstream関数
Cont	Copy	CurDir\$関数
CurDrive\$関数	Declare	Del
Eof関数	Eval関数	FbusIO_GetBusStatus関数
FbusIO_GetDeviceStatus関数	FbusIO_SendMsg	FileDateTime\$関数
FileExists関数	FileLen関数	FolderExists関数
FreeFile関数	GetCurrentUser\$	Hofs
Hofs関数	ImportPoints	InputBox

LogIn関数	MCalComplete関数	MCal
MCordr	MCordr関数	MKDir
MsgBox	Recover関数	Rename
RenDir	Restart	RmDir
Robot	Robot関数	ROpen
RunDialog	Seek	Shutdown
UOpen	WOpen	

6. Appendix C: Epson RC+8.0のコマンド

6.1 C-1: EPSON RC+ 4.0以降で追加されたコマンド一覧

A		
AbortMotion	AccelMax関数	AglToPls関数
AIO_Out	AIO_Out関数	AIO_OutW
AIO_OutW関数	AIO_Set	AIO_Set関数
AIO_TrackingSet	AIO_TrackingStart	AIO_TrackingEnd
AIO_TrackingOn関数	AIO_In関数	AIO_InW関数
Align関数	AlignECP関数	AreaCorrection関数
AreaCorrectionClr	AreaCorrectionDef関数	AreaCorrectionInv関数
AreaCorrectionOffset関数	AreaCorrectionSet	ArmCalib
ArmCalib関数	ArmCalibSet	ArmCalibSet関数
ArmCalibClr	ArmCalibDef関数	ArmDef関数
ATCLR	AtHome関数	ATRQ
ATRQ関数	AutoLJM	AutoLJM関数
AutoSTM	AutoSTM関数	AvoidSingularity
AvoidSingularity関数		

B		
BClr関数	BClr64関数	Box
Box関数	BoxClr関数	BoxDef関数
Brake関数	BSet関数	BSet64関数
BTst関数	BTst64関数	

C		
ChDisk	ChkCom関数	ChkNet関数
CloseCom	CloseDB	CloseNet
Cls	CP	CP関数
CP_Offset	CP_Offset関数	CR
CR関数	CS	CS関数

C		
CT	CT関数	CtrlDev関数
Curve	CVMove	Cnv_Accel
Cnv_Accel関数	Cnv_AccelLim	Cnv_AccelLim関数
Cnv_Adjust	Cnv_AdjustClear	Cnv_AdjustGet
Cnv_AdjustSet	Cnv_DownStream	Cnv_Mode
Cnv_Mode関数	Cnv_OffsetAngle	Cnv_OffsetAngle関数
Cnv_PosErrOffset	Cnv_Upstream	CollisionDetect
CollisionDetect関数		

D		
DegToRad関数	DeleteDB	DiffPoint関数
DispDev	DispDev関数	Dist関数

E		
EcpDef関数	ElapsedTime関数	EResume
Errb関数	ErrorOn関数	Error
EStopOn関数	Exit	ExportPoints

F		
FindPos関数	Find	FineDist
FineDist関数	FineStatus関数Fix関数	Flush
Fmtstr		

G	
GetRobotInsideBox関数	GetRobotInsidePlane関数

H		
Hand_On	Hand_On関数	Hand_Off
Hand_Off関数	Hand_TW関数	Hand_Def関数
Hand_Type関数	Hand_Label\$関数	Hand_Number関数

H		
HealthCalcPeriod	HealthCalcPeriod関数	HealthCtrlAlarmOn関数
HealthCtrlInfo	HealthCtrlInfo関数	HealthCtrlRateOffset
HealthCtrlReset	HealthCtrlWarningEnable	HealthCtrlWarningEnable関数
HealthRateCtrlInfo関数	HealthRateRBInfo関数	HealthRBAlarmOn関数
HealthRBAalysis	HealthRBAalysis関数	HealthRBDistance
HealthRBDistance関数	HealthRBInfo	HealthRBInfo関数
HealthRBRateOffset	HealthRBReset	HealthRBSpeed
HealthRBSpeed関数	HealthRBStart	HealthRBStop
HealthRBTRQ	HealthRBTRQ関数	HealthRBWarningEnable
HealthRBWarningEnable関数	Here	Here関数
Hex\$関数	HofsJointAccuracy	HomeClr
HomeDef関数		

I		
InReal関数	InsideBox関数	InsidePlane関数
InStr関数	IODef関数	IOLabel\$関数
IONumber関数		

J		
J1Angle	J1Angle関数	J4Angle
JA関数	Joint	JointAccuracy
JointAccuracy関数	JumpTLZ	JTran

L		
LatchEnable	LatchState関数	LatchPos関数
LimZMargin	LimZMargin関数	LimitTorque
LimitTorque関数	LimitTorqueLP	LimitTorqueLP関数
LimitTorqueStop	LimitTorqueStop関数	LimitTorqueStopLP
LimitTorqueStopLP関数	LJM関数	LocalDef関数

L		
LShift64関数		

M		
MemInW関数	MemOutW	MHour関数

O		
OLAccel	OLAccel関数	OpenCom
OpenCom関数	OpenDB	OpenNet
OpenNet関数	OutReal	OutReal関数

P		
P#	PalletClr	PauseOn関数
PDef関数	PDel	PDescription
PDescription関数	PerformMode	PerformMode関数
PG_FastStop	PG_LSpeed	PG_LSpeed関数
PG_Scan	PG_SlowStop	PLabel
PLabel\$関数	PlaneClr	PlaneDef
Plane	Plane関数	PList
PLocal	PLocal関数	PNumber関数
PosFound関数	PTCLR	PTPBoostOK関数
PTPTIME関数	PTran	PTRQ
PTRQ関数		

Q		
QPDECEL	QPDECEL関数	QPDECELS
QPDECELS関数		

R		
RadToDeg関数	Randomize	ReadBin
Read	RealAccel関数	RealPls関数

R		
RealPos関数	RealTorque関数	RecoverPos関数
Recover	Redim	ResetElapsedTime
Rnd関数	RobotInfo関数	RobotInfo\$関数
RobotModel\$関数	RobotName\$関数	RobotSerial\$関数
RobotType関数	ROTOK関数	RShift64関数

S		
SafetyOn関数	SelectDB関数	SetCom
SetInW	SetIn	SetNet
SetSw	SF_GetParam関数	SF_GetParam\$関数
SF_GetStatus関数	SF_LimitSpeedS	SF_LimitSpeedS関数
SF_LimitSpeedEnable	SF_LimitSpeedEnable関数	SF_PeakSpeedS
SF_PeakSpeedS関数	SF_PeakSpeedSClear	SF_RealSpeedS
SF_RealSpeedS関数	SF_SetIn	SF_SetSw
Shutdown関数	SimGet	SimSet
SingularityAngle	SingularityAngle関数	SingularityDist
SingularityDist関数	SingularitySpeed	SingularitySpeed関数
SoftCP	SoftCP関数	SpeedFactor
SpeedFactor関数	StartMain	STM関数
SyncRobots	SyncRobots関数	SysErr関数

T		
Tab\$関数	TargetOK関数	TaskDone関数
TaskInfo関数	TaskInfo\$関数	TaskState
TaskState関数	TaskWait	TC
TCLim	TCLim関数	TCSpeed
TCSpeed関数	TeachOn関数	TillOn関数
TIDef関数	Toff	Ton

U	
UBound関数	UpdateDB

V		
VDefArm	VDefLocal	VDefSetMotionRange
VDefGetMotionRange	VDefTool	VGoCenter
VSD	VSD関数	VxCalib
VxCalDelete	VxCalLoad	VxCalInfo関数
VxCalSave		

W		
WaitNet	WaitPos	Where
WindosStatus関数	WorkQue_Add	WorkQue_AutoRemove
WorkQue_AutoRemove関数	WorkQue_Get関数	WorkQue_Len関数
WorkQue_List	WorkQue_Reject	WorkQue_Reject関数
WorkQue_Remove	WorkQue_Sort	WorkQue_Sort関数
WorkQue_UserData	WorkQue_UserData関数	WriteBin
Write		

X		
XYLimClr	XYLimDef	XY関数

6.2 C-2: Epson RC+ 8.0のバージョンごとに追加されたコマンド一覧

Epson RC+ 8.0のバージョン	追加されたコマンド
Ver.8.0.0.0	ClearHistory, ContManualRecover, EncTemper, EncTemper関数, History, SpeedRLimitation, SpeedRLimitation関数
Ver.8.1.0.0	ArmLib, ArmReserve, CtrlPref, GetIODef, GetProjectInfo, GetSetNet, GUIVer\$, JogPanel, LocalReserve, SetIODef, SignalReserve, SyncLockReserve, TaskReserve, TeachPoint, TimerReserve, TLReserve, ToolLib, ToolWizard, UserErrorDef, UserErrorLabel\$, UserErrorNumber

Epson RC+ 8.0の バージョン	追加されたコマンド
Ver.8.1.1.0	AutoSTM, AutoSTM関数, STM関数, SF_SetIn, SF_SetSw

6.3 C-3: EPSON RC+ 7.0のバージョンごとに追加されたコマンド一覧

EPSON RC+ 6.0, 5.0, 4.0 共通

EPSON RC+ 7.0のバージョン	追加されたコマンド
Ver.7.5.4	Cnv_AccelLim, Cnv_AccelLim関数 SF_GetParam関数, SF_GetParam\$関数 SF_GetStatus関数, SF_LimitSpeedS SF_LimitSpeedS関数, SF_LimitSpeedEnable SF_LimitSpeedEnable関数, SF_PeakSpeedS SF_PeakSpeedS関数, SF_PeakSpeedSClear SF_RealSpeedS, SF_RealSpeedS関数
Ver.7.5.3	AreaCorrection関数, AreaCorrectionClr AreaCorrectionDef関数, AreaCorrectionInv関数 AreaCorrectionOffset関数, AreaCorrectionSet Cnv_PosErrOffset
Ver.7.5.2	XYLimMode, XYLimMode関数
Ver.7.5.1	ArmCalib, ArmCalib関数 ArmCalibSet, ArmCalibSet関数 ArmCalibClr, ArmCalibDef関数 JointAccuracy, JointAccuracy関数 HofsJointAccuracy, Hand_On Hand_On関数, Hand_Off Hand_Off関数, Hand_TW関数 Hand_Def関数, Hand_Type関数 Hand_Label\$関数, Hand_Number関数 Cnv_Adjust, Cnv_AdjustClear Cnv_AdjustGet, Cnv_AdjustSet DiffPoint関数, ROTOK関数
Ver.7.4.3	AIO_TrackingSet, AIO_TrackingStart AIO_TrackingEnd, AIO_TrackingOn関数
Ver.7.4.1	AutoOrientationFlag, AutoOrientationFlag関数
Ver.7.3.4	SimGet, SimSet
Ver.7.3.3	HealthCtrlWarningEnable, HealthCtrlWarningEnable関数 HealthRBWarningEnable, HealthRBWarningEnable関数
Ver.7.3.2	PDescription, PDescription関数

EPSON RC+ 7.0のバージョン	追加されたコマンド
Ver.7.3.1	AIO_Out, AIO_Out関数 AIO_OutW, AIO_OutW関数 AIO_Set, AIO_Set関数AIO_In関数 AIO_InW関数, HealthCalcPeriod HealthCalcPeriod関数
Ver.7.3.0	VDefTool, VDefArm VDefLocal, VGoCenter VDefSetMotionRange, VDefGetMotionRange
Ver.7.2.0	CP_Offset, CP_Offset関数 HealthCtrlAlarmOn関数, HealthCtrlInfo HealthCtrlInfo関数, HealthCtrlRateOffset HealthCtrlReset, HealthRateCtrlInfo関数 HealthRateRBInfo関数, HealthRBAlarmOn関数 HealthRBAnalysis, HealthRBAnalysis関数 HealthRBDistance, HealthRBDistance関数 HealthRBInfo, HealthRBInfo関数 HealthRBRateOffset, HealthRBReset HealthRBSpeed HealthRBSpeed関数, HealthRBStart HealthRBStop, HealthRBTRQ HealthRBTRQ関数, J4Angle JumpTLZ, LimitTorqueLP LimitTorqueLP関数, LimitTorqueStop LimitTorqueStop関数, LimitTorqueStopLP LimitTorqueStopLP関数, VSD VSD関数
Ver.7.1.4	CollisionDetect関数
Ver.7.1.3	CollisionDetect, MHour関数
Ver.7.1.2	SingularityDist, SingularityDist関数 ExportPoints
Ver.7.1.0	BClr64関数, BSet64関数 BTst64関数, FineDist FineDist関数, FineStatus関数 Fmtstr, IODef関数 LShift64関数, RealAccel関数 RShift64関数, WorkQue_Add WorkQue_AutoRemove, WorkQue_AutoRemove関数 WorkQue_Get関数, WorkQue_Len関数 WorkQue_List, WorkQue_Reject WorkQue_Reject関数, WorkQue_Remove WorkQue_Sort, WorkQue_Sort関数 WorkQue_UserData, WorkQue_UserData関数
Ver.7.0.3	PerformMode, PerformMode関数

EPSON RC+7.0 Ver.7.0.0の場合は、
EPSON RC+ 6.0, 5.0と追加されたコマンドが異なります。

EPSON RC+ 6.0, 5.0に対して追加されたコマンド

EPSON RC+ 7.0のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0
Ver.7.0.0	AutoLJM AutoLJM関数 AvoidSingularity AvoidSingularity関数	AbortMotion AutoLJM AutoLJM関数 AvoidSingularity AvoidSingularity関数
	Cnv_Accel Cnv_Accel関数 Cnv_DownStream Cnv_Mode Cnv_Mode関数 Cnv_Upstream	ChDisk CloseDB Cnv_Accel Cnv_Accel関数 Cnv_DownStream Cnv_Mode Cnv_Mode関数 Cnv_Upstream
		CR CR関数 CS CS関数 CT CT関数
	DeleteDB	DeleteDB
	ElapsedTime関数 Errb関数	Errb関数 Flush GetRobotInsideBox関数 GetRobotInsidePlane関数 J1Angle J1Angle関数
	LimZMargin LimZMargin関数 LimitTorque LimitTorque関数	LimZMargin LimZMargin関数 LimitTorque LimitTorque関数 OpenDB

EPSON RC+ 7.0のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0
	PalletClr	PalletClr PG_FastStop PG_LSpeed PG_LSpeed関数 PG_Scan PG_SlowStop QPDECELR QPDECELR関数 QPDECELS QPDECELS関数 RecoverPos関数 Recover
	ResetElapsedTime	SelectDB関数 Shutdown関数
	SingularityAngle SingularityAngle関数 SingularitySpeed SingularitySpeed関数 SpeedFactor SpeedFactor関数	SingularityAngle SingularityAngle関数 SingularitySpeed SingularitySpeed関数 SpeedFactor SpeedFactor関数
		StartMain SyncRobots SyncRobots関数 TeachOn関数
	UpdateDB	UpdateDB WindosStatus関数

EPSON RC+ 4.0に対して追加されたコマンド

EPSON RC+ 7.0のバージョン	EPSON RC+ 4.0
Ver.7.0.0	AbortMotion, AccelMax関数 AglToPls関数, Align関数 AlignECP関数, ArmDef関数 ATCLR, AtHome関数 ATRQ, ATRQ関数 AutoLJM, AutoLJM関数 AvoidSingularity, AvoidSingularity関数
	BClr関数, Box Box関数, BoxClr関数 BoxDef関数, Brake関数 BSet関数, BTst 関数
	ChDisk, ChkCom関数 ChkNet関数, CloseCom CloseDB, CloseNet Cls, CP CP関数, CR CR関数, CS CS関数, CT CT関数, CtrlDev関数 Curve, CVMove Cnv_Accel, Cnv_Accel関数Cnv_DownStream Cnv_Mode Cnv_Mode関数, Cnv_OffsetAngle Cnv_OffsetAngle関数, Cnv_Upstream
	DegToRad関数, DeleteDB DispDev, DispDev関数 Dist関数
	EcpDef関数, EResume Errb関数, ErrorOn関数 Error, EStopOn関数 Exit
	FindPos関数, Find FineStatus関数, Fix関数 Flush
	GetRobotInsideBox関数, GetRobotInsidePlane関数
	Here, Here関数 Hex\$関数, HomeClr HomeDef関数
	InReal関数, InsideBox関数 InsidePlane関数, InStr関数 IOLabel\$関数, IONumber関数

EPSON RC+ 7.0のバージョン	EPSON RC+ 4.0
	J1Angle, J1Angle関数 JA関数, Joint JTran
	LatchEnable, LatchState関数 LatchPos関数, LimZMargin LimZMargin関数, LimitTorque LimitTorque関数, LJM関数 LocalDef関数
	MemInW関数, MemOutW
	OLAccel, OLAccel関数 OpenCom, OpenCom関数 OpenDB, OpenNet OpenNet関数, OutReal OutReal関数
	P#, PalletClr PauseOn関数, PDef関数 PDel, PG_FastStop PG_LSpeed, PG_LSpeed関数 PG_Scan, PG_SlowStop PLabel, PLabel\$関数 PlaneClr, PlaneDef Plane, Plane関数 PList, PLocal PLocal関数, PNumber関数 PosFound関数, PTCLR PTPBoostOK関数, PTPTime関数 PTran, PTRQ PTRQ関数
	QPDECELR, QPDECELR関数 QPDECELS, QPDECELS関数
	RadToDeg関数, Randomize ReadBin, Read RealPls関数, RealPos関数 RealTorque関数, RecoverPos関数 Recover, Redim RobotInfo関数, RobotInfo\$ 関数RobotModel\$関数, RobotName\$関数 RobotSerial\$関数, RobotType関数

EPSON RC+ 7.0のバージョン	EPSON RC+ 4.0
	SafetyOn関数, SelectDB関数 SetCom, SetInW SetIn, SetNet SetSw, Shutdown関数 SingularityAngle, SingularityAngle関数 SingularitySpeed, SingularitySpeed関数 SoftCP, SoftCP関数 SpeedFactor, SpeedFactor関数 StartMain, SyncRobots SyncRobots関数, SysErr関数
	Tab\$関数, TargetOK関数 TaskDone関数, TaskInfo関数 TaskInfo\$関数, TaskState TaskState関数, TaskWait TC, TCLim TCLim関数, TCSpeed TCSpeed関数, TeachOn関数 TillOn関数, TlDef関数 Toff, Ton
	UBound関数, UpdateDB
	VxCalib, VxCalDelete VxCalLoad, VxCalInfo関数 VxCalSave, VxTrans関数
	WaitNet, WaitPos Where, WindosStatus関数 WriteBin, Write
	XYLimClr, XYLimDef XY関数

6.4 C-4: EPSON RC+ 7.0のバージョンごとに削除されたコマンド一覧

EPSON RC+ 6.0, 5.0, 4.0に対して削除されたコマンド

EPSON RC+ 7.0のバージョン	EPSON RC+ 6.0	EPSON RC+ 5.0	EPSON RC+ 4.0
Ver.7.1.2	SetLCD	SetLCD	SetLCD
Ver.7.0.0	Dir Type	-	Dir Type

7. Appendix D: 定数

7.1 Appendix D: 定数

SPELプログラム中で使用可能な、定数があります。プロジェクトビルドでは、定数値が使用されます。

定数名	値	用法
TRUE	-1	論理式
FALSE	0	論理式
High	1	
Low	0	
Off	0	
On	1	
Above	1	
Below	2	
NoFlip	1	
Flip	2	
Righty	1	
Lefty	2	
J1	1	
J2	2	
J3	4	
J4	8	
J5	16	
J6	32	
J7	64	
CR	CHR\$(13)	
CRLF	CHR\$(13)+CHR\$(10)	
LF	CHR\$(10)	
MB_OK	0	MsgBox フラグ
MB_OKCANCEL	1	MsgBox フラグ
MB_ABORTRETRYIGNORE	2	MsgBox フラグ

定数名	値	用法
MB_YESNOCANCEL	3	MsgBox フラグ
MB_YESNO	4	MsgBox フラグ
MB_RETRYCANCEL	5	MsgBox フラグ
MB_ICONSTOP	16	MsgBox フラグ
MB_ICONQUESTION	32	MsgBox フラグ
MB_ICONEXCLAMATION	48	MsgBox フラグ
MB_ICONINFORMATION	64	MsgBox フラグ
MB_DEFBUTTON1	0	MsgBox フラグ
MB_DEFBUTTON2	256	MsgBox フラグ
IDOK	1	MsgBox 戻り値
IDCANCEL	2	MsgBox 戻り値
IDABORT	3	MsgBox 戻り値
IDRETRY	4	MsgBox 戻り値
IDIGNORE	5	MsgBox 戻り値
IDYES	6	MsgBox 戻り値
IDNO	7	MsgBox 戻り値
ARC_SHORT	0	Arc, Arc3
ARC_LONG	1	Arc, Arc3
ARC_PLUS	1	Arc
ARC_MINUS	-1	Arc
BACKCOLORMODE_VISUALSTYLE	0	GUI Builder用
BACKCOLORMODE_USER	1	GUI Builder用
BORDERSTYLE_NONE	0	GUI Builder用
BORDERSTYLE_FIXEDSINGLE	1	GUI Builder用
BORDERSTYLE_FIXED3D	2	GUI Builder用
CNV_QUELEN_ALL	0	Cnv_QueLen
CNV_QUELEN_UPSTREAM	1	Cnv_QueLen
CNV_QUELEN_PICKUPAREA	2	Cnv_QueLen

定数名	値	用法
CNV_QUELEN_DOWNSTREAM	3	Cnv_QueLen
DEVID_SELF	21	CLS
DEVID_TP	24	CLS
DEVID_TP3	30	CLS
DIALOGRESULT_NOE	0	GUI Builder用
DIALOGRESULT_OK	1	GUI Builder用
DIALOGRESULT_CANCEL	2	GUI Builder用
DLG_IOMON	102	RunDialog
DLG_ROBOTMNG	100	RunDialog
DLG_VGUIDE	110	RunDialog
DOCK_NONE	0	GUI Builder用
DOCK_TOP	1	GUI Builder用
DOCK_BOTTOM	2	GUI Builder用
DOCK_LEFT	3	GUI Builder用
DOCK_RIGHT	4	GUI Builder用
DOCK_FILL	5	GUI Builder用
DROPDOWNSSTYLE_SIMPLE	0	GUI Builder用
DROPDOWNSSTYLE_DROPDOWN	1	GUI Builder用
DROPDOWNSSTYLE_DROPDOWNLIST	2	GUI Builder用
ERROR_DOINGMOTION	2999	GUI Builder用
ERROR_NOMOTION	2998	GUI Builder用
EVENTTASKTYPE_NORMAL	0	GUI Builder用
EVENTTASKTYPE_NOPAUSE	1	GUI Builder用
EVENTTASKTYPE_NOEMGABORT	2	GUI Builder用
FORMBORDERSTYLE_NONE	0	GUI Builder用
FORMBORDERSTYLE_FIXEDSINGLE	1	GUI Builder用
FORMBORDERSTYLE_FIXED3D	2	GUI Builder用
FORMBORDERSTYLE_FIXEDDIALOG	3	GUI Builder用

定数名	値	用法
FORMBORDERSTYLE_SIZABLE	4	GUI Builder用
IMAGEALIGN_TOPLEFT	1	GUI Builder用
IMAGEALIGN_TOPCENTER	2	GUI Builder用
IMAGEALIGN_TOPRIGHT	3	GUI Builder用
IMAGEALIGN_MIDDLELEFT	4	GUI Builder用
IMAGEALIGN_MIDDLECENTER	5	GUI Builder用
IMAGEALIGN_MIDDLERIGHT	6	GUI Builder用
IMAGEALIGN_BOTTOMLEFT	7	GUI Builder用
IMAGEALIGN_BOTTOMCENTER	8	GUI Builder用
IMAGEALIGN_BOTTOMRIGHT	9	GUI Builder用
IOTYPE_INPUT	0	IOLabel関数
IOTYPE_OUTPUT	1	IOLabel関数
IOTYPE_MEMORY	2	IOLabel関数
IOSIZE_BIT	1	IOLabel関数
IOSIZE_BYTE	8	IOLabel関数
IOSIZE_WORD	16	IOLabel関数
LANGID_ENGLISH	0	ErrMsg\$
LANGID_JAPANESE	1	ErrMsg\$
LANGID_GERMAN	2	ErrMsg\$
LANGID_FRENCH	3	ErrMsg\$
LANGID_SIMPLIFIED_CHINESE	4	ErrMsg\$
LANGID_TRADITIONAL_CHINESE	5	ErrMsg\$
MODE_STANDARD	1	PerformMode
MODE_HIGH_SPEED	2	PerformMode
MODE_LOW_OSCILLATION	3	PerformMode
ORIENT_HORIZONTAL	0	GUI Builder用
ORIENT_VERTICAL	1	GUI Builder用
PROGRESSBAR_STYLE_BLOCKS	0	GUI Builder用

定数名	値	用法
PROGRESSBAR_STYLE_CONT	1	GUI Builder用
PROGRESSBAR_STYLE_MARQUEE	2	GUI Builder用
SCROLLBARS_NONE	0	GUI Builder用
SCROLLBARS_HORIZ	1	GUI Builder用
SCROLLBARS_VERT	2	GUI Builder用
SCROLLBARS_BOTH	3	GUI Builder用
SETLATCH_PORT_CU_0	24	SetLatch
SETLATCH_PORT_CU_1	25	SetLatch
SETLATCH_PORT_CU_2	26	SetLatch
SETLATCH_PORT_CU_3	27	SetLatch
SETLATCH_TRIGGERMODE_LEADINGEDGE	1	SetLatch
SETLATCH_TRIGGERMODE_TRAILINGEDGE	0	SetLatch
SHUTDOWN_ALL	0	Shutdown
SHUTDOWN_RESTART	1	Shutdown
SHUTDOWN_EPSONRC	2	Shutdown
SING_NONE	0	AvoidSingularity
SING_THRU	1	AvoidSingularity
SING_THRUROT	2	AvoidSingularity
SING_VSD	3	AvoidSingularity
SING_AUTO	4	AvoidSingularity
SIZEMODE_NORMAL	0	GUI Builder用
SIZEMODE_STRETCHIMAGE	1	GUI Builder用
SIZEMODE_AUTOSIZE	2	GUI Builder用
SIZEMODE_CENTERIMAGE	3	GUI Builder用
SIZEMODE_ZOOM	4	GUI Builder用
STARTPOSITION_MANUAL	0	GUI Builder用
STARTPOSITION_CENTERSCREEN	1	GUI Builder用
STARTPOSITION_CENTERPARENT	2	GUI Builder用

定数名	値	用法
TEXTALIGN_LEFT	1	GUI Builder用
TEXTALIGN_CENTER	2	GUI Builder用
TEXTALIGN_RIGHT	3	GUI Builder用
TEXTALIGN_TOPLEFT	1	GUI Builder用
TEXTALIGN_TOPCENTER	2	GUI Builder用
TEXTALIGN_TOPRIGHT	3	GUI Builder用
TEXTALIGN_MIDDLELEFT	4	GUI Builder用
TEXTALIGN_MIDDLECENTER	5	GUI Builder用
TEXTALIGN_MIDDLERIGHT	6	GUI Builder用
TEXTALIGN_BOTTOMLEFT	7	GUI Builder用
TEXTALIGN_BOTTOMCENTER	8	GUI Builder用
TEXTALIGN_BOTTOMRIGHT	9	GUI Builder用
TICKSTYLE_NONE	0	GUI Builder用
TICKSTYLE_TOPLEFT	1	GUI Builder用
TICKSTYLE_BOTTOMRIGHT	2	GUI Builder用
TICKSTYLE_BOTH	3	GUI Builder用
VISION_SORT_NONE	0	Vision Guide用
VISION_SORT_PIXELX	1	Vision Guide用
VISION_SORT_PIXELY	2	Vision Guide用
VISION_SORT_PIXELXY	3	Vision Guide用
VISION_SORT_CAMERAX	4	Vision Guide用
VISION_SORT_CAMERAY	5	Vision Guide用
VISION_SORT_CAMERAXY	6	Vision Guide用
VISION_SORT_ROBOTX	7	Vision Guide用
VISION_SORT_ROBOTY	8	Vision Guide用
VISION_SORT_ROBOTXY	9	Vision Guide用
VISION_SIZETO_FIND_ANY	0	Vision Guide用
VISION_SIZETO_FIND_LARGEST	1	Vision Guide用

定数名	値	用法
VISION_SIZEOFIND_SMALLEST	2	Vision Guide用
VISION_BACKCOLOR_NONE	0	Vision Guide用
VISION_BACKCOLOR_BLACK	1	Vision Guide用
VISION_BACKCOLOR_WHITE	2	Vision Guide用
VISION_CAMORIENT_STANDALONE	1	Vision Guide用
VISION_CAMORIENT_FIXEDDOWN	2	Vision Guide用
VISION_CAMORIENT_FIXEDUP	3	Vision Guide用
VISION_CAMORIENT_MOBILEJ2	4	Vision Guide用
VISION_CAMORIENT_MOBILEJ4	5	Vision Guide用
VISION_CAMORIENT_MOBILEJ5	6	Vision Guide用
VISION_CAMORIENT_MOBILEJ6	7	Vision Guide用
VISION_FOUNDCOLOR_LIGHTGREEN	1	Vision Guide用
VISION_FOUNDCOLOR_DARKGREEN	2	Vision Guide用
VISION_GRAPHICS_ALL	1	Vision Guide用
VISION_GRAPHICS_POONLY	2	Vision Guide用
VISION_GRAPHICS_NONE	3	Vision Guide用
VISION_OPERATION_OPEN	1	Vision Guide用
VISION_OPERATION_CLOSE	2	Vision Guide用
VISION_OPERATION_ERODE	3	Vision Guide用
VISION_OPERATION_DILATE	4	Vision Guide用
VISION_OPERATION_SMOOTH	5	Vision Guide用
VISION_OPERATION_SHARPEN1	6	Vision Guide用
VISION_OPERATION_SHARPEN2	7	Vision Guide用
VISION_OPERATION_HORIZEDGE	8	Vision Guide用
VISION_OPERATION_VERTEDGE	9	Vision Guide用
VISION_OPERATION_EDGEDETECT1	10	Vision Guide用
VISION_OPERATION_EDGEDETECT2	11	Vision Guide用
VISION_OPERATION_LAPLACE1	12	Vision Guide用

定数名	値	用法
VISION_OPERATION_LAPLACE2	13	Vision Guide用
VISION_OPERATION_THIN	14	Vision Guide用
VISION_OPERATION_THICKEN	15	Vision Guide用
VISION_OPERATION_BINARIZE	16	Vision Guide用
VISION_OPERATION_ROTATE	17	Vision Guide用
VISION_OPERATION_FLIPHORIZ	18	Vision Guide用
VISION_OPERATION_FLIPVERT	19	Vision Guide用
VISION_OPERATION_FLIPBOTH	20	Vision Guide用
VISION_OPERATION_COLORFILTER	21	Vision Guide用
VISION_OPERATION_SUBTRACTABS	22	Vision Guide用
VISION_OPERATION_ZOOM	23	Vision Guide用
VISION_ACQUIRE_NONE	0	Vision Guide用
VISION_ACQUIRE_STATIONARY	1	Vision Guide用
VISION_ACQUIRE_STROBED	2	Vision Guide用
VISION_TRIGGERMODE_LEADINGEDGE	1	Vision Guide用
VISION_TRIGGERMODE_TRAILINGEDGE	2	Vision Guide用
VISION_THRESHCOLOR_BLACK	1	Vision Guide用
VISION_THRESHCOLOR_WHITE	2	Vision Guide用
VISION_OBJTYPE_CORRELATIO	1	Vision Guide用
VISION_OBJTYPE_BLOB	2	Vision Guide用
VISION_OBJTYPE_EDGE	3	Vision Guide用
VISION_OBJTYPE_POLAR	4	Vision Guide用
VISION_OBJTYPE_LINE	5	Vision Guide用
VISION_OBJTYPE_POINT	6	Vision Guide用
VISION_OBJTYPE_FRAME	7	Vision Guide用
VISION_OBJTYPE_IMAGEOP	8	Vision Guide用
VISION_OBJTYPE_OCR	9	Vision Guide用
VISION_OBJTYPE_CODEREADER	10	Vision Guide用

定数名	値	用法
VISION_OBJTYPE_GEOMETRIC	11	Vision Guide用
VISION_DETAILLEVEL_MEDIUM	1	Vision Guide用
VISION_DETAILLEVEL_HIGH	2	Vision Guide用
VISION_DETAILLEVEL_VERYHIGH	3	Vision Guide用
VISION_IMAGESOURCE_CAMERA	1	Vision Guide用
VISION_IMAGESOURCE_FILE	2	Vision Guide用
VISION_CODETYPE_AUTO	0	Vision Guide用
VISION_CODETYPE_EAN13	2	Vision Guide用
VISION_CODETYPE_CODE39	3	Vision Guide用
VISION_CODETYPE_INTERLEAVED25	4	Vision Guide用
VISION_CODETYPE_CODE128	5	Vision Guide用
VISION_CODETYPE_CODABAR	6	Vision Guide用
VISION_CODETYPE_PDF417	8	Vision Guide用
VISION_CODETYPE_QR	10	Vision Guide用
VISION_CODETYPE_EAN8	13	Vision Guide用
VISION_CODETYPE_UPCA	18	Vision Guide用
VISION_CODETYPE_UPCE	19	Vision Guide用
VISION_CODETYPE_UPC	20	Vision Guide用
VISION_EDGETYPE_SINGLE	1	Vision Guide用
VISION_EDGETYPE_PAIR	2	Vision Guide用
VISION_IMAGECOLOR_ALL	1	Vision Guide用
VISION_IMAGECOLOR_RED	2	Vision Guide用
VISION_IMAGECOLOR_GREEN	3	Vision Guide用
VISION_IMAGECOLOR_BLUE	4	Vision Guide用
VISION_IMAGECOLOR_GRAYSCALE	5	Vision Guide用
VISION_POINTTYPE_POINT	0	Vision Guide用
VISION_POINTTYPE_ENDPOINT	1	Vision Guide用
VISION_POINTTYPE_MIDPOINT	2	Vision Guide用

定数名	値	用法
VISION_POINTTYPE_PERPTOLINE	3	Vision Guide用
VISION_POINTTYPE_STARTPOINT	4	Vision Guide用
VISION_POINTTYPE_PERPTOSTARTPOINT	5	Vision Guide用
VISION_POINTTYPE_PERPTOMIDPOINT	6	Vision Guide用
VISION_POINTTYPE_PERPTOENDPOINT	7	Vision Guide用
VISION_REFTYPE_TAUGHTPOINTS	1	Vision Guide用
VISION_REFTYPE_UPWARDCAMERA	2	Vision Guide用
VISION_IMAGESIZE_320X240	1	Vision Guide用
VISION_IMAGESIZE_640X480	2	Vision Guide用
VISION_IMAGESIZE_800X600	3	Vision Guide用
VISION_IMAGESIZE_1024X768	4	Vision Guide用
VISION_IMAGESIZE_1280X1024	5	Vision Guide用
VISION_IMAGESIZE_1600X1200	6	Vision Guide用
VISION_IMAGESIZE_2048X1536	7	Vision Guide用
VISION_IMAGESIZE_2560X1920	8	Vision Guide用
VISION_WINTYPE_RECTANGLE	1	Vision Guide用
VISION_WINTYPE_ROTATEDRECT	2	Vision Guide用
VISION_WINTYPE_CIRCLE	3	Vision Guide用
VISION_ORIENT_BOTH	1	Vision Guide用
VISION_ORIENT_HORIZ	2	Vision Guide用
VISION_ORIENT_VERT	3	Vision Guide用
VISION_DIRECTION_INSIDEOUT	1	Vision Guide用
VISION_DIRECTION_OUTSIDEIN	2	Vision Guide用
VISION_POLARITY_DARK	1	Vision Guide用
VISION_POLARITY_LIGHT	2	Vision Guide用
VISION_PASSTYPE_SOMEFOUND	1	Vision Guide用
VISION_PASSTYPE_ALLFOUND	2	Vision Guide用
VISION_PASSTYPE_SOMENOTFOUND	3	Vision Guide用

定数名	値	用法
VISION_PASSTYPE_ALLNOTFOUND	4	Vision Guide用
WIN_IOMON	-1	GUI Builder用
WIN_TASKMGR	-2	GUI Builder用
WIN_FORCEMON	-3	GUI Builder用
WIN_SIMULATOR	-4	GUI Builder用
WINDOWSTATE_NORMAL	0	WindowsStatus
WINDOWSTATE_MINIMIZED	1	WindowsStatus
WINDOWSTATE_MAXIMIZED	2	WindowsStatus
WithMove	0	Recover
WithoutMove	1	Recover
DRYRUNOFF	1	SF_GetParam関数
SLS_1_HAND_EN	2	SF_GetParam関数
SLS_1_SPEED	3	SF_GetParam関数
SLS_1_ELBOW_EN	4	SF_GetParam関数
SLS_1_JOINT_EN	5	SF_GetParam関数
SLS_1_JOINTSPEED	6	SF_GetParam関数
SLS_1_WRIST_EN	7	SF_GetParam関数
SLS_1_SHOULDER_EN	8	SF_GetParam関数
SLS_2_HAND_EN	9	SF_GetParam関数
SLS_2_SPEED	10	SF_GetParam関数
SLS_2_ELBOW_EN	11	SF_GetParam関数
SLS_2_JOINT_EN	12	SF_GetParam関数
SLS_2_JOINTSPEED	13	SF_GetParam関数
SLS_2_WRIST_EN	14	SF_GetParam関数
SLS_2_SHOULDER_EN	15	SF_GetParam関数
SLS_3_HAND_EN	16	SF_GetParam関数
SLS_3_SPEED	17	SF_GetParam関数
SLS_3_ELBOW_EN	18	SF_GetParam関数

定数名	値	用法
SLS_3_JOINT_EN	19	SF_GetParam関数
SLS_3_JOINTSPEED	20	SF_GetParam関数
SLS_3_WRIST_EN	21	SF_GetParam関数
SLS_3_SHOULDER_EN	22	SF_GetParam関数
SLS_T2_HAND_EN	23	SF_GetParam関数
SLS_T2_SPEED	24	SF_GetParam関数
SLS_T2_ELBOW_EN	25	SF_GetParam関数
SLS_T2_JOINT_EN	26	SF_GetParam関数
SLS_T2_JOINTSPEED	27	SF_GetParam関数
SLS_T2_WRIST_EN	28	SF_GetParam関数
SLS_T2_SHOULDER_EN	29	SF_GetParam関数
SLS_T_SPEED	30	SF_GetParam関数
SLS_T_JOINT_EN	31	SF_GetParam関数
SLS_T_JOINTSPEED	32	SF_GetParam関数
SLS_HAND_OFS_X	33	SF_GetParam関数
SLS_HAND_OFS_Y	34	SF_GetParam関数
SLS_HAND_OFS_Z	35	SF_GetParam関数
SLS_1_DELAY	36	SF_GetParam関数
SLS_2_DELAY	37	SF_GetParam関数
SLS_3_DELAY	38	SF_GetParam関数
SLS_JOINT_POS_EN	39	SF_GetParam関数
SLS_JOINT_POS_ANGLE	40	SF_GetParam関数
SLP_A_XU_EN	41	SF_GetParam関数
SLP_A_XU_POS	42	SF_GetParam関数
SLP_A_XL_EN	43	SF_GetParam関数
SLP_A_XL_POS	44	SF_GetParam関数
SLP_A_YU_EN	45	SF_GetParam関数
SLP_A_YU_POS	46	SF_GetParam関数

定数名	値	用法
SLP_A_YL_EN	47	SF_GetParam関数
SLP_A_YL_POS	48	SF_GetParam関数
SLP_A_ZU_EN	49	SF_GetParam関数
SLP_A_ZU_POS	50	SF_GetParam関数
SLP_A_ZL_EN	51	SF_GetParam関数
SLP_A_ZL_POS	52	SF_GetParam関数
SLP_B_XU_EN	53	SF_GetParam関数
SLP_B_XU_POS	54	SF_GetParam関数
SLP_B_XL_EN	55	SF_GetParam関数
SLP_B_XL_POS	56	SF_GetParam関数
SLP_B_YU_EN	57	SF_GetParam関数
SLP_B_YU_POS	58	SF_GetParam関数
SLP_B_YL_EN	59	SF_GetParam関数
SLP_B_YL_POS	60	SF_GetParam関数
SLP_B_ZU_EN	61	SF_GetParam関数
SLP_B_ZU_POS	62	SF_GetParam関数
SLP_B_ZL_EN	63	SF_GetParam関数
SLP_B_ZL_POS	64	SF_GetParam関数
SLP_C_XU_EN	65	SF_GetParam関数
SLP_C_XU_POS	66	SF_GetParam関数
SLP_C_XL_EN	67	SF_GetParam関数
SLP_C_XL_POS	68	SF_GetParam関数
SLP_C_YU_EN	69	SF_GetParam関数
SLP_C_YU_POS	70	SF_GetParam関数
SLP_C_YL_EN	71	SF_GetParam関数
SLP_C_YL_POS	72	SF_GetParam関数
SLP_C_ZU_EN	73	SF_GetParam関数
SLP_C_ZU_POS	74	SF_GetParam関数

定数名	値	用法
SLP_C_ZL_EN	75	SF_GetParam関数
SLP_C_ZL_POS	76	SF_GetParam関数
SLP_J2_MON_RAD	77	SF_GetParam関数
SLP_J3_MON_RAD	78	SF_GetParam関数
SLP_J5_MON_RAD	79	SF_GetParam関数
SLP_J6_MON_RAD	80	SF_GetParam関数
SLP_J1_RANGE_MAX	81	SF_GetParam関数
SLP_J1_RANGE_MIN	82	SF_GetParam関数
SLP_J2_RANGE_MAX	83	SF_GetParam関数
SLP_J2_RANGE_MIN	84	SF_GetParam関数
SLP_J3_RANGE_MAX	85	SF_GetParam関数
SLP_J3_RANGE_MIN	86	SF_GetParam関数
SLP_J4_RANGE_MAX	87	SF_GetParam関数
SLP_J4_RANGE_MIN	88	SF_GetParam関数
SLP_J5_RANGE_MAX	89	SF_GetParam関数
SLP_J5_RANGE_MIN	90	SF_GetParam関数
SLP_J6_RANGE_MAX	91	SF_GetParam関数
SLP_J6_RANGE_MIN	92	SF_GetParam関数
SIN_1_SLS_1_EN	93	SF_GetParam関数
SIN_1_SLS_2_EN	94	SF_GetParam関数
SIN_1_SLS_3_EN	95	SF_GetParam関数
SIN_1_SLP_A_EN	96	SF_GetParam関数
SIN_1_SLP_B_EN	97	SF_GetParam関数
SIN_1_SLP_C_EN	98	SF_GetParam関数
SIN_1_SG_EN	99	SF_GetParam関数
SIN_1_ESTOP_EN	100	SF_GetParam関数
SIN_2_SLS_1_EN	101	SF_GetParam関数
SIN_2_SLS_2_EN	102	SF_GetParam関数

定数名	値	用法
SIN_2_SLS_3_EN	103	SF_GetParam関数
SIN_2_SLP_A_EN	104	SF_GetParam関数
SIN_2_SLP_B_EN	105	SF_GetParam関数
SIN_2_SLP_C_EN	106	SF_GetParam関数
SIN_2_SG_EN	107	SF_GetParam関数
SIN_2_ESTOP_EN	108	SF_GetParam関数
SIN_3_SLS_1_EN	109	SF_GetParam関数
SIN_3_SLS_2_EN	110	SF_GetParam関数
SIN_3_SLS_3_EN	111	SF_GetParam関数
SIN_3_SLP_A_EN	112	SF_GetParam関数
SIN_3_SLP_B_EN	113	SF_GetParam関数
SIN_3_SLP_C_EN	114	SF_GetParam関数
SIN_3_SG_EN	115	SF_GetParam関数
SIN_3_ESTOP_EN	116	SF_GetParam関数
SIN_4_SLS_1_EN	117	SF_GetParam関数
SIN_4_SLS_2_EN	118	SF_GetParam関数
SIN_4_SLS_3_EN	119	SF_GetParam関数
SIN_4_SLP_A_EN	120	SF_GetParam関数
SIN_4_SLP_B_EN	121	SF_GetParam関数
SIN_4_SLP_C_EN	122	SF_GetParam関数
SIN_4_SG_EN	123	SF_GetParam関数
SIN_4_ESTOP_EN	124	SF_GetParam関数
SIN_5_SLS_1_EN	125	SF_GetParam関数
SIN_5_SLS_2_EN	126	SF_GetParam関数
SIN_5_SLS_3_EN	127	SF_GetParam関数
SIN_5_SLP_A_EN	128	SF_GetParam関数
SIN_5_SLP_B_EN	129	SF_GetParam関数
SIN_5_SLP_C_EN	130	SF_GetParam関数

定数名	値	用法
SIN_5_SG_EN	131	SF_GetParam関数
SIN_5_ESTOP_EN	132	SF_GetParam関数
SOUT_1_STO	133	SF_GetParam関数
SOUT_1_SLS_1	134	SF_GetParam関数
SOUT_1_SLS_2	135	SF_GetParam関数
SOUT_1_SLS_3	136	SF_GetParam関数
SOUT_1_SLS_T2	137	SF_GetParam関数
SOUT_1_SLS_T	138	SF_GetParam関数
SOUT_1_SLP_A	139	SF_GetParam関数
SOUT_1_SLP_B	140	SF_GetParam関数
SOUT_1_SLP_C	141	SF_GetParam関数
SOUT_1_EP_RC	142	SF_GetParam関数
SOUT_1_EP_TP	143	SF_GetParam関数
SOUT_1_EN_SW	144	SF_GetParam関数
SOUT_2_STO	145	SF_GetParam関数
SOUT_2_SLS_1	146	SF_GetParam関数
SOUT_2_SLS_2	147	SF_GetParam関数
SOUT_2_SLS_3	148	SF_GetParam関数
SOUT_2_SLS_T2	149	SF_GetParam関数
SOUT_2_SLS_T	150	SF_GetParam関数
SOUT_2_SLP_A	151	SF_GetParam関数
SOUT_2_SLP_B	152	SF_GetParam関数
SOUT_2_SLP_C	153	SF_GetParam関数
SOUT_2_EP_RC	154	SF_GetParam関数
SOUT_2_EP_TP	155	SF_GetParam関数
SOUT_2_EN_SW	156	SF_GetParam関数
SOUT_3_STO	157	SF_GetParam関数
SOUT_3_SLS_1	158	SF_GetParam関数

定数名	値	用法
SOUT_3_SLS_2	159	SF_GetParam関数
SOUT_3_SLS_3	160	SF_GetParam関数
SOUT_3_SLS_T2	161	SF_GetParam関数
SOUT_3_SLS_T	162	SF_GetParam関数
SOUT_3_SLP_A	163	SF_GetParam関数
SOUT_3_SLP_B	164	SF_GetParam関数
SOUT_3_SLP_C	165	SF_GetParam関数
SOUT_3_EP_RC	166	SF_GetParam関数
SOUT_3_EP_TP	167	SF_GetParam関数
SOUT_3_EN_SW	168	SF_GetParam関数
POS_ROT_U	169	SF_GetParam関数
POS_ROT_V	170	SF_GetParam関数
POS_ROT_W	171	SF_GetParam関数
POS_OFS_X	172	SF_GetParam関数
POS_OFS_Y	173	SF_GetParam関数
POS_OFS_Z	174	SF_GetParam関数
SF_TOOLVERSION	1	SF_GetParam\$関数
SF_CHECKSUM	2	SF_GetParam\$関数
SF_LAST_MODIFIED	3	SF_GetParam\$関数
SF_ROBOT_MODEL_NAME	4	SF_GetParam\$関数
SF_ROBOT_CHECKSUM	5	SF_GetParam\$関数
SF_HOFS	6	SF_GetParam\$関数
SF_HOFS_LAST_MODIFIED	7	SF_GetParam\$関数
SLS_1	1	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_2	2	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_3	3	SF_LimitSpeedS, SF_LimitSpeedSEnable

定数名	値	用法
SLS_T	9	SF_LimitSpeedS, SF_LimitSpeedSEnable
SLS_T2	10	SF_LimitSpeedS, SF_LimitSpeedSEnable
STM_H_MODE_FASTEST	1	STM関数, AutoSTM
STM_OFF	0	STM関数, AutoSTM